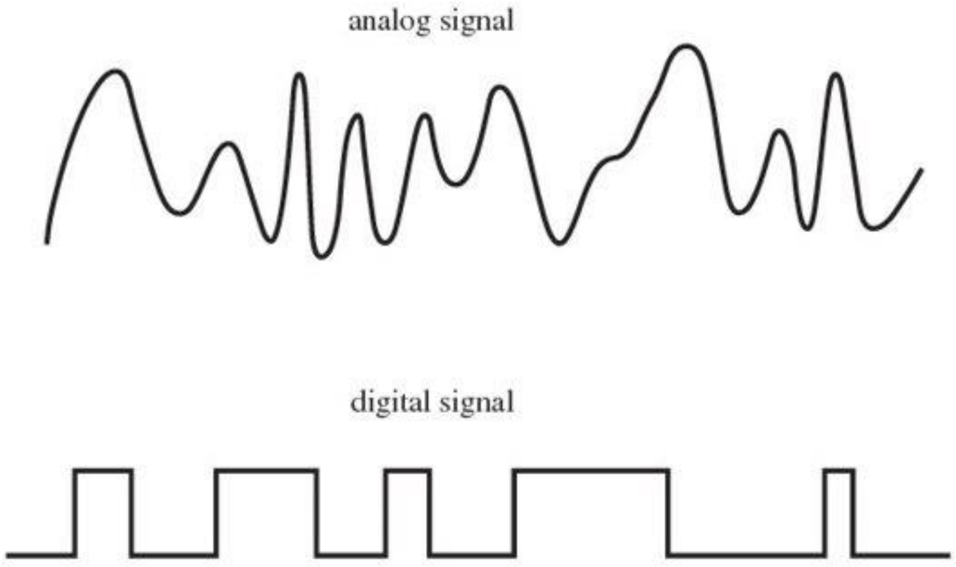
جامعة بيرزيت
BIRZEIT UNIVERSITY

ANALOG TO DIGITAL CONVERTER (ADC)

Shadi Daana

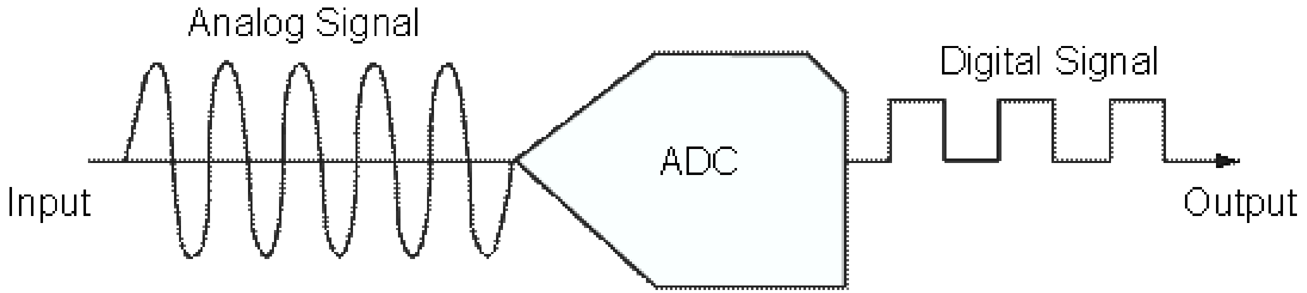
Basics of analog and digital signals

- In the real world, **analog signals** are **continuous**, time-varying signals that represent information using a continuous range of values. These types of signals can come from sound, light, temperature, and motion
- **Digital signals** are **discrete**, non-continuous signals that represent information using a finite set of values. These signals are typically binary, consisting of discrete 0s and 1s.

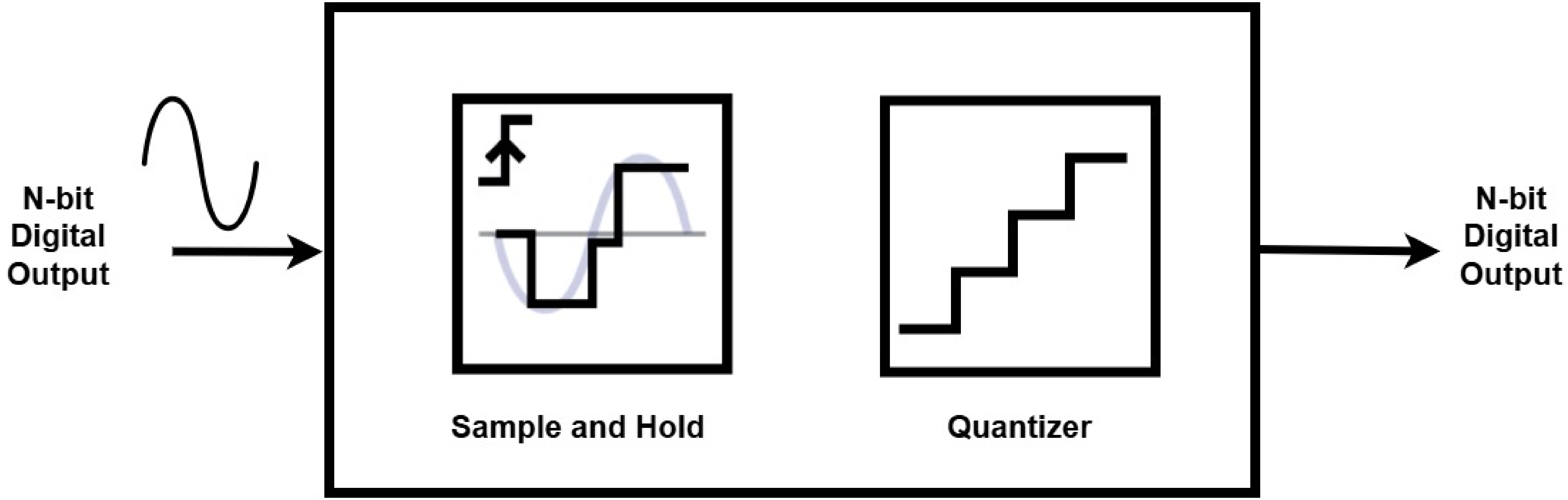


Basics of analog and digital signals

- To process a continuous signal in a computer or other digital system, you must first translate it into **digital representation**.
- **Analog-to-Digital converters (ADC)** translate analog signals, real world signals like temperature, pressure, voltage, current, distance, or light intensity, into a digital representation of that signal.
- This digital representation can then be processed, manipulated, computed, transmitted or stored.



