# BME3321:Introduction to Microcontroller Programming

## Topic 7: Universal Synchronous/Asynchronous Serial Communications – USART peripheral

Assist. Prof. Dr. İsmail Cantürk

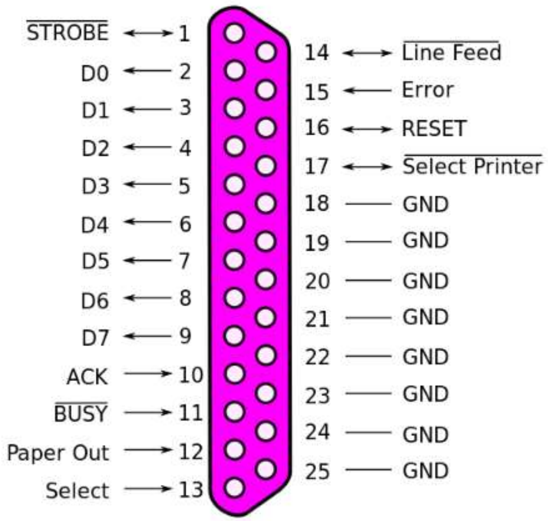
Mastering STM32 (ch8)

STM32F407 reference manual (ch 30)

## Some terminologies

To exchange data between two (or even more) digital devices, we have two alternatives:

Parallel communication: The number of physical communication lines equal to the size of the each data packet (e.g., eight independent lines for a byte (eight bits) transmission).

## Some terminologies

Serial communication: Transmission of bits one by one.


| Pin | Signal | Color |
|---|---|---|
| 1 | VCC | |
| 2 | D- | |
| 3 | D+ | |
| 4 | GND | |

## Serial Communications

There is a really high number of serial communication protocols and hardware interfaces available in the electronics industry.

One of this is the Universal Synchronous/Asynchronous Receiver/Transmitter interface, also simply known as USART.

Almost every microcontroller provides at least one UART peripheral. Almost all STM32 MCUs provide at least two UART/USART interfaces. E.g., STM32F4 has six USART peripherals.

MCUs can use USART peripheral to communicate with computers, other MCUs, or different digital systems.

## Serial Communications

When the information flows between two devices via USART, both devices (transmitter and the receiver) have to agree on the **timing** (i.e., how long it takes to transmit each bit)

Synchronous transmission: the transmitter and the receiver share a common clock generated by one of the two devices (usually the device that acts as the master of this interconnection system).

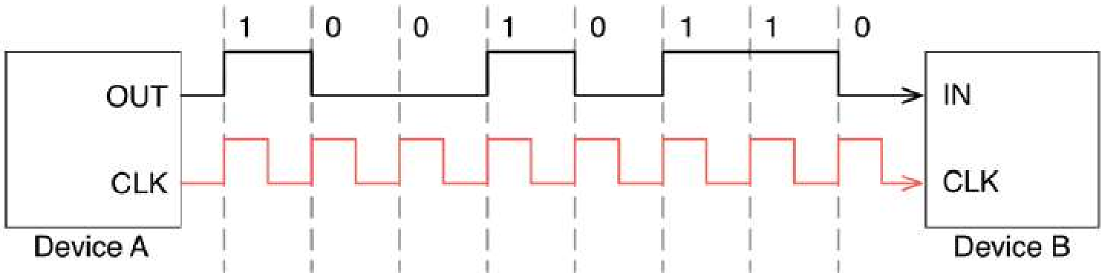



Figure 1: A serial communication between two devices using a shared clock source

The transmission speed and duration are defined by the clock:
When the master device starts clocking the dedicated line, it means that it is going to send a sequence of bits.
Its frequency determines how fast we can transmit a single byte on the communication channel.

## Serial Communications

Asynchronous transmission: It does not use a dedicated clock line. Both devices involved in data transmission agree on how long it takes to transmit a single bit and when to start and finish transmission.

