Unit 4

Serial Port: USART

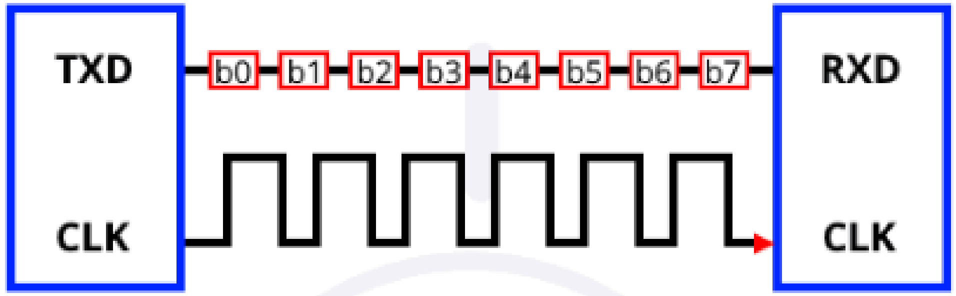
Unit 4: Syllabus

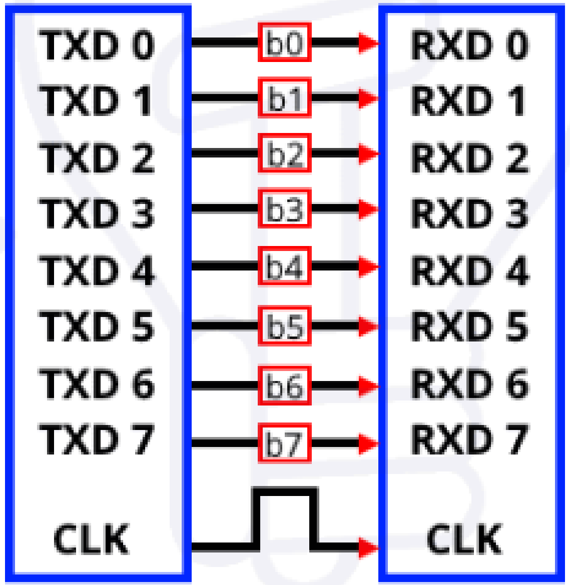
Serial Port: USART(Universal Synchronous/Asynchronous Receiver Transmitter)

Basics of serial communication (Synchronous, asynchronous), Framing, Sampling, Baud rate generation, Programming USART for character transmission, Serial Peripheral Interface, Programming SPI for data transfer

Basics of serial communication

- Serial communication is a method of transmitting data between two devices or systems one bit at a time over a single communication line.
- It's a fundamental communication method used in electronics and computer systems for connecting devices like microcontrollers, sensors, modems, and more.
- In contrast to parallel communication, where multiple bits are transmitted simultaneously on separate lines, serial communication uses a single data line to transmit data sequentially.





Note: In Asynchronous serial communication like UART, no clock signal is transmitted.

- **Simplex**

E.g: Large electronic billboards and LED displays in public spaces can be considered simplex systems.

- **Half Duplex**

E.g: Devices communicating using I2C serial communication

- **Full Duplex**

E.g: Devices communicating using SPI, UART serial communication

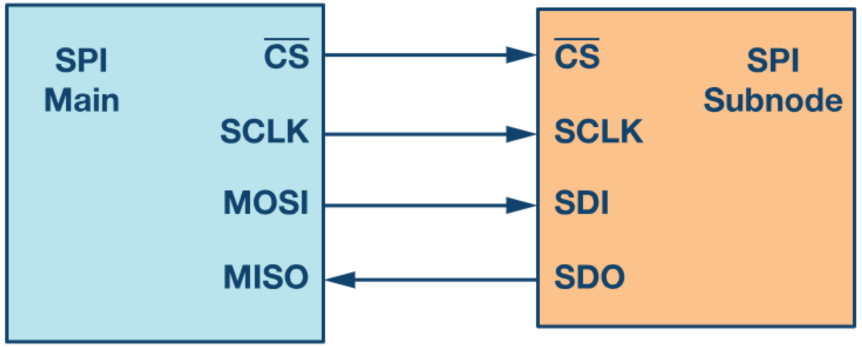
There are two main types of serial communication

- **Synchronous Serial Communication**
  - -Data is transmitted in a continuous stream along with a clock signal.
  - - The clock signal helps the receiving device synchronize with the transmitted data.
  - -This method is generally more reliable at higher speeds.

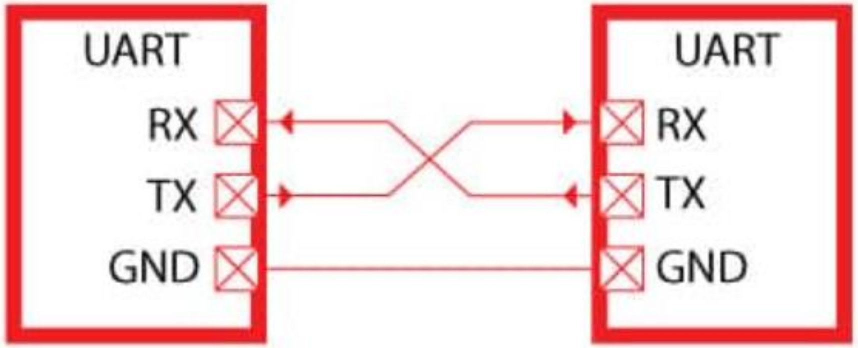
  Example: I2C,SPI

- **Asynchronous Serial Communication**
  - -Each data byte is sent with start and stop bits, but there is no continuous clock signal.
  - -This is the most common form of serial communication and is used in applications where data rates are not extremely high and the devices might not have a shared clock.

  Example: UART(Universal Asynchronous Receiver Transmitter)



Synchronous: SPI Communication to slave



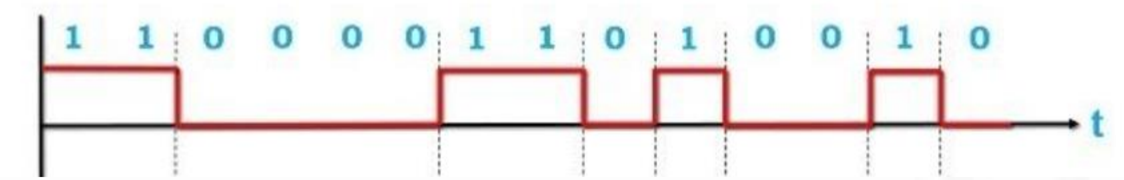
Asynchronous: UART Communication to slave

Baud rate Vs Bitrate

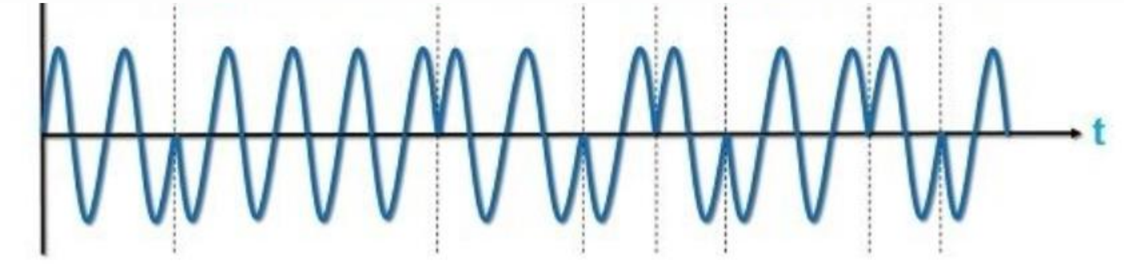
- Baud rate, also known as "symbol rate," refers to the number of signal changes (symbols) per second transmitted.
- In communication, each symbol can represent multiple bits of information depending on the modulation scheme used.