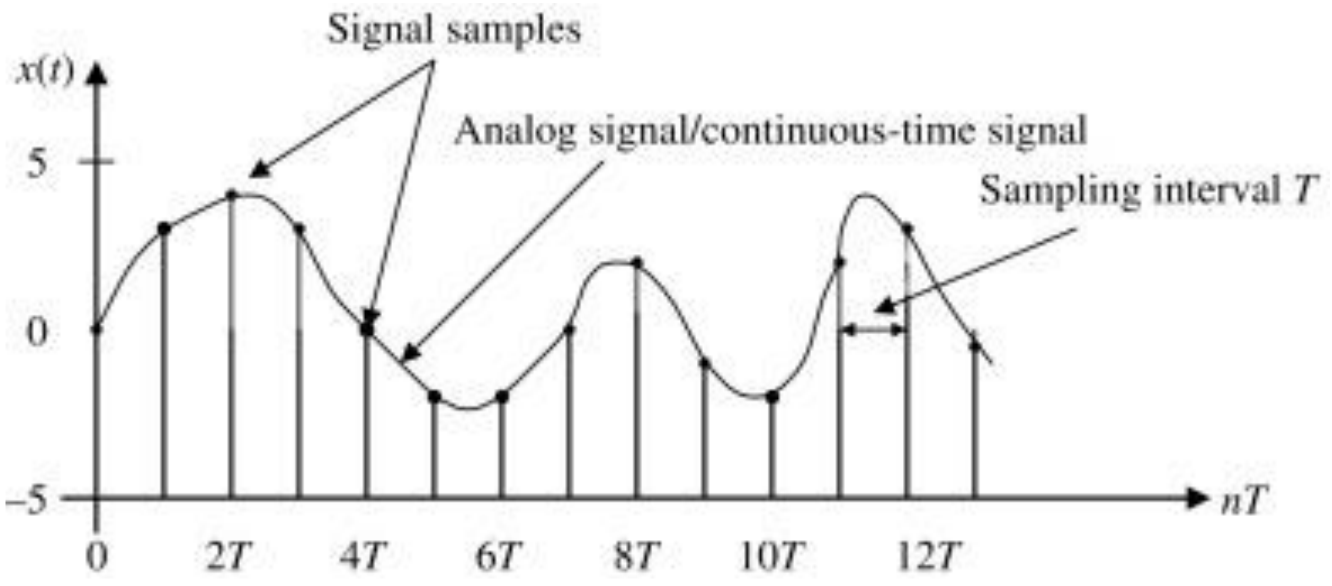
ADC Conversion Process



ADC Block Diagram

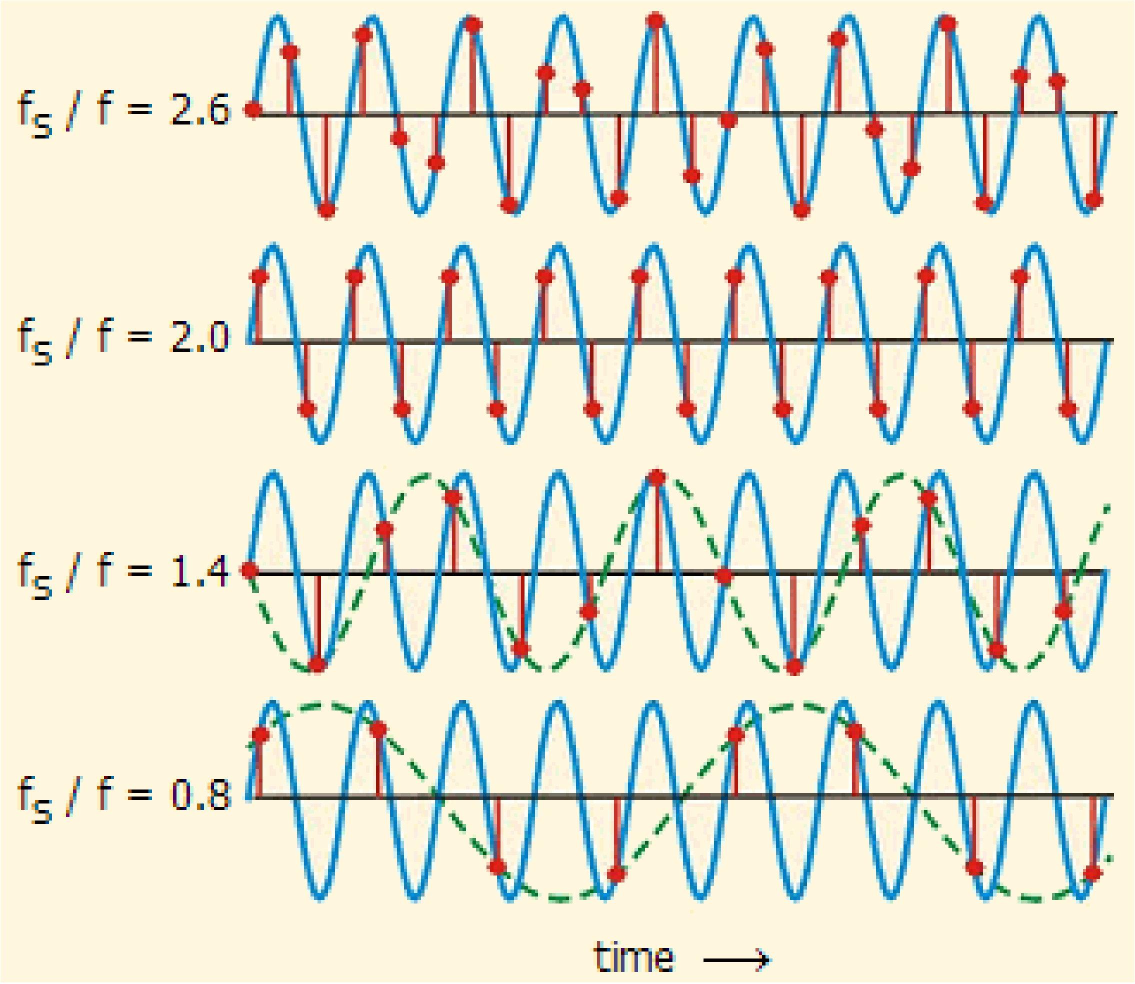
ADC Sampling

- To process an analog signal in a digital system, it needs to be converted into a series of discrete values.
- **Sampling** in ADC refers to the process of taking **discrete samples** of an analog signal at regular intervals **or sampling rate**.



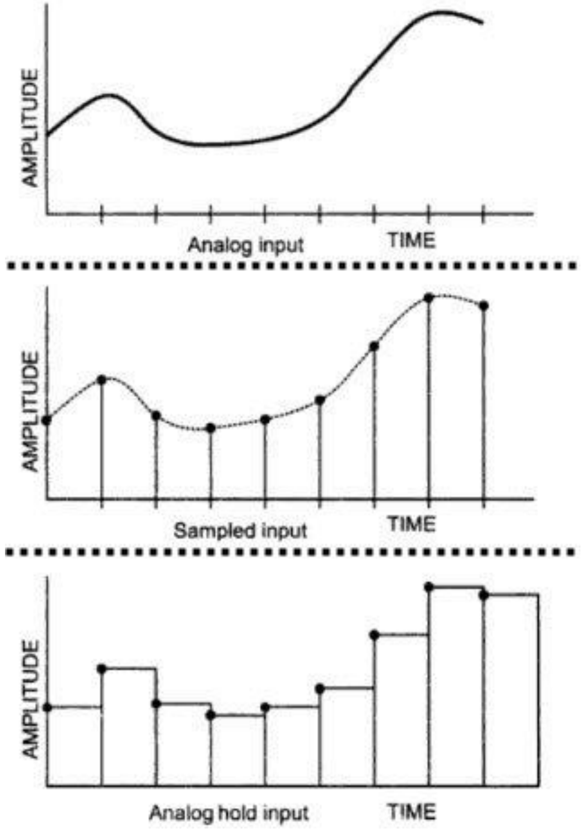
What sampling rate do we need?

- The **Nyquist-Shannon sampling theorem** dictates that the sampling rate must be at least twice the highest frequency present in the analog signal to avoid **aliasing**.
- **Aliasing** is a phenomenon where high-frequency components incorrectly appear as lower frequencies

