STM Cube software:

STM Cube program is a graphical software configuration tool, it generates a source code for the user in c language which includes drivers and interfaces that are implemented in a standard library Called HAL library (Hardware abstraction layer library).

The HAL library is complementary and covers a wide range of requirements for the user since it offers high-level and feature-oriented APIs with a high probability level.

Peripheral drivers APIs are organized in four groups:

Group	Examples
Initialization and deinitialization group	HAL_GPIO_Init(); HAL_GPIO_DeInit();
Process operation	HAL_UART_Receive(); HAL_UART_Transmit(); HAL_UART_Transmit_IT();
Peripherical subsystem configuration	HAL_ADC_ConfigChannel(); HAL_RTC_SetAlarm();
Peripherical GetState and GetError	HAL_I2C_GetState(); HAL_I2C_GetError();

UART:

UART stands for Universal asynchronous receiver-transmitter, it is one of the most important protocols used in communication system. UART involves transmitting and receiving serial data using only two wires.

For any chip contains UART protocol, it has two pins (RX and TX) and when the chip is connected for another one, each TX is connected to the RX of the second chip as shown in figure 1.

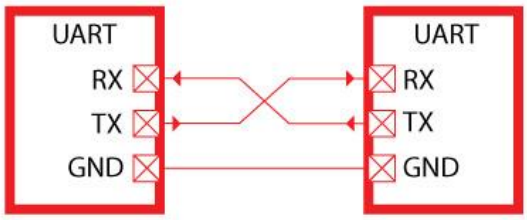


Figure 1 UART connection between two chips.

In this project, UART is used to communicate between the micro controller (STM32) and Raspberry Pi, they are full duplex which means that one can be a transmitter and another one is a receiver or vice versa.

STM32F103C8 has two USARTs; USART1 which its pins are PA2