Example: Modulation scheme -> BPSK(Binary Phase Shift Keying)

Information sequence:

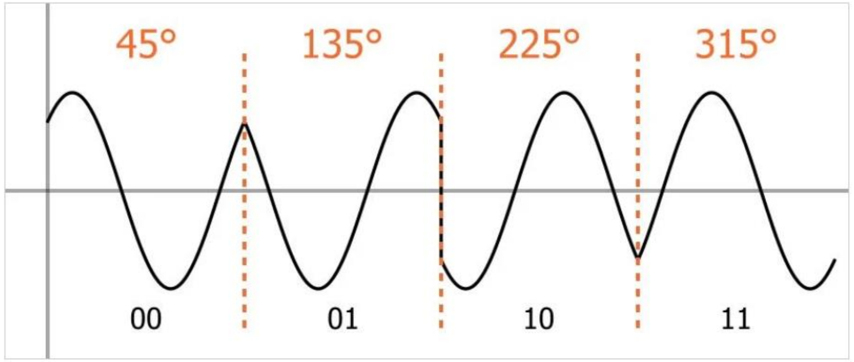


Modulated Signal:



Each symbol can represent one bit (0 or 1). So, in BPSK, the baud rate and the bit rate would be the same.

- **Example: Modulation scheme -> QPSK(Quadrature Phase Shift Keying)**



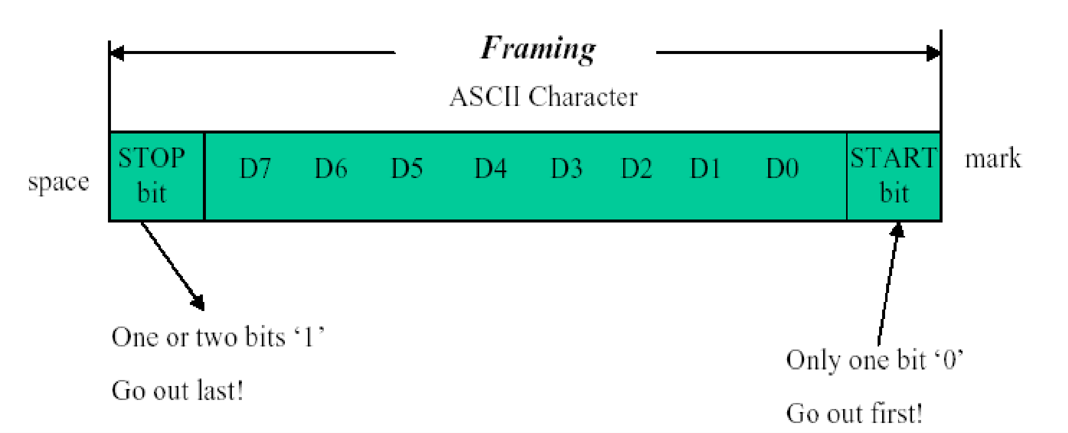
Each symbol can represent two bits (00,01,10,11). So, in QPSK, bit rate equal to twice the baud rate.

- In microcontrollers, data is transmitted in 0's and 1's only. So, no difference between bit rate and baud rate.

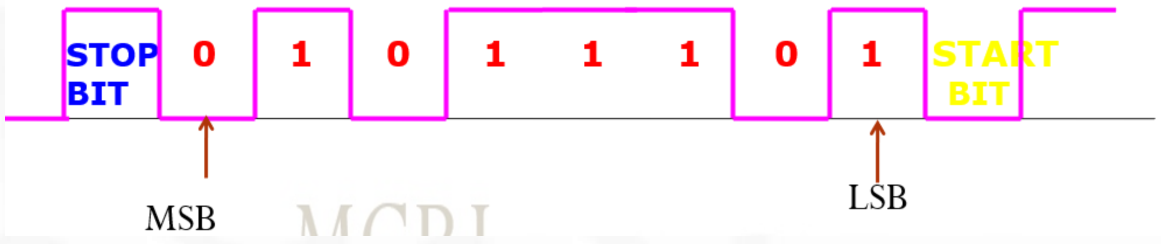
Example: 9600 baud=9600 bps(Bits per second)

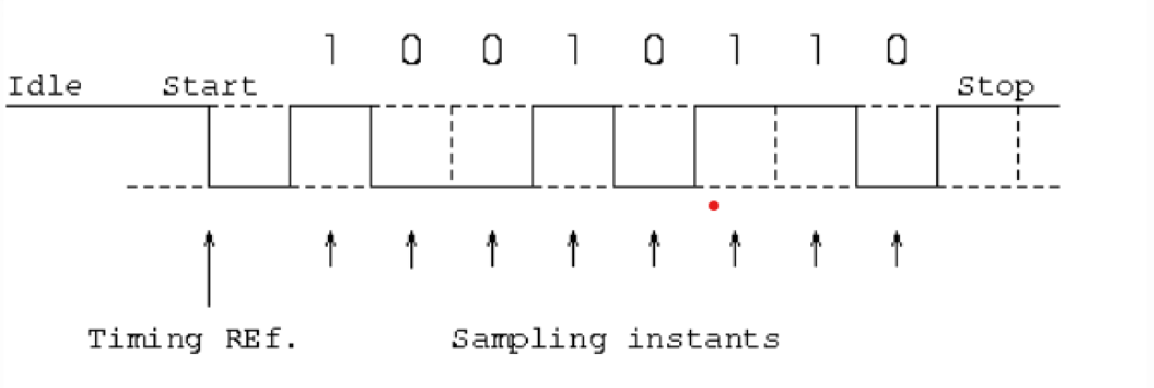
Framing in Asynchronous Communication

- Asynchronous serial communication is used for character oriented data transmission.
- Each byte is accompanied by start and stop bits before transmission and removed at the receiver side.
- Asynchronous Transmission of 5Dh.
- Receiver samples data at centre of bit intervals to decide bit transmitted.