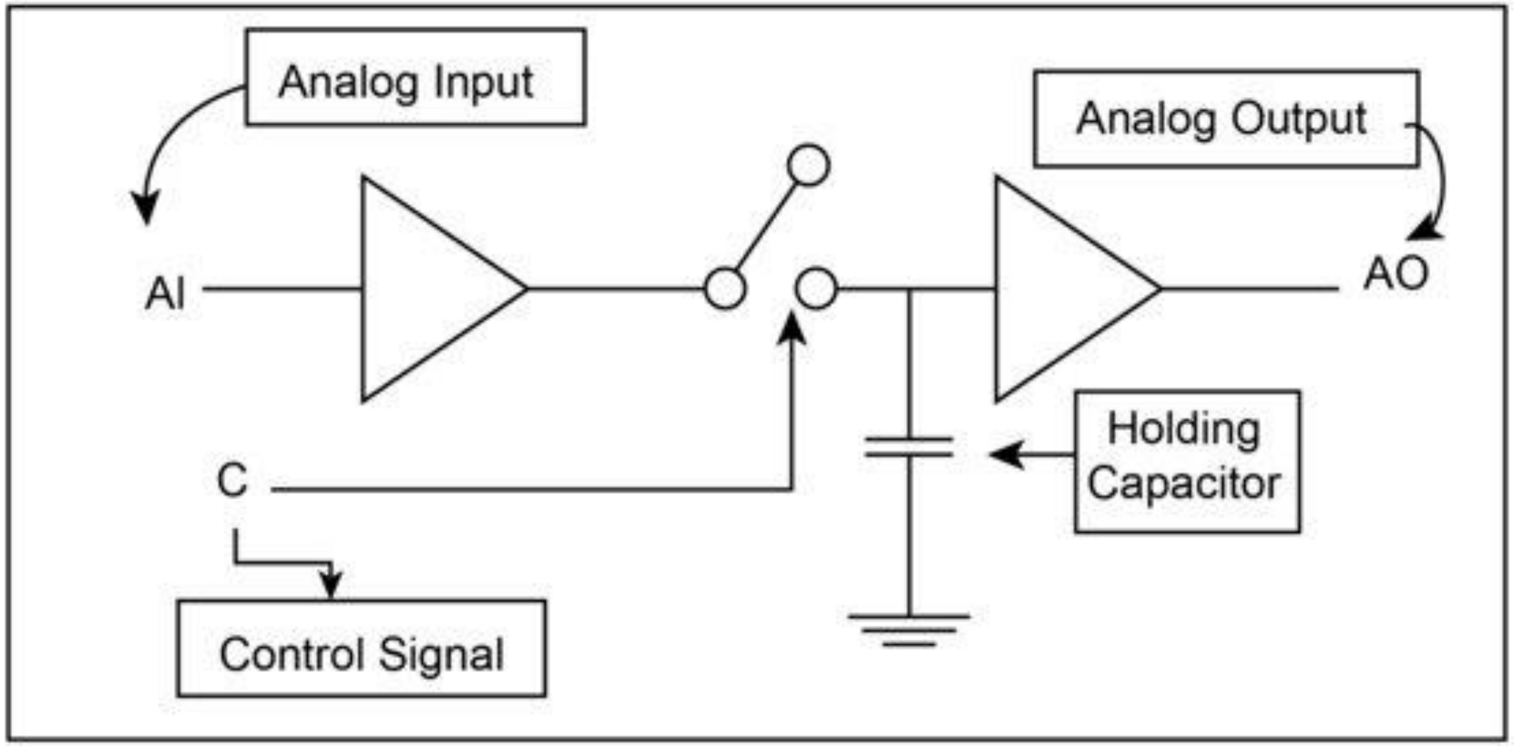


Sample and hold

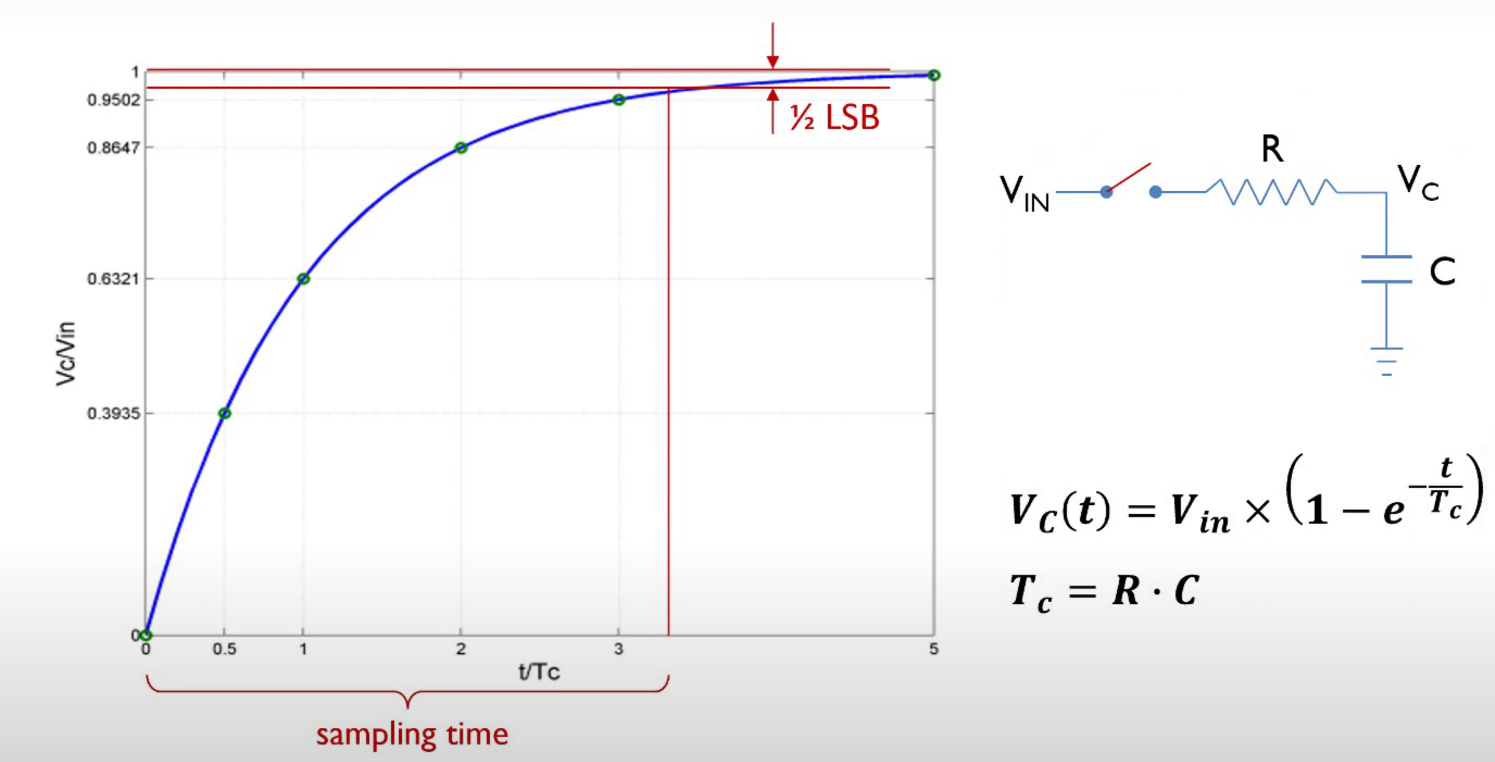
- The main function of a sample-and-hold (S/H) circuit is to take samples of its input signal and hold these samples in its output for some period of time.
- Typically, the samples are taken at uniform time intervals; thus, the sampling rate (or clock rate) of the circuit can be determined.



Sample and Hold Circuit



Determining minimum sampling time



$$V_C(t) = V_{in} \times \left(1 - e^{-\frac{t}{T_c}}\right)$$

$$T_c = R \cdot C$$

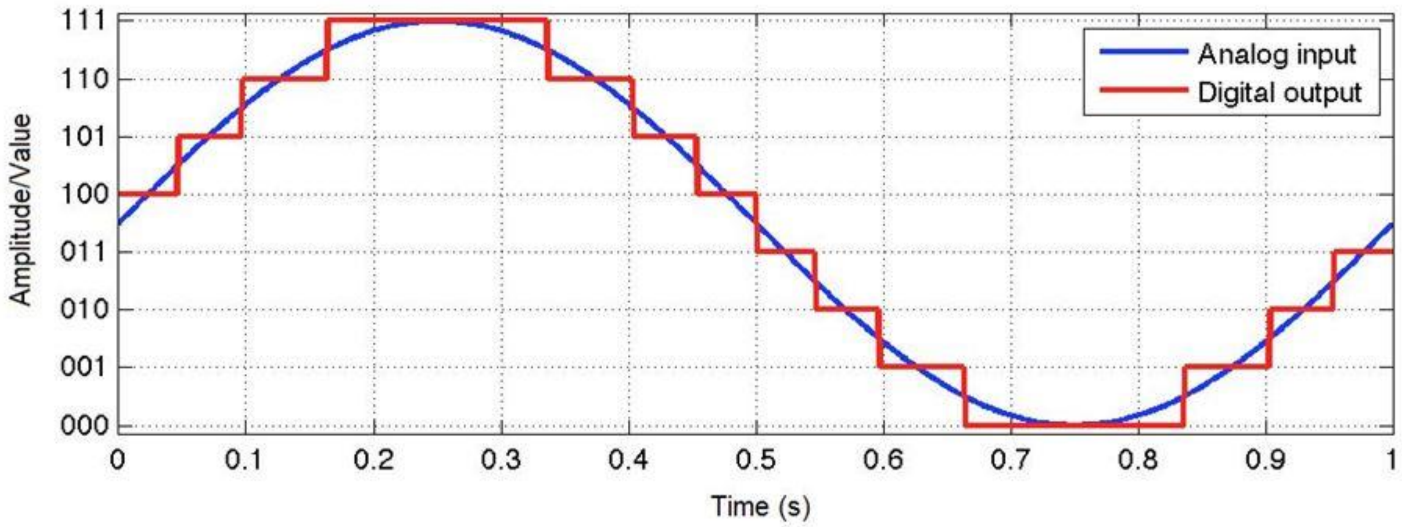
- Sampling time is software programmable
- Sampling time must be long enough to settle within ½ LSB