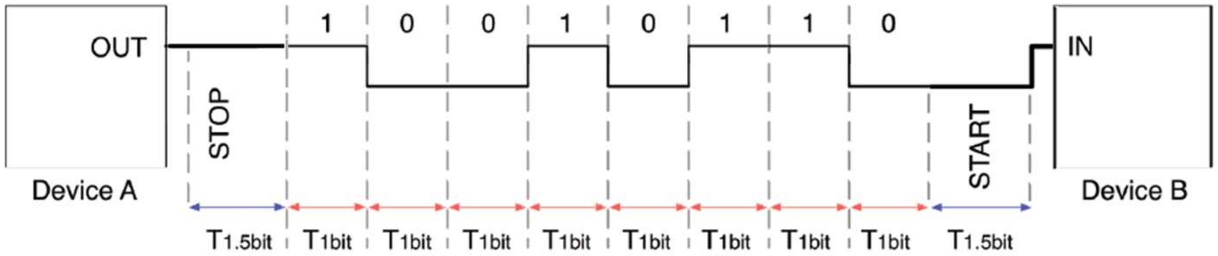


Figure 2: The timing diagram of a serial communication without a dedicated clock line

The idle state (that is, no transmission occurring) is represented by the high signal.

Transmission begins with a START bit, which is represented by the low level. The negative edge is detected by the receiver and 1.5 bit periods after this the sampling of bits begins.

The transmission is ended by a STOP bit, which last 1.5 bits.

Eight data bits are transmitted. The least significant bit (LSB) is typically transmitted first. An optional parity bit is then transmitted (for error checking of the data bits). Often this bit is omitted if the transmission channel is assumed to be noise free.

## Difference between a USART and a UART

USART : A Universal Synchronous Receiver/Transmitter interface is a device able to transmit data serially using two I/Os, one acting as transmitter (TX) and one as receiver (RX), plus one additional I/O as one clock line.

UART : a Universal Asynchronous Receiver/Transmitter uses only two RX/TX I/Os.

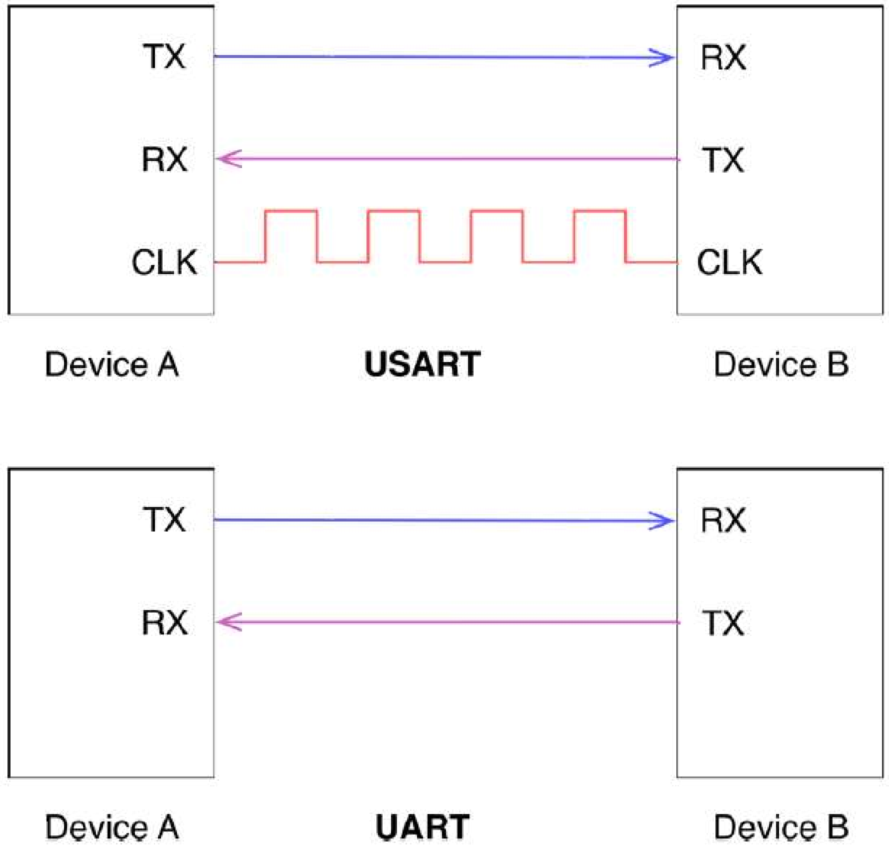
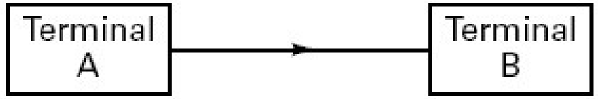



Figure 3: The signaling difference between a USART and a UART

## Some other communication terminologies: Full Duplex, Half Duplex, and Simplex

**Simplex**

One way communication.