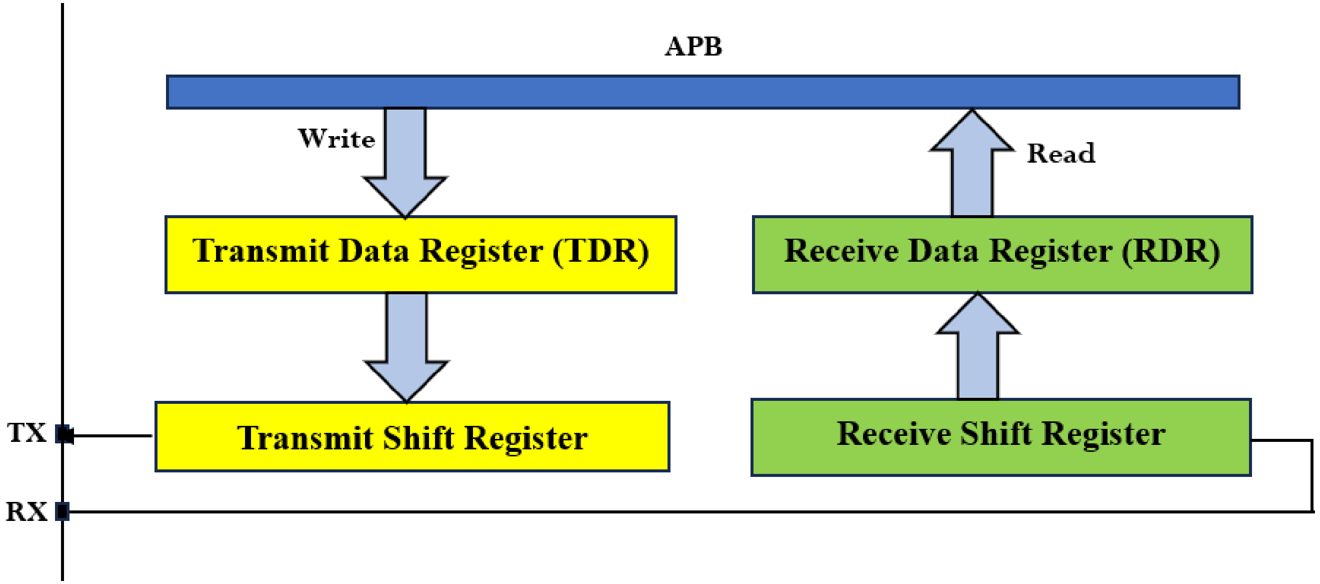
5Dh





UART Module in STM32F407VG

- It supports full duplex, asynchronous communication and synchronous half duplex communication.
- The simplified UART diagram is shown in figure.
- To send data, we simply write to the Transmit Data register (USART_TDR).The USART peripheral sends out the content of the data transmit register through the serial transmit pin (TXD).



- The received data is stored in the Receive Data register (USART_RDR).
- As two separate registers are provided, transmit and reception can happen simultaneously.
- Many registers are provided for configuration, control and status indication.
- Data is **8 or 9** bits with LSB first, **1 or 2** Stop bits indicating that the frame is complete.
- Configurable over sampling rate.

Baud rate generation

- The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - OVER8) \times USARTDIV}$$

- f_{CK} is the APB 1 (or APB2) peripheral clock depending upon USART selected(USART1,USART2,UART3,UART4,USART5,USART6).
- USARTDIV is an unsigned fixed point number that is coded on the USART_BRR(Baud Rate Register) register.
- When **OVER8=0**(a bit in USART_CR1), the fractional part is coded on 4 bits and programmed by the DIV_fraction[3:0] bits in the USART_BRR register.
- When **OVER8=1**, the fractional part is coded on 3 bits and programmed by the DIV_fraction[2:0] bits in the USART_BRR register, and bit DIV_fraction[3] must be kept cleared

USART_BRR register

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
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV

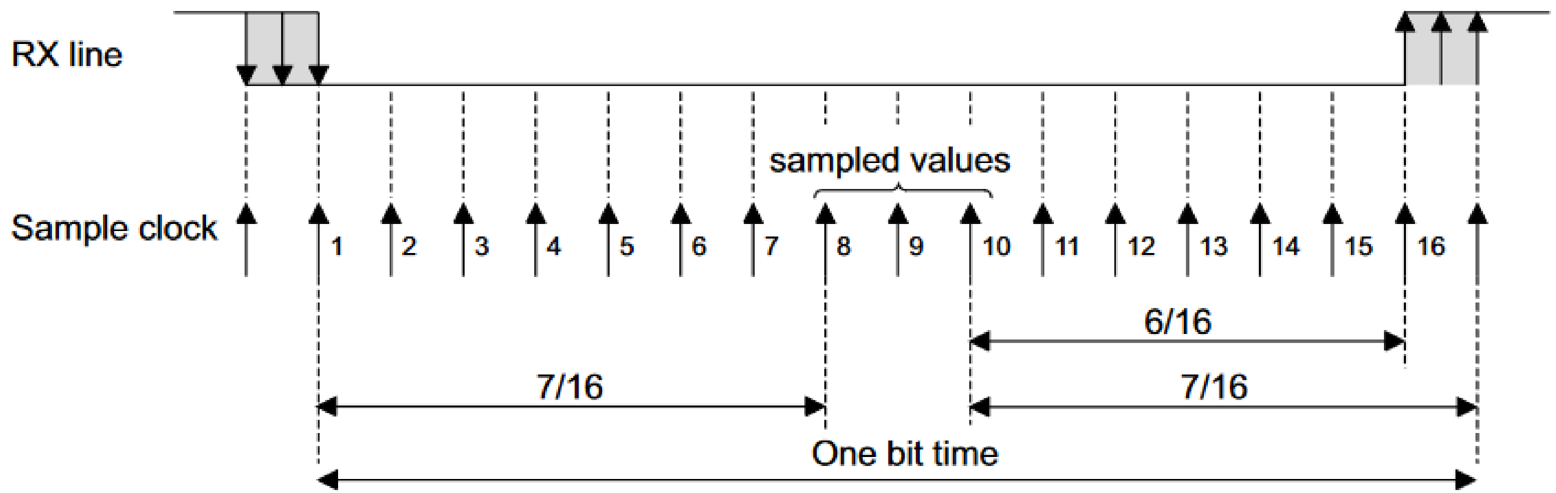
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

Oversampling (OVER bit)

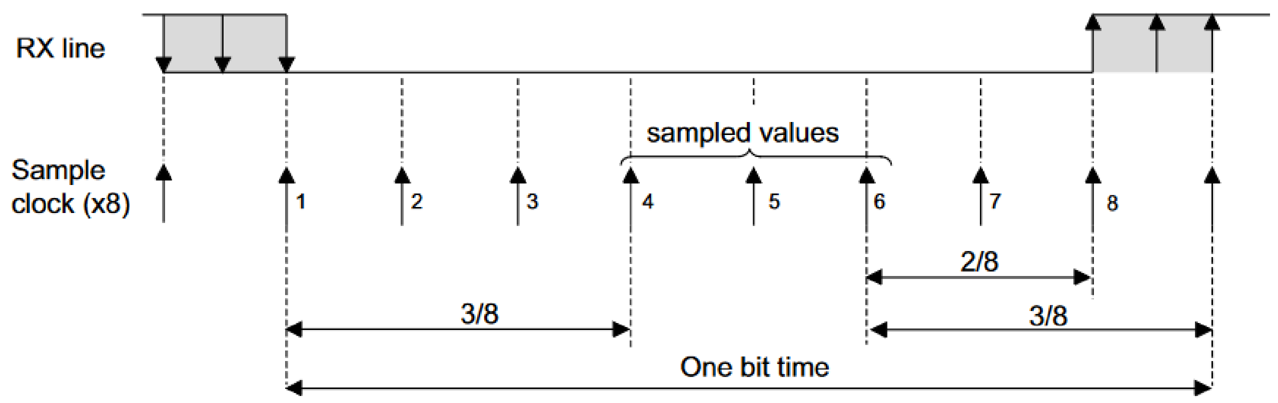
- The receiver of the USART peripheral implements different user-configurable oversampling techniques (by 8 and 16) for data recovery by discriminating between the valid incoming data and noise.
- When oversampling by 16 is used, the receiver engine samples a one-bit period 16 times. That means it takes 16 samples to understand that bit. A bit can be either 0 or 1.
- o understand whether the bit is 1 or 0, a receiver engine takes 16 samples, in which the samples taken at the position of 8, 9, and 10 will be analyzed.