ADC Quantization

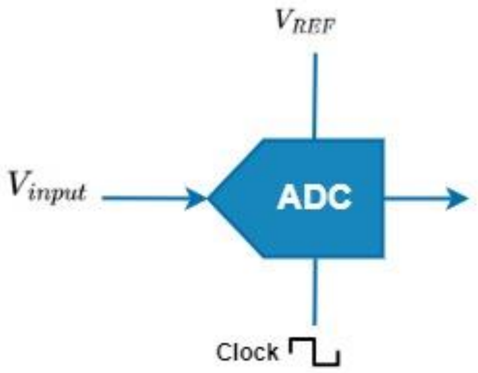
- Sampling converts a time-varying voltage signal into a discrete-time signal, a sequence of real numbers.
- Quantization replaces each real number with the closest value among a limited number of discrete levels
- The type of ADC depends on how it's performing the quantization process; it can be
  - Analog integration
  - Digital counter
  - **Successive approximation**
  - Or flash ADC .

Basics of analog and digital signals

- Basically, an analogue to digital converter takes a **snapshot** of an analogue voltage at **one instant in time** and produces a **digital output** code which represents this analogue voltage.
- **Resolution of ADC**: number of binary bits in ADC output used to represent this analog voltage value



ADC resolution



$$ADC\ \ Output\ = round\left((2^N - 1)\frac{V_{input}}{V_{REF}}\right)$$

N : Resolution

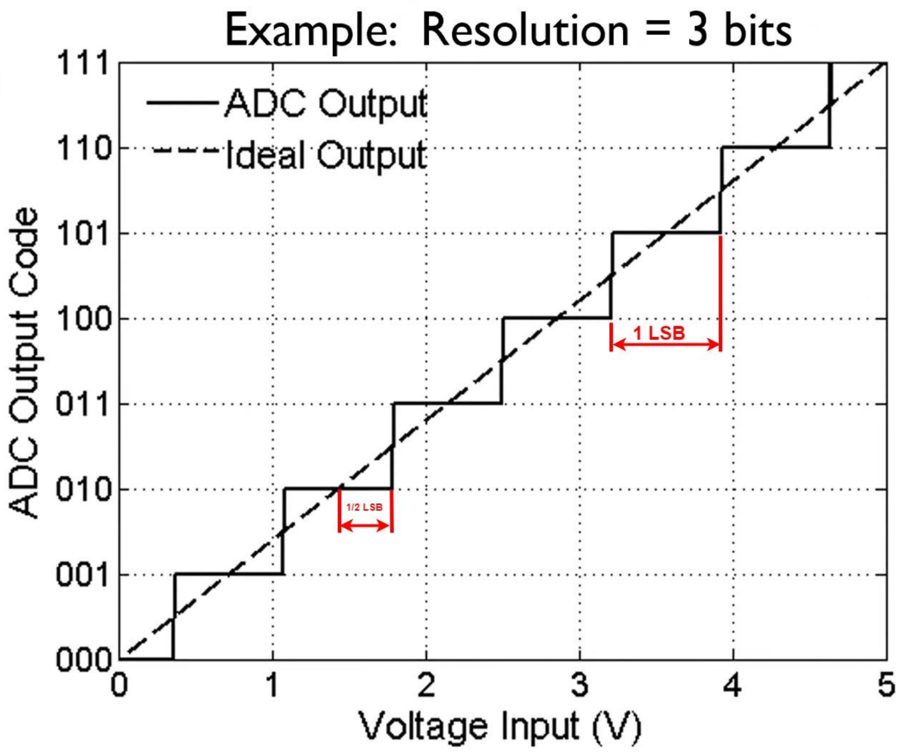
V_{REF} : The maximum input voltage that can be converted by ADC

Example: for 12-bit ADC, V_{input}= 1, V_{REF} = 3.3

$$ADC\ \ Output\ = round\left((2^{12} - 1)\frac{1}{3.3}\right) = round(1240.9) = 1241 = \text{0x4D9}$$

ADC quantization error

- Analog to digital conversion **destroys information**: we convert a range of input voltages to a single digital value



Least significant bit (LSB) voltage: $$LSB = \frac{V_{REF}}{2^N - 1}$$

Maximum quantization error = $\pm\ 1/2\ LSB$

Example: 3-Bit ADC and input range [0, 3.3V]

Maximum quantization error = $\pm\ \frac{1}{2} \times \frac{3.3}{2^3 - 1}$

$= \pm\ 0.236$

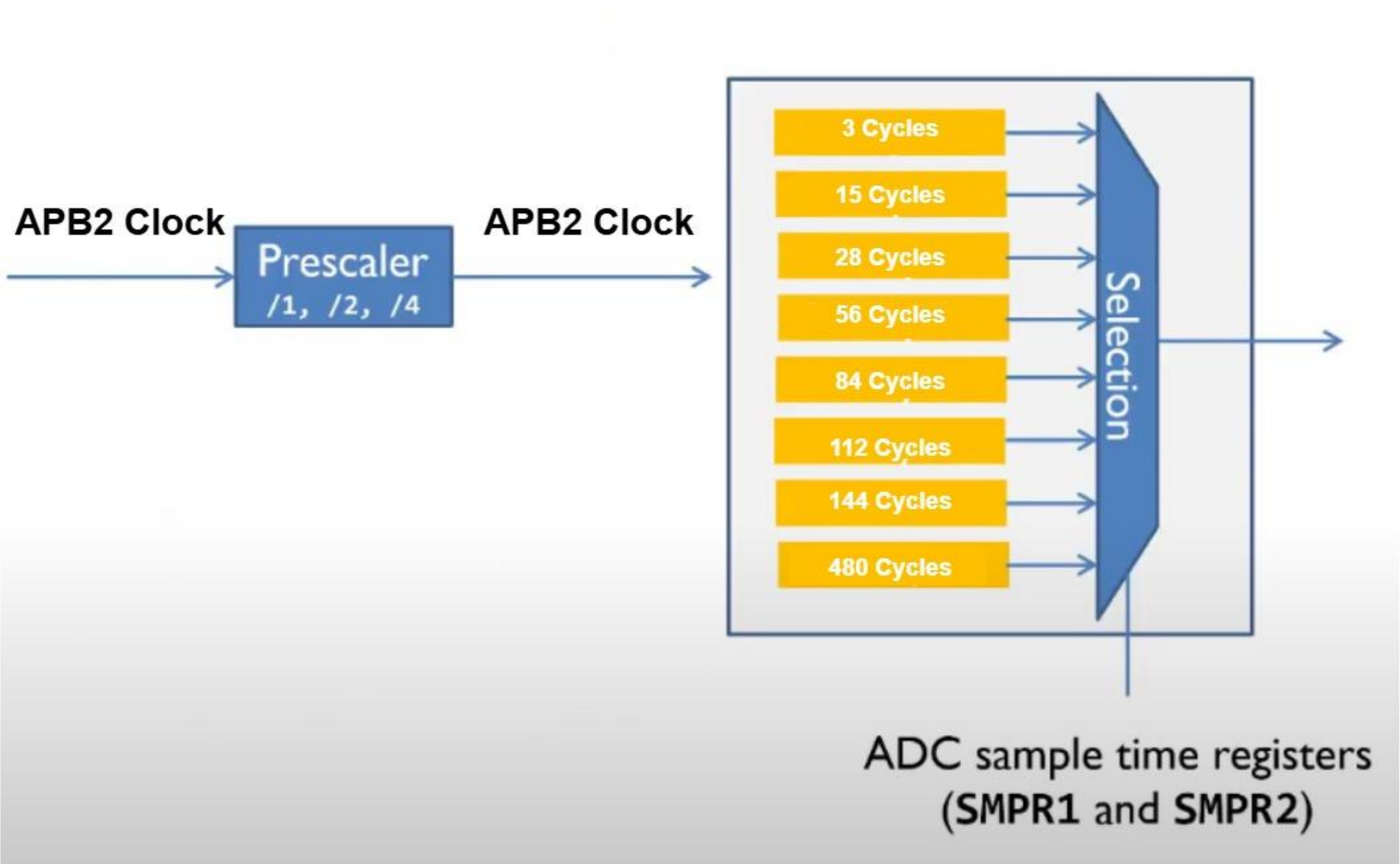
STM32 ADC Peripheral

- The ADC in the STM32F4 microcontroller is a peripheral that converts analog signals into digital values, allowing the microcontroller to process real-world sensor data
- It utilizes **successive approximation** approach to perform analog to digital conversions.
- It has up to 19 multiplexed channels allowing it to measure signals from 16 external sources, two internal sources, and the VBAT channel.
- The resolution of the ADC in the STM32F4 microcontroller is **configurable**, and it can be 12-bit, 10-bit, 8-bit, or 6-bit.
- The **sample time** in STM32 microcontroller is **programmable**