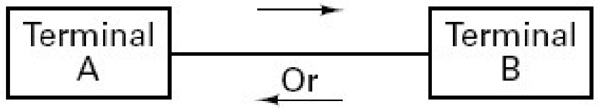



Transmission in only one direction

(a)

**Half Duplex**

You can  whether transmit or receive data.

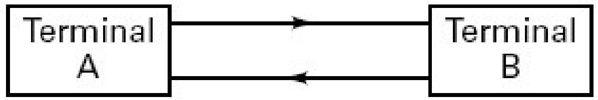



Transmission in either direction,
but not simultaneously

(b)

**Full Duplex**

You can transmit and receive data at the same time by using the same terminal




Transmission in both directions simultaneously

(c)

# USARTs of STM32F4

Table 148. USART mode configuration[(1)]

| USART modes | USART 1 | USART 2 | USART 3 | UART4 | UART5 | USART 6 |
|---|---|---|---|---|---|---|
| Asynchronous mode | X | X | X | X | X | X |
| Hardware flow control | X | X | X | NA | NA | X |
| Multibuffer communication (DMA) | X | X | X | X | X | X |
| Multiprocessor communication | X | X | X | X | X | X |
| Synchronous | X | X | X | NA | NA | X |
| Smartcard | X | X | X | NA | NA | X |
| Half-duplex (single-wire mode) | X | X | X | X | X | X |
| IrDA | X | X | X | X | X | X |
| LIN | X | X | X | X | X | X |

1. X = supported; NA = not applicable.

USARTs can be configured to work both in synchronous and asynchronous mode.

UARTs can only work in asynchronous mode.

## USARTs

Like all STM32 peripherals, even the USARTs are mapped in the memory (i.e., peripheral region of the memory mapping).

Check from reference manual.

They are connected to the different buses.

| | | |
|---|---|---|
| 0x4000 5000 - 0x4000 53FF | UART5 | |
| 0x4000 4C00 - 0x4000 4FFF | UART4 | |
| 0x4000 4800 - 0x4000 4BFF | USART3 | |
| 0x4000 4400 - 0x4000 47FF | USART2 | |
| 0x4000 4000 - 0x4000 43FF | I2S3ext | |
| 0x4000 3C00 - 0x4000 3FFF | SPI3 / I2S3 | |
| 0x4000 3800 - 0x4000 3BFF | SPI2 / I2S2 | APB1 |
| 0x4000 3400 - 0x4000 37FF | I2S2ext | |

## USART management with HAL

UART management are  done with a C struct *UART_HandleTypeDef*, which is defined in the following way:

```
typedef struct {
USART_TypeDef                *Instance;       /* UART registers base address */
UART_InitTypeDef             Init;            /* UART communication parameters */
UART_AdvFeatureInitTypeDef   AdvancedInit;    /* UART Advanced Features initialization parameters */
uint8_t                      *pTxBuffPtr;     /* Pointer to UART Tx transfer Buffer */
uint16_t                     TxXferSize;      /* UART Tx Transfer size */
uint16_t                     TxXferCount;     /* UART Tx Transfer Counter */
uint8_t                      *pRxBuffPtr;     /* Pointer to UART Rx transfer Buffer */
uint16_t                     RxXferSize;      /* UART Rx Transfer size */
uint16_t                     RxXferCount;     /* UART Rx Transfer Counter */
DMA_HandleTypeDef            *hdmatx;         /* UART Tx DMA Handle parameters */
DMA_HandleTypeDef            *hdmarx;         /* UART Rx DMA Handle parameters */
HAL_LockTypeDef              Lock;            /* Locking object */
__IO HAL_UART_StateTypeDef   State;           /* UART communication state */
__IO HAL_UART_ErrorTypeDef   ErrorCode;       /* UART Error code */
} UART_HandleTypeDef;
```

## USART management

Some most important fields of this struct:

**Instance** : is the pointer to the USART descriptor that we are going to use. For example, USART2.

**Init** : is a nested C struct UART_InitTypeDef, which is used to configure the UART interface.

**AdvancedInit** : This field is used to configure more advanced UART features like the automatic Baud Rate detection and the TX/RX pins swapping.