Oversampling (OVER bit)

- If oversampling by 8 is used, then the receiver engine takes 8 samples in a 1-bit period or 1-bit time, and the samples taken at the position of 4, 5, and 6 (or sampled values) will be compared to understand whether this bit is 1 or 0.
- By using these sampled values, the USART engine understands whether it is valid data or noise.
- The noise detected flag(NF) in USART_SR is set to 1 if noise is detected on sampled values 4,5, and as follows:

sampled values	NF status	Received bit value
000	0	0
001	1	0
010	1	0
011	1	1
100	1	0
101	1	1
110	1	1
111	0	1



Derivation USARTDIV from USART_BRR register values when OVER8=0

Example 1:

If DIV_Mantissa = 27 and DIV_Fraction = 12 (USART_BRR = 0x1BC), then

Mantissa (USARTDIV) = 27

Fraction (USARTDIV) = 12/16 = 0.75

Therefore USARTDIV = 27.75

Example 2:

To program USARTDIV = 25.62

This leads to:

DIV_Fraction = 16*0.62 = 9.92

The nearest real number is 10 = 0xA

DIV_Mantissa = mantissa (25.620) = 25 = 0x19

Then, USART_BRR = 0x19A. Hence USARTDIV = 25.625

Example 3:

To program USARTDIV = 50.99

This leads to:

DIV_Fraction = 16*0.99 = 15.84

The nearest real number is 16 = 0x10 => overflow of DIV_frac[3:0] => carry must be added up to the mantissa

DIV_Mantissa = mantissa (50.990 + carry) = 51 = 0x33

Then, USART_BRR = 0x330. Hence USARTDIV = 51.000

Derivation of USARTDIV from USART_BRR register values when OVER8=1

Example 1:

If DIV_Mantissa = 0x27 and DIV_Fraction[2:0]= 6 (USART_BRR = 0x1B6), then

Mantissa (USARTDIV) = 27

Fraction (USARTDIV) = 6/8 = 0.75

Therefore USARTDIV = 27.75

Example 2:

To program USARTDIV = 25.62

This leads to:

DIV_Fraction = 8*0.62 = 4.96

The nearest real number is 5 = 0x5

DIV_Mantissa = mantissa (25.620) = 25 = 0x19

Then, USART_BRR = 0x195 => USARTDIV = 25.625

Example 3:

To program USARTDIV = 50.99

This leads to:

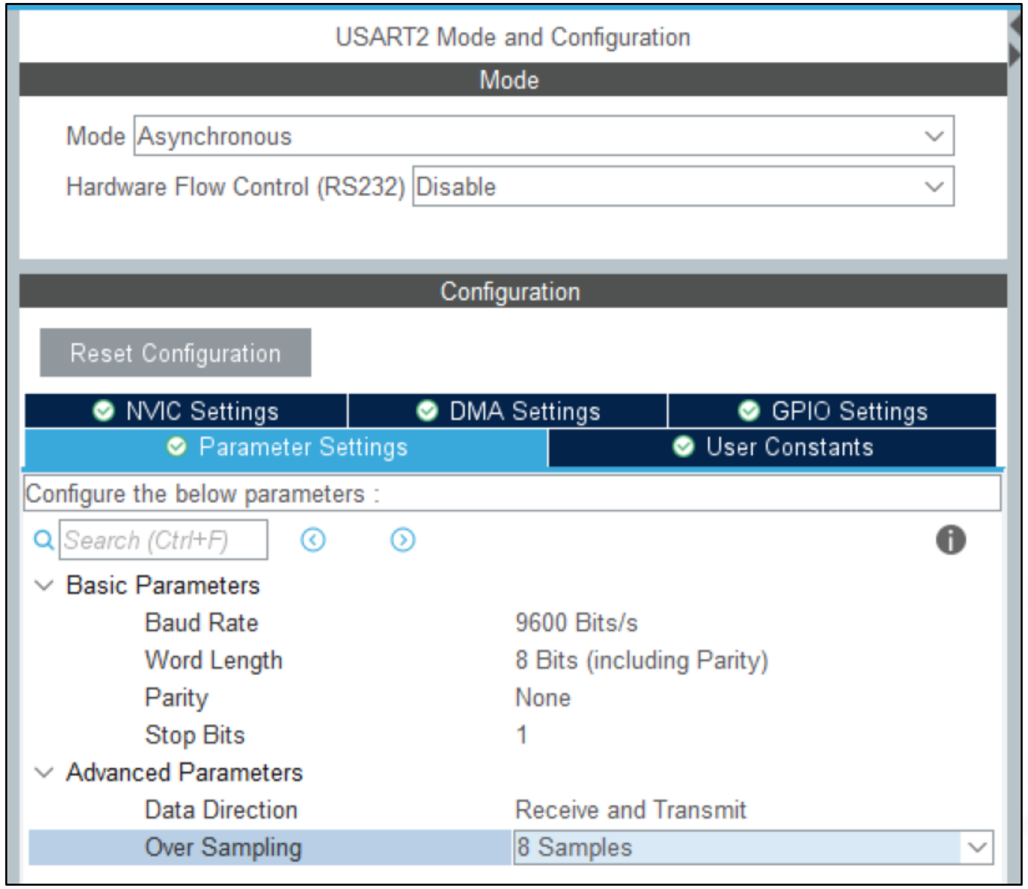
DIV_Fraction = 8*0.99 = 7.92

The nearest real number is 8 = 0x8 => overflow of the DIV_frac[2:0] => carry must be added up to the mantissa

DIV_Mantissa = mantissa (50.990 + carry) = 51 = 0x33

Then, USART_BRR = 0x0330 => USARTDIV = 51.000

Programming USART in STM32CubeMX



UART_HandleTypeDef huart2;

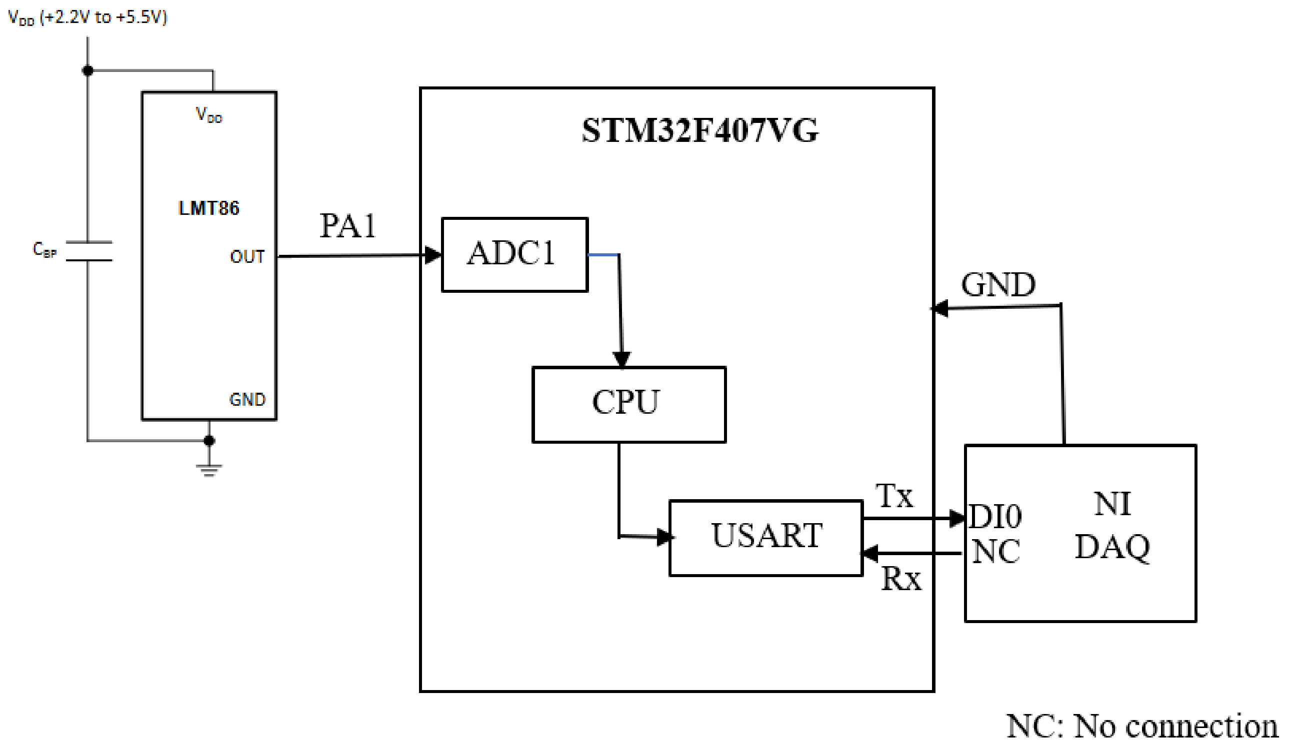
void MX_USART2_UART_Init(**void**);