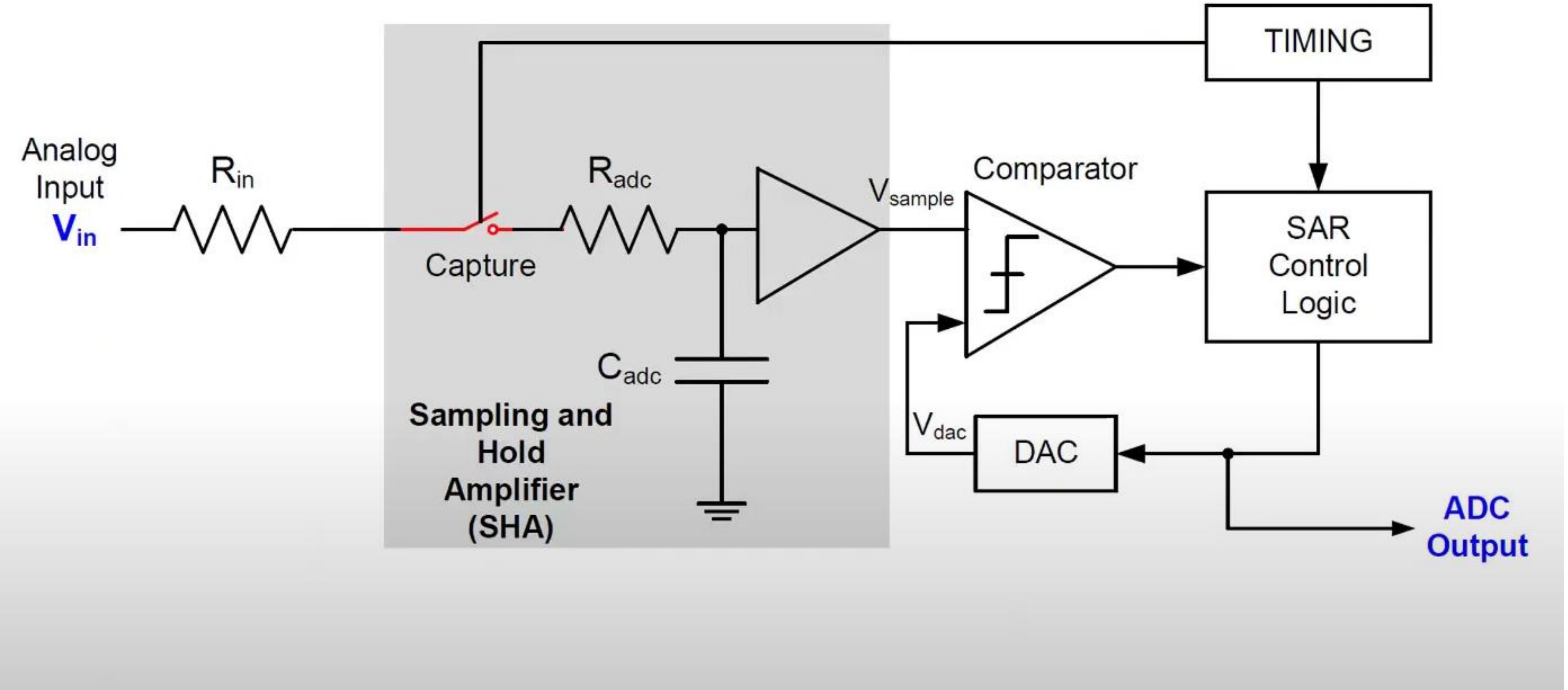
STM32 ADC Peripheral

- The STM32F4 microcontroller has up to three ADC modules (ADC1, ADC2, and ADC3) connected to the APB2 bus.
- This means that the ADCs Receive their clock source through this bus. (**check Functional overview section in the datasheet**)
- The clock from the APB2 bus might be too fast for the ADCs to operate correctly.
- To adjust the clock speed to a suitable value for the ADC, there is an internal prescaler dedicated to the ADCs
- This prescaler is used to divide the APB2 clock down to a frequency that is compatible with the ADC that does not exceed its maximum operating frequency
- (Typically up to 36 MHz for the STM32F40xx and STM32F41xx ADCs). (check ADC characteristics table in the datasheet)

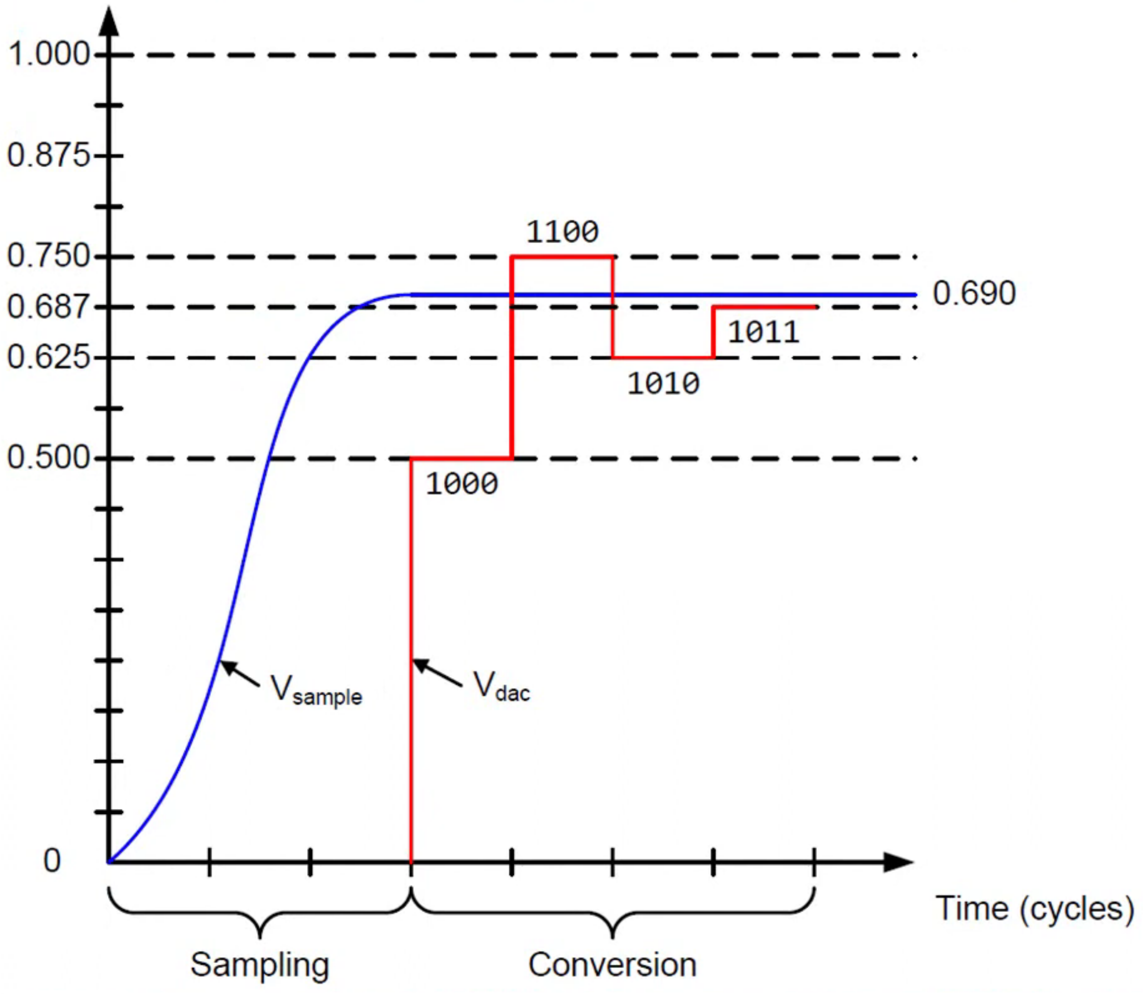
Programming ADC sampling time



Successive approximation (SAR)