**pTxBuffPtr** and **pRxBuffPtr**: these fields point to the transmit and receive buffer respectively.

pTxBuffPtr and pRxBuffPtr are used as source to transmit **TxXferSize** bytes over the UART and to receive **RxXferSize** when the UART is configured in Full Duplex Mode.

TxXferCount, RxXferCount, Lock : these fields are used internally by the HAL.

## USART management

All the UART configuration activities are performed by using the C struct *UART_InitTypeDef*, which is defined in the following way:

```
typedef struct {
uint32_t BaudRate;
uint32_t WordLength;
uint32_t StopBits;
uint32_t Parity;
uint32_t Mode;
uint32_t HwFlowCtl;
uint32_t OverSampling;
} UART_InitTypeDef;
```

## USART management

Some important fields of this struct:

BaudRate : this parameter refers to the connection speed, expressed in bits per seconds. Even if the parameter can assume an arbitrary value, usually the BaudRate comes from a list of well-known and standard values. Not all BaudRates can be easily achieved without introducing communication errors.

| Baud rate | | Oversampling by 16 | | Oversampling by 8 | |
|---|---|---|---|---|---|
| S.No | Desired (Bps) | Actual | %Error | Actual | %Error |
| 2 | 2400 | 2400 | 0 | 2400 | 0 |
| 3 | 9600 | 9600 | 0 | 9600 | 0 |
| 4 | 19200 | 19200 | 0 | 19200 | 0 |
| 5 | 38400 | 38400 | 0 | 38400 | 0 |
| 6 | 57600 | 57620 | 0.03 | 57590 | 0.02 |
| 7 | 115200 | 115110 | 0.08 | 115250 | 0.04 |
| 8 | 230400 | 230760 | 0.16 | 230210 | 0.8 |
| 9 | 460800 | 461540 | 0.16 | 461540 | 0.16 |
| 10 | 921600 | 923070 | 0.16 | 923070 | 0.16 |
| 11 | 2000000 | 2000000 | 0 | 2000000 | 0 |
| 12 | 3000000 | 3000000 | 0 | 3000000 | 0 |
| 13 | 4000000 | N.A. | N.A. | 4000000 | 0 |

Check the reference manual.

## USART management

WordLength : it specifies the number of data bits transmitted or received in a frame. We can transmit 8 or 9 data bits in a frame. This number does not include the overhead bits transmitted, such as the start and stop bits.

StopBits : This field specifies the number of stop bits transmitted. We can use one or two stop bits to signal the end of the frame.

Parity : it indicates the parity mode. Parity is a very simple form of error checking. When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the Word length is set to 8 data bits).

| Parity Mode | Description |
|---|---|
| UART_PARITY_NONE | No parity check enabled |
| UART_PARITY_EVEN | The parity bit is set to 1 if the count of bits equal to 1 is odd |
| UART_PARITY_ODD | The parity bit is set to 1 if the count of bits equal to 1 is even |

## USART management

Mode : it specifies whether the RX or TX mode is enabled or disabled.

| UART Mode | Description |
|---|---|
| UART_MODE_RX | The UART is configured only in receive mode |
| UART_MODE_TX | The UART is configured only in transmit mode |
| UART_MODE_TX_RX | The UART is configured to work bot in receive an transmit mode |

HwFlowCtl : it is for RS232 communication.

OverSampling : when the UART receives a frame from the remote peer, it samples the signals in order to compute the number of 1 and 0 constituting the message. This part can be 8 or 16.

For example, if the over sampling field is 16, 16 samples are taken for each frame bit.