HAL_StatusTypeDef HAL_UART_Transmit(**UART_HandleTypeDef** *huart, **const uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

HAL_StatusTypeDef HAL_UART_Receive(**UART_HandleTypeDef** *huart, **uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

Question

- Design STM32F407VG based system to interface temperature senso6 LMT86 and send temperature value through the serial port. Configure baud rate as 9600, 8 bits data and use USART 2 module. Use following interfacing diagram.



Question

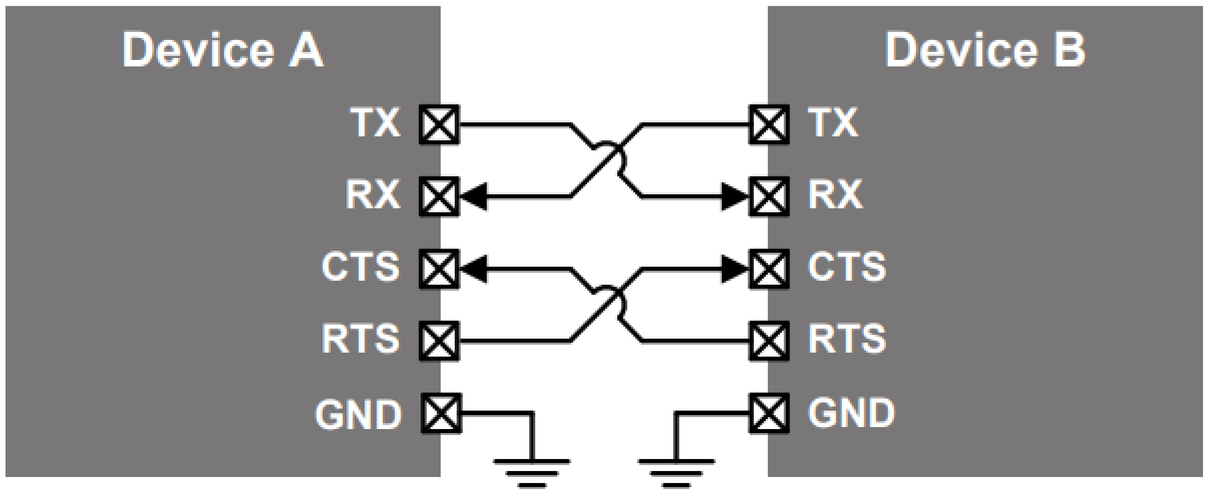
- Design STM32F407VG based system to monitor switch and display status on serial port.
- Design STM32F407VG based system to receive a string on serial port and count number of vowels and blink LED correspondingly.

Hardware Flow Control

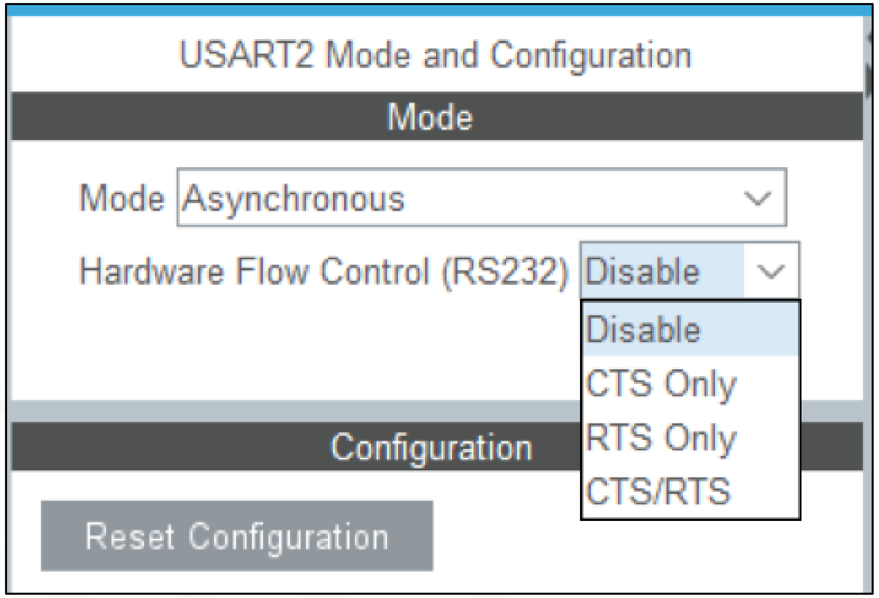
- UART Flow Control is a method for slow and fast devices to communicate with each other over UART without the risk of losing data.
- With hardware flow control (also called RTS/CTS flow control), two extra wires are needed in addition to the data lines. They are called **RTS (Request to Send) and CTS (Clear to Send).**
- These wires are cross-coupled between the two devices, so RTS on one device is connected to CTS on the other device and vice versa.
- Each device will use its RTS to output if it is ready to accept new data and read
- CTS to see if it is allowed to send data to the other device.

Hardware Flow Control…

Cross Coupled Connection



STM32CubeMx Configuration



Note: Software flow control can also be used, which makes use of three pins(TXD,RXD and GND). Transmission is started and stopped by sending special flow control characters. The flow control characters are sent over the normal TX and RX lines.