Successive approximation (SAR) ADC



N cycles for N-bit ADC

- Binary search algorithm is used to gradually approach the input voltage
- Settle into ± ½ LSB bound within the time allowed

ADC conversion time

$$T_{ADC} = T_{Sampling} + T_{Conversion}$$

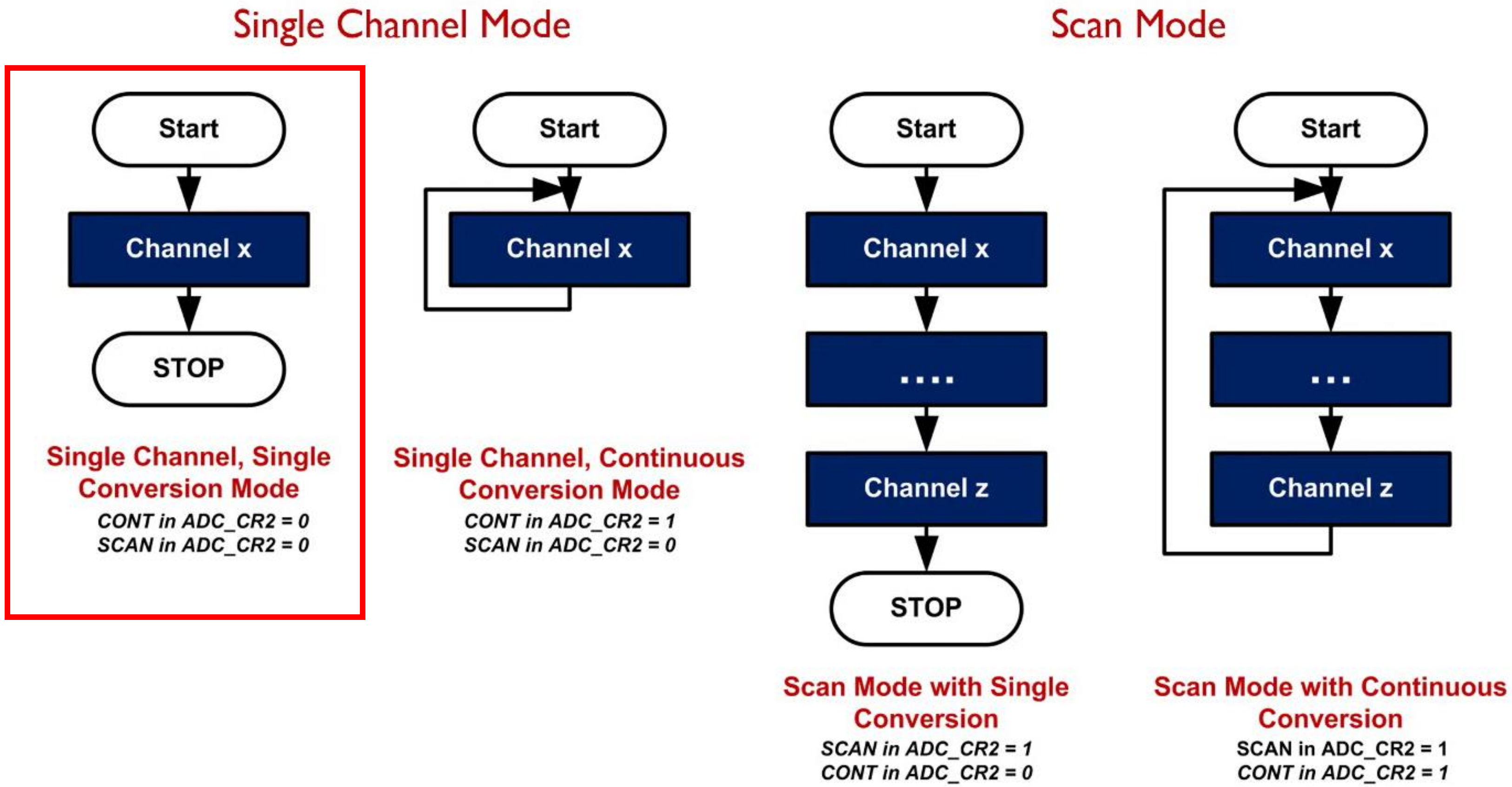
- Suppose ADC clock = 16 MHz and sampling time = 4 cycles
- For 12-bit ADC

$$T_{ADC} = 4 + 12 = 16 \text{ cycles} = 1\ \mu s$$

- For 6-bit ADC

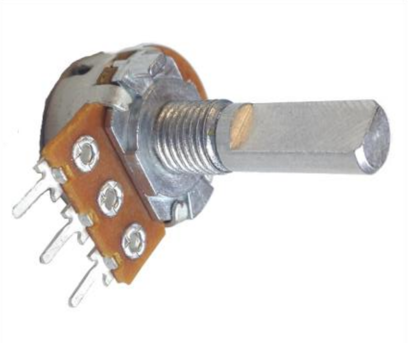
$$T_{ADC} = 4 + 6 = 10 \text{ cycles} = 0.625\ \mu s$$