## UART Communication in Polling Mode

To transmit a sequence of bytes over the USART in polling mode the HAL provides the function:

```
HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

huart: it is the pointer which identifies and configures the UART peripheral;

pData: is the pointer to an array, with a length equal to the Size parameter, containing the sequence of bytes we are going to transmit.

Timeout: is the maximum time, expressed in milliseconds, we are going to wait for the completion of transmitting .

## UART Communication in Polling Mode

To receive a sequence of bytes over the USART in polling mode the HAL provides the function:

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

huart: it is the pointer which identifies and configures the UART peripheral;

pData: is the pointer to an array, with a length equal to the Size parameter, containing the sequence of bytes we are going to receive.

Timeout: is the maximum time, expressed in milliseconds, we are going to wait for the completion of receiving.

## UART Communication in Interrupt Mode

USART peripheral may generate IRQs depending on different situations. The source of these interrupts include both IRQs related to data transmission and to communication errors.

Table 6: The list of USART related interrupts

| Interrupt Event | Event Flag | Enable Control Bit |
|---|---|---|
| Transmit Data Register Empty | TXE | TXEIE |
| Clear To Send (CTS) flag | CTS | CTSIE |
| Transmission Complete | TC | TCIE |
| Received Data Ready to be Read | RXNE | RXNEIE |
| Overrun Error Detected | ORE | RXNEIE |
| Idle Line Detected | IDLE | IDLEIE |
| Parity Error | PE | PEIE |
| Break Flag | LBD | LBDIE |
| Noise Flag, Overrun error and Framing Error in multi buffer communication | NF or ORE or FE | EIE |

## UART Communication in Interrupt Mode

However, STM32 MCUs are designed so that all these IRQs are bound to just one ISR for every USART peripheral.  It is up to the user code to analyze the corresponding event Flag to infer which interrupt has generated the request.

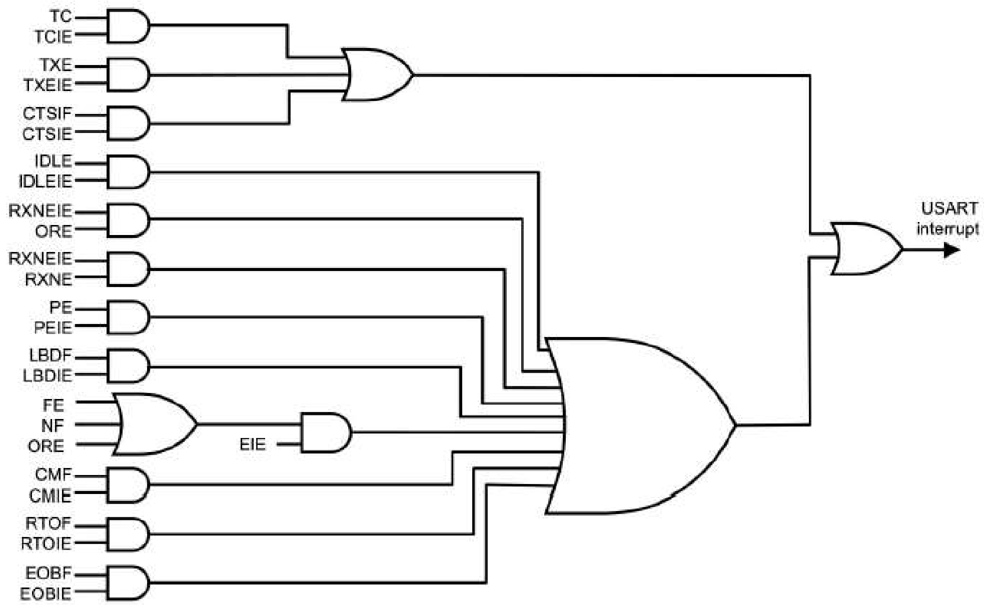



Figure 11: How the USART interrupt events are connected to the same interrupt vector

## UART Communication in Interrupt Mode

To transmit a sequence of bytes in interrupt mode, the HAL defines the function:

HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, **uint8_t** *pData, **uint16_t** Size);

To receive sequence of bytes in interrupt mode, the HAL defines the function:

HAL_UART_Receive_IT(UART_HandleTypeDef *huart, **uint8_t** *pData, **uint16_t** Size);

huart: it is the pointer which identifies and configures the UART peripheral.

pData: it is the pointer to an array, with a length equal to the Size parameter, containing the sequence of bytes we are going to transmit.