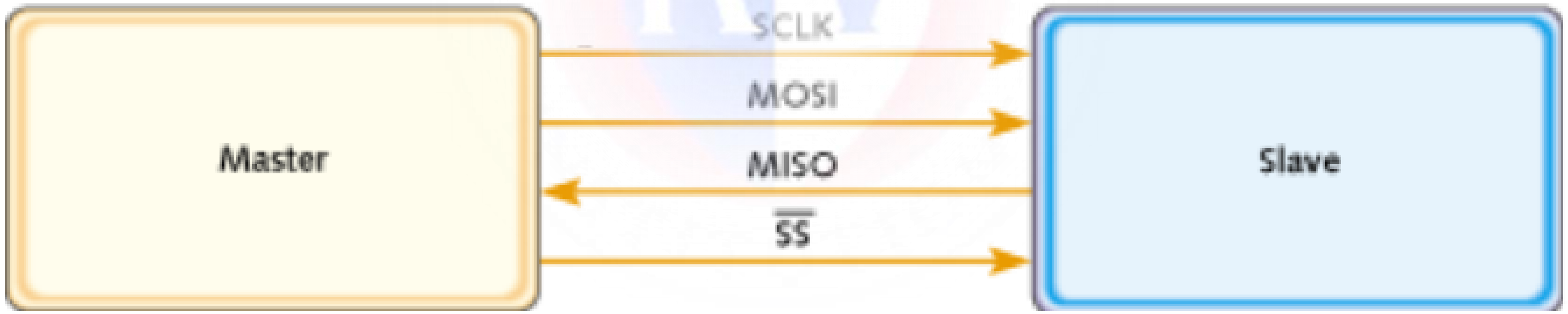
Serial Peripheral Interface(SPI)

- It is a synchronous serial data link that operates in full duplex
- A serial clock line synchronizes the shifting and sampling of the information on two serial data lines.
- The SPI is mainly used to allow a microcontrollers to communicate with peripheral devices such as EEPROMs.
- SPI devices communicate using a master-slave relationship.
- The theoretical speed can reach up to 60 Mbps.
- The STM32F407VG microcontroller board used in the lab support up to 42 Mbps at APB clock of 84 MHz(Maximum).

SPI signals

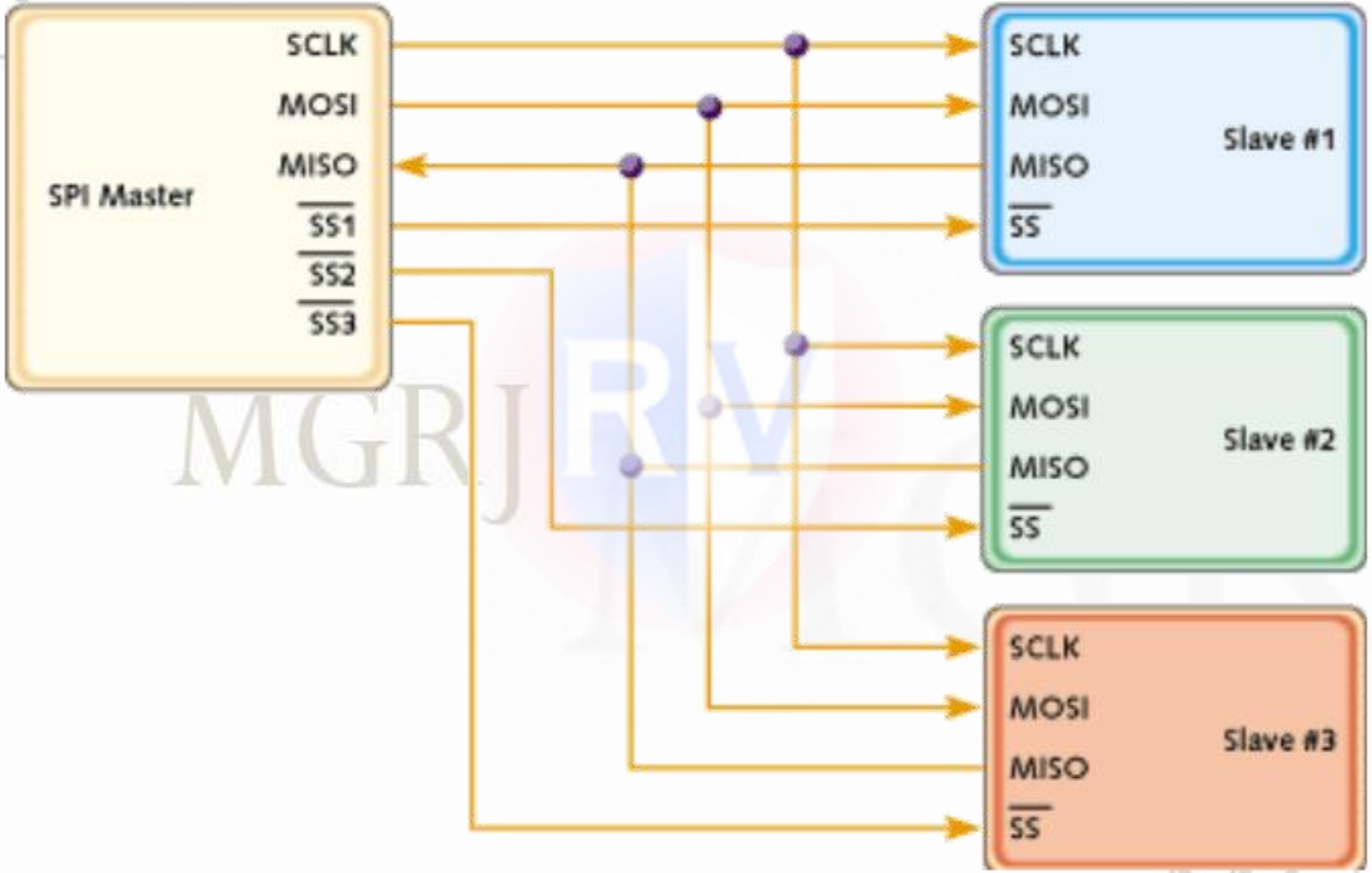
- Clock: SCLK
- Master Data Output, Slave Data Input: MOSI
- Master Data Input, Slave Data Output: MISO
- Slave Select: SS

- Actually a "3 + n" wire interface with n = number of devices



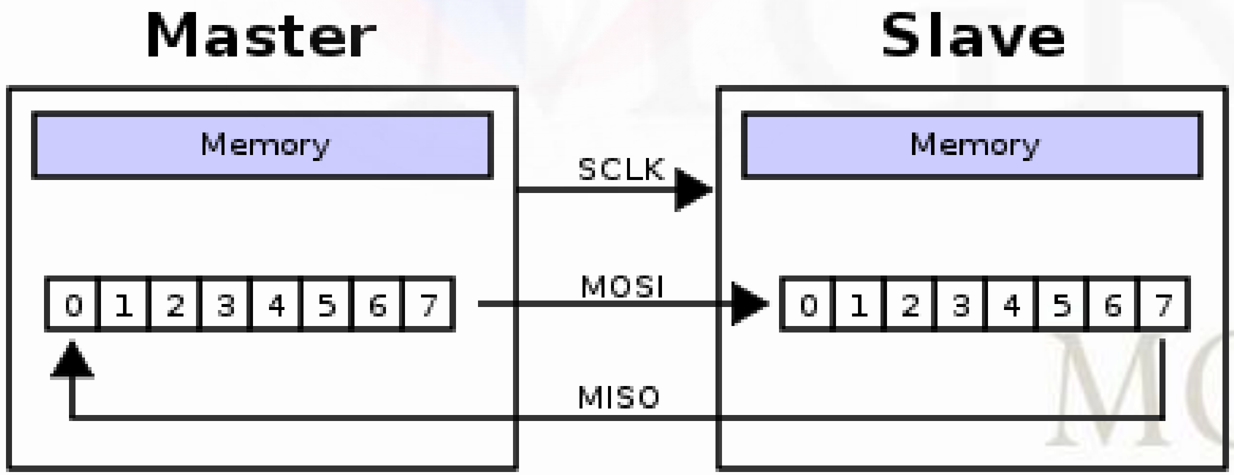
Single master, single slave SPI implementation

Single master, multiple slave SPI implementation



Data transmission

- Transmissions normally involve two shift registers of some given word size (8 /16 bit) one in the master and one in the slave; they are connected in a ring.
- Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register.
- Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.