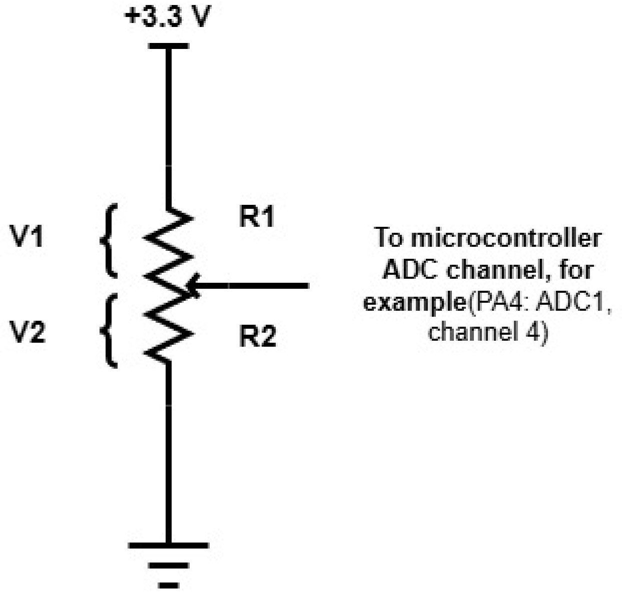
ADC conversion modes



Example: Read ADC Value From Potentiometer

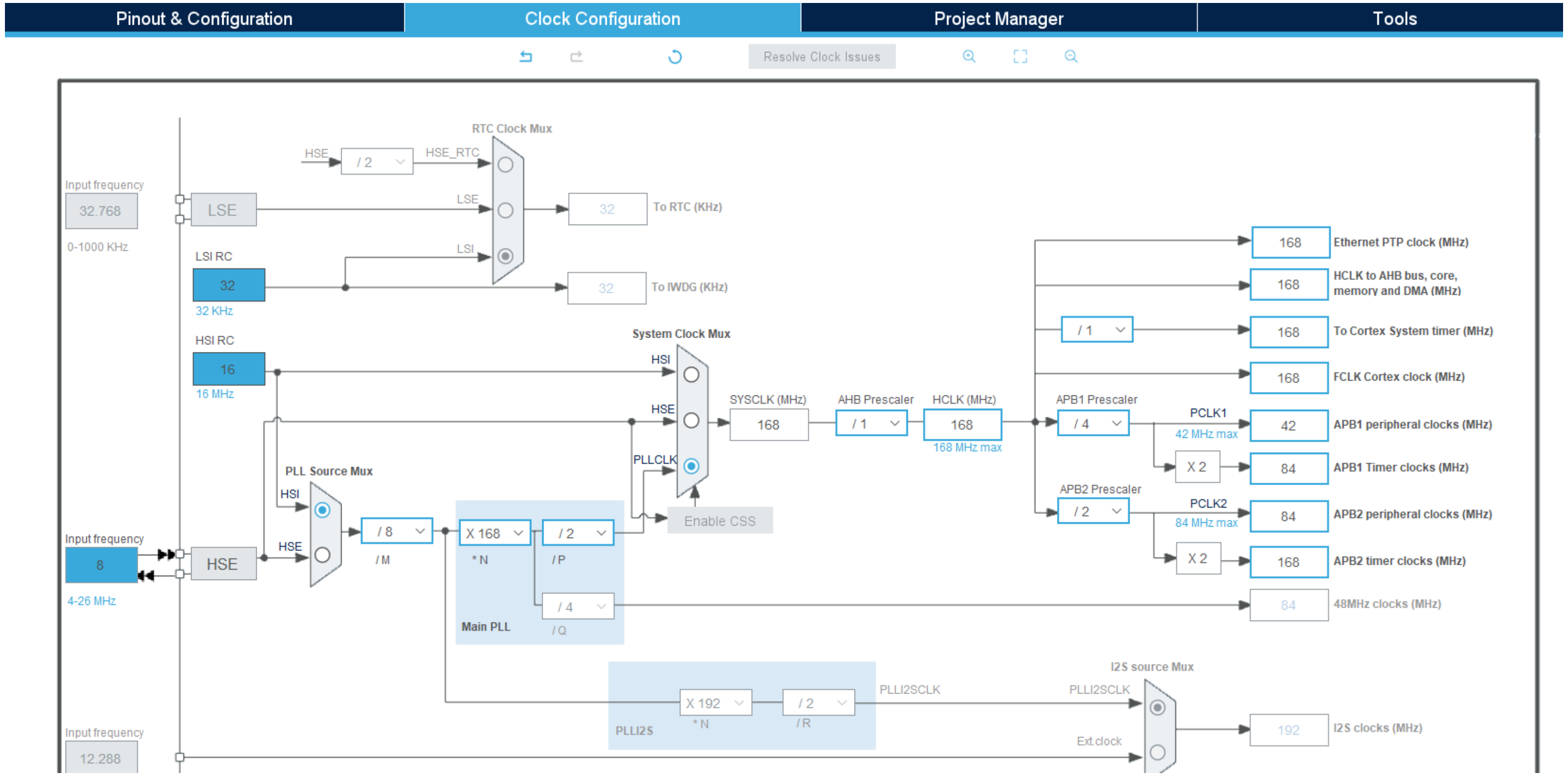
- The potentiometer is a variable voltage divider based on the amount of shaft rotation
- The voltage is divided into two portions, V1 and V2, with V2 being the voltage presented to the microcontroller’s analog input.
- You need to connect the middle pin of the potentiometer to an ADC channel of the microcontroller
- In this example the middle pin is connected to channel 4 of ADC1, which is PA4
- ***Note: the Vref in STM32 microcontroller is 3.3, it cannot handle more than 3.3 voltage input***





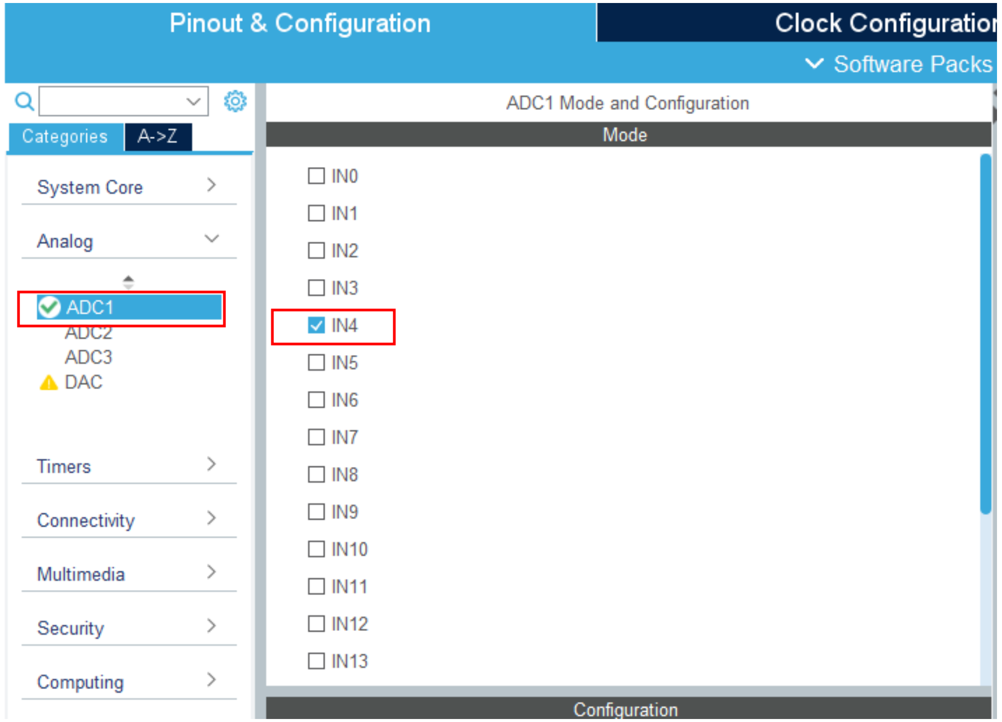
Example: Read ADC Value From Potentiometer

- Open STM32CubeIDE and create a new STM32 project for your STM32F4 microcontroller.
- Set frequencies for the system clock (SYSCLK), (APB) clocks, and AHB bus (for example: Select HSI as a clock source to PLL, and set APB2 clock to 84 MHz)

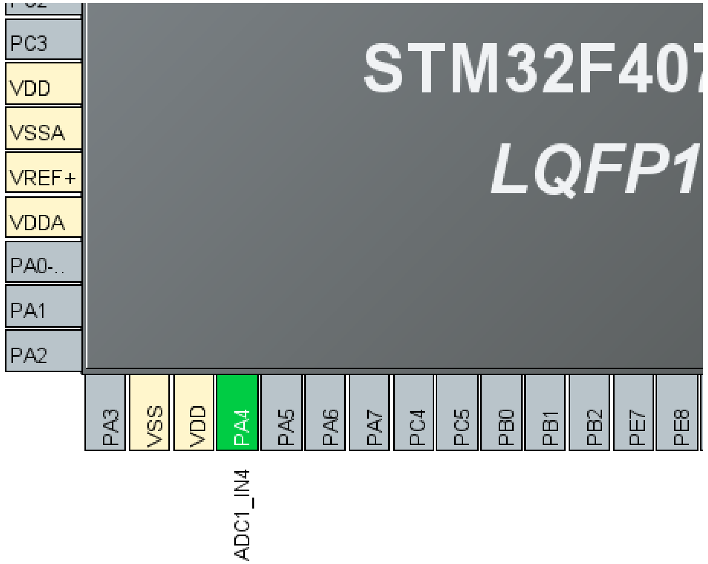


Example: Read ADC Value From Potentiometer

- Enable ADC1 channel 4 as shown in the following figure.

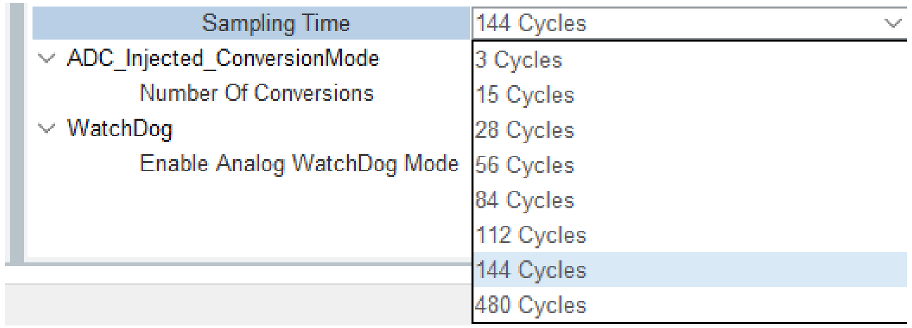


- PA4 will be activated when you select ADC1, Channel 4

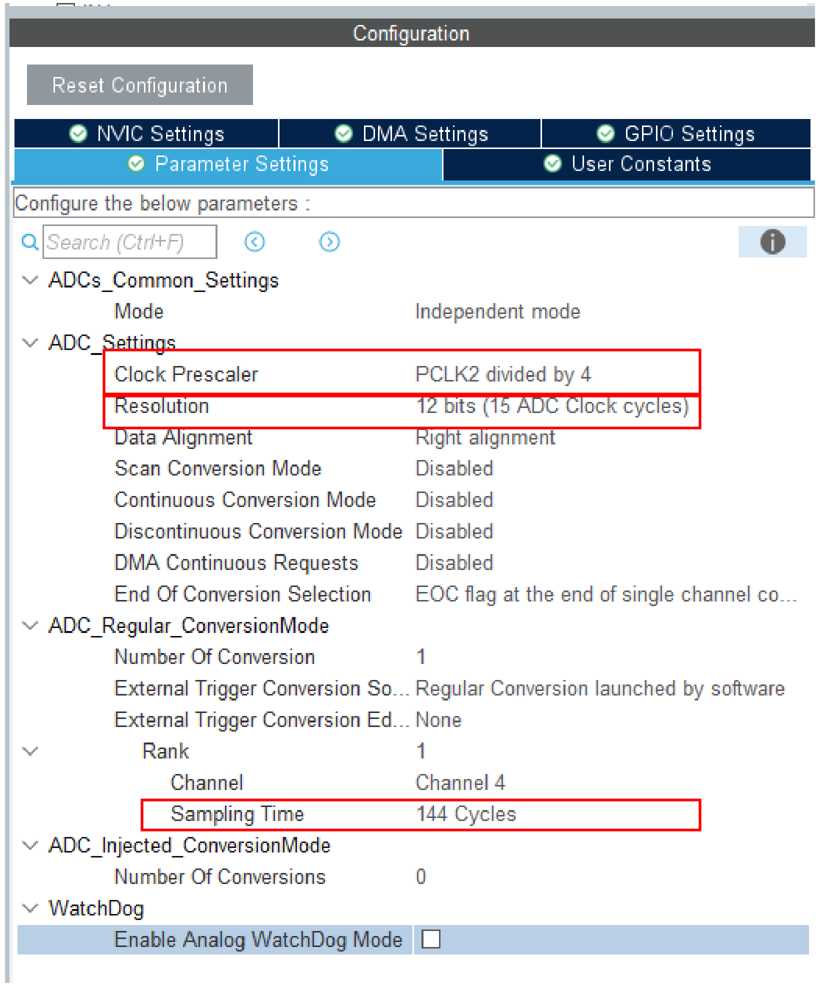


Example: Read ADC Value From Potentiometer

- The frequency of the clock to ADC module is determined by dividing the **APB2 clock frequency** by the **clock prescaler**
- In this example, the prescaler is selected to be 4.
- Thus, the ADC clock is (84Mhz/4 = 21MHz).
- You can select the sampling time from drop menu



- **Can you calculate the total ADC time?**



Example: Read ADC Value From Potentiometer

- Click on the "Project" menu and select "Generate Code" to generate the initialization code based on your configuration.
- Inside the main.c, you can write the code to read the ADC value
- Since we just covered **single channel, single conversion mode,** every time we need to read an ADC value, we need to
  - Tell the ADC to start the conversion

```
HAL_ADC_Start(&hadc1);
```

`hadc1` is a declaration of a structure (`ADC_HandleTypeDef`) that is automatically generated by the Cubemx tool. It encapsulates all the information and configurations related to a specific ADC instance

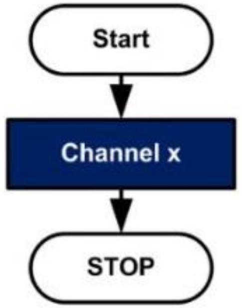
```
ADC_HandleTypeDef hadc1;
```

  - Wait until the ADC module convert the value, then read the ADC value and save it into a variable (i.e. `uint16_t adcValue;`)

```
if(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {
adc_value  = HAL_ADC_GetValue(&hadc1);// Read ADC value
}
```

  - Stop the conversion

```
HAL_ADC_Stop(&hadc1);
```



Single Channel, Single Conversion Mode

CONT in ADC_CR2 = 0
SCAN in ADC_CR2 = 0

Example: Read ADC Value From Potentiometer

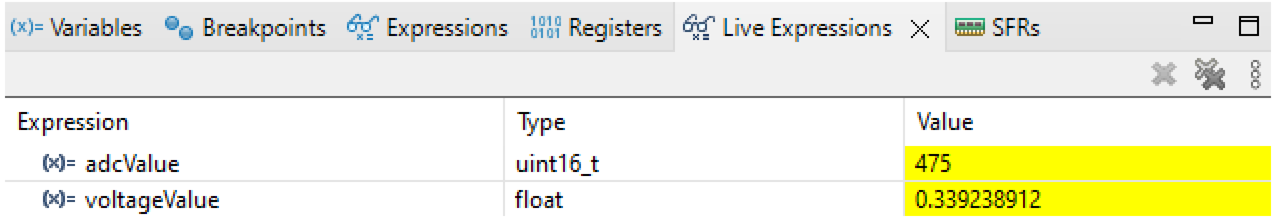
- If you have more than one ADC channel, you must select the channel from which you need to acquire data
- To simplify the process, you can create your own function for selecting the channel and getting the ADC value.
- For Example,
  - ADC_Get_Value is used to return ADC value using single channel, single conversion mode.
  - ADC_Select_Channel is used to select the channel that you will get data from.
- Do not forget to include **Function Declarations (Function Prototypes)**

```
uint32_t ADC_Get_Value(ADC_HandleTypeDef *hadc, uint32_t adc_channel) {
    ADC_Select_Channel(hadc, adc_channel);
    uint32_t adc_value;
    HAL_ADC_Start(hadc);
        if(HAL_ADC_PollForConversion(hadc, 100) == HAL_OK)
        {
            adc_value  = HAL_ADC_GetValue(hadc);
        }
        HAL_ADC_Stop(hadc);
    return adc_value;
}
```

```
void ADC_Select_Channel(ADC_HandleTypeDef *hadc, uint32_t adc_channel)
  {
    ADC_ChannelConfTypeDef sConfig = {0};
    sConfig.Channel = adc_channel;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;
    if (HAL_ADC_ConfigChannel(hadc, &sConfig) != HAL_OK) {
        Error_Handler();
    }
}
```

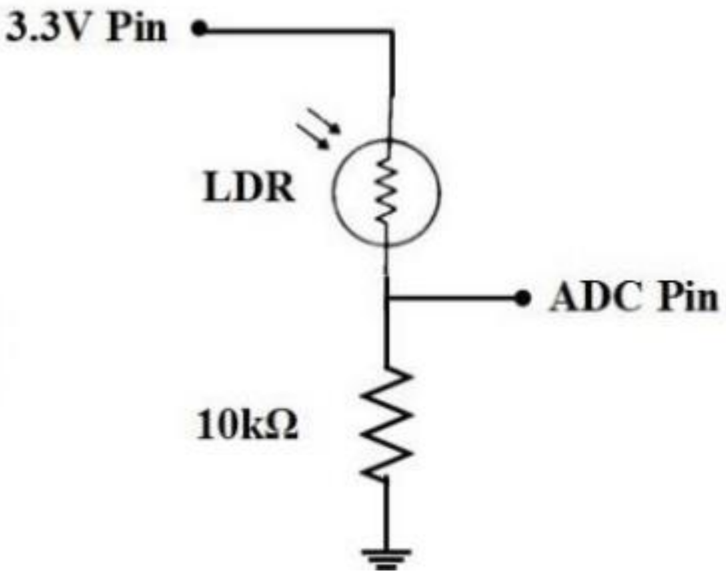
Example: Program Code

- In the debug view, you can add the variables that you want to see to the Live Expression tab.
- It will display the value based on the change of input voltage from the potentiometer



Exercise: Interfacing a Light Dependent Resistor (LDR)

- Develop firmware to read data from an LDR sensor



Questions?