Clock polarity and phase

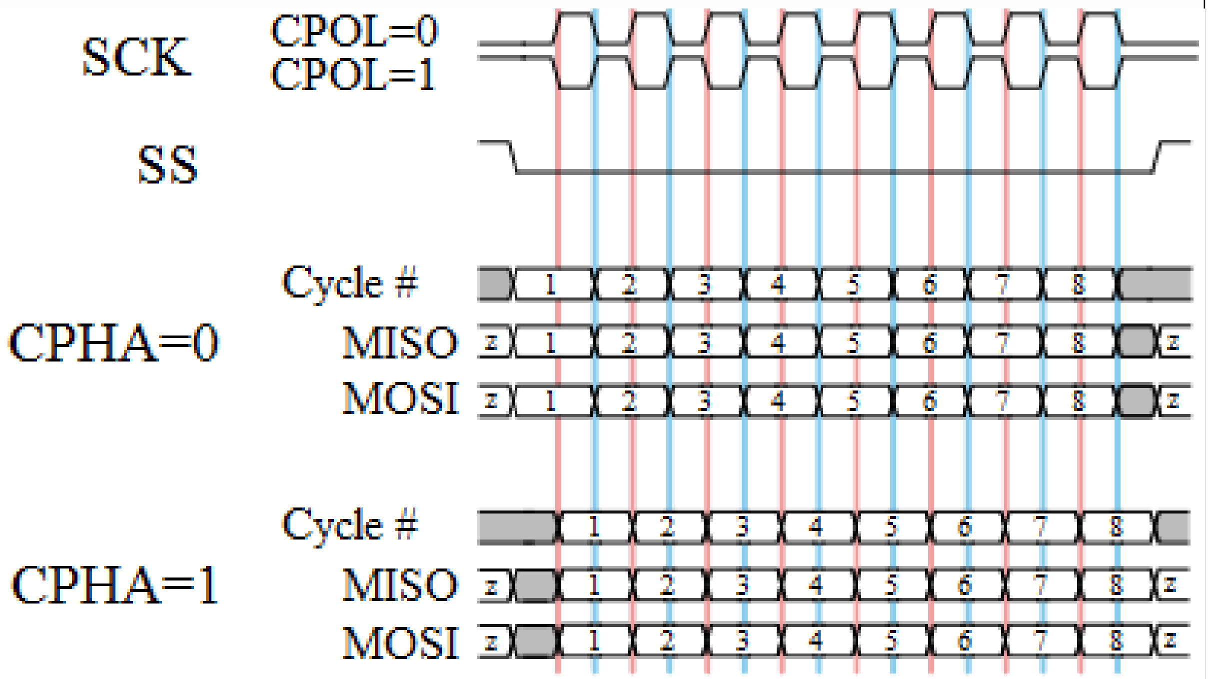
- In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data.

  At CPOL=0,the base value of the clock is zero

  At CPOL=1, the base value of the clock is one

  For CPHA=0, data is captured on the clock's rising edge and data is propagated on a falling edge.

  For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.



SPI Module in STM32F407VG

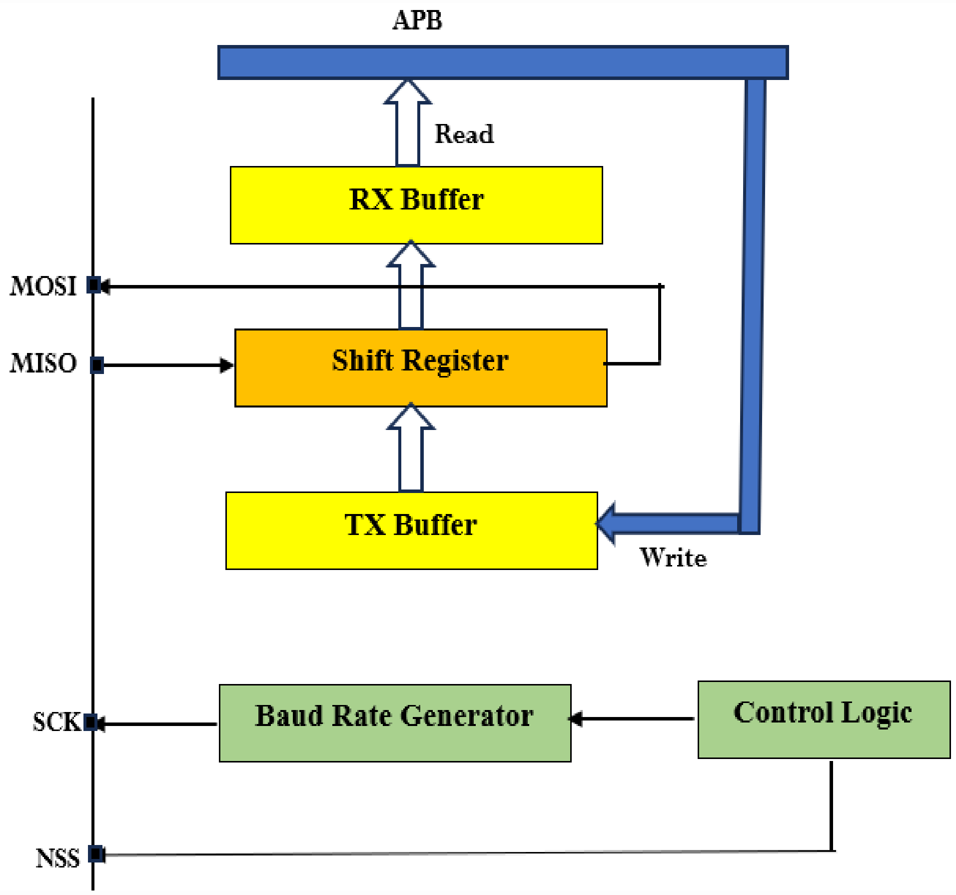
- 3 SPI modules in MCU
- Full-duplex synchronous transfers on three lines
- Master or slave operation & Multimaster mode capability &
- 8 master mode baud rate prescalers.
- The SPI is connected to external devices through four pins:

  -**MISO:** Master In / Slave Out data.

  -**MOSI:** Master Out / Slave In data.

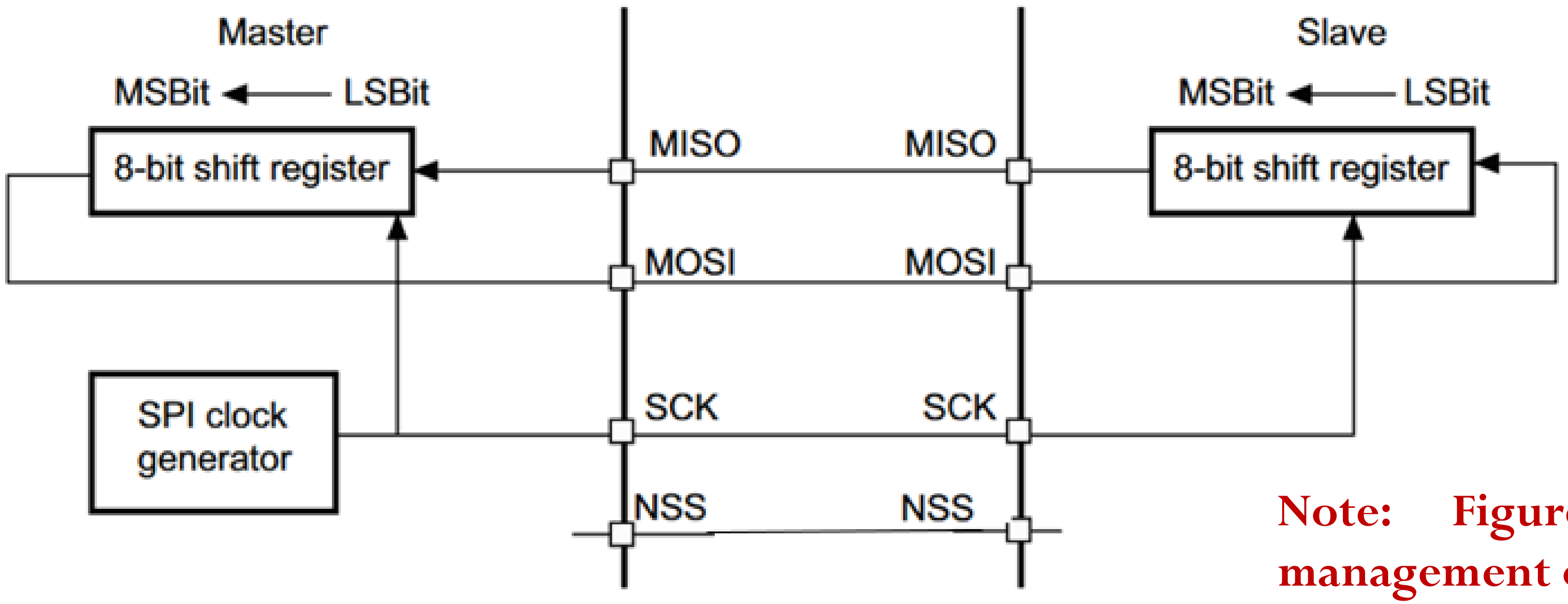
  -**SCK:** Serial Clock output for SPI masters and input for SPI slaves.

  - **NSS:** Slave select. This is an optional pin to select a slave device.



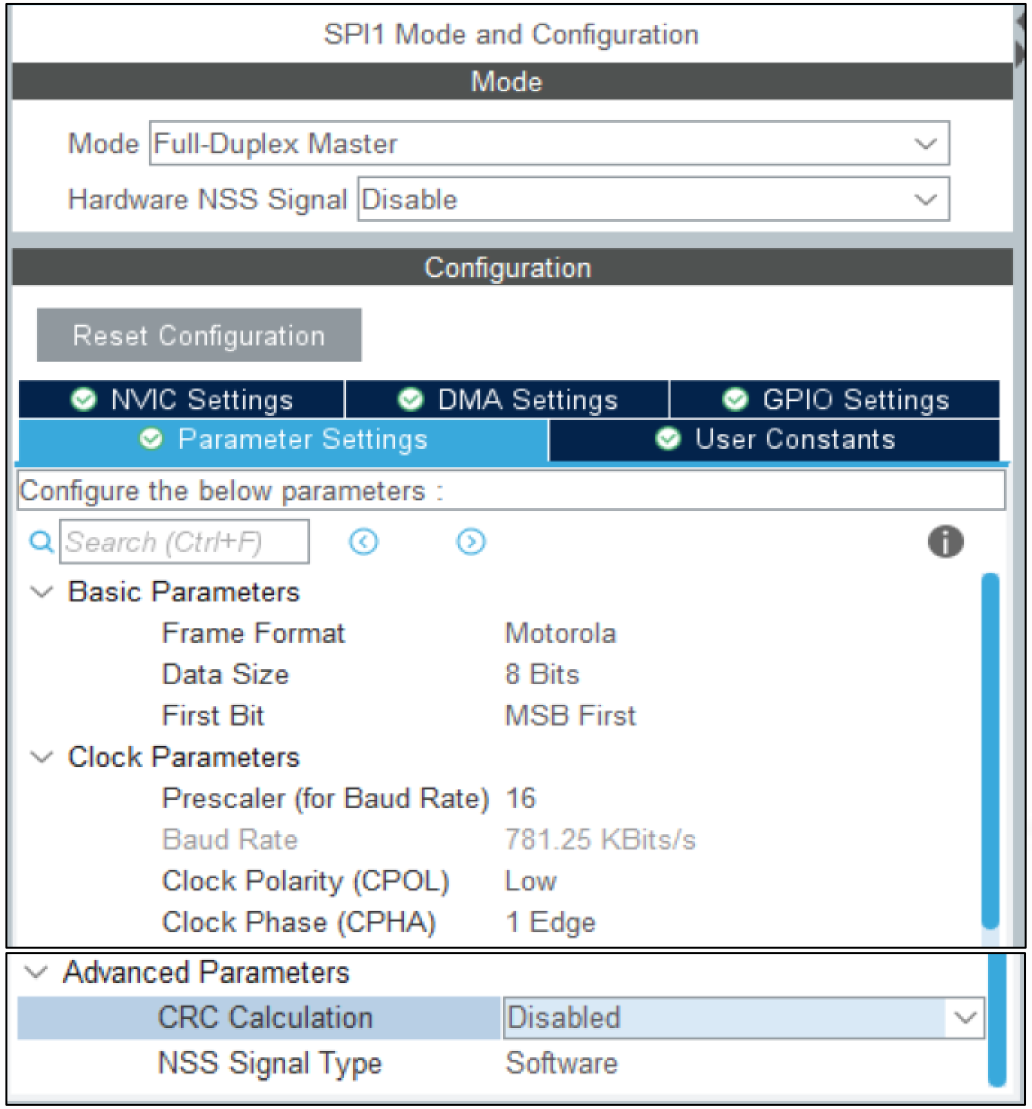
Single Master/ Single Slave Application

- The MOSI pins are connected together and the MISO pins are connected together.
- When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin.
- This implies full-duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).



Note: Figure shows hardware management of slave select signal.

Programming SPI in STM32CubeMX



Hardware NSS Signal Disabled:

- With this option, NSS signal type is selected as Software.
- Software NSS management:
  The slave select information is driven internally program statements. The external NSS pin remains free for other application uses.

CRC Calculation Disabled:

- Error checking of transmitted data and received data using Cyclic Redundancy Check is disabled.

SPI_HandleTypeDef hspi1;

void MX_SPI1_Init(**void**);

Programming SPI in STM32CubeMX

HAL_StatusTypeDef HAL_SPI_Transmit(**SPI_HandleTypeDef** *hspi, **uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

HAL_StatusTypeDef HAL_SPI_Receive(**SPI_HandleTypeDef** *hspi, **uint8_t** *pData, **uint16_t** Size, **uint32_t** Timeout)

Explanation as seen in Keil:

```
/**
  * @brief  Receive an amount of data in blocking mode.
  * @param  hspi pointer to a SPI_HandleTypeDef structure that contains
  *              the configuration information for SPI module.
  * @param  pData pointer to data buffer
  * @param  Size amount of data to be received
  * @param  Timeout Timeout duration
  * @retval HAL status
  */
```

Question

Design a system based STM32F407VG MCU to connect a sensor that require ADC resolution of 16 bits. Th designers are suggested to use TI's ADS1118 ADC which provide SPI compatible interface. Read the data sheet to understand the scheme of interfacing. Use STM32CubeMx to generate HAL for SPI. Develop functions to read to read digital value from ADS118. Write application code to test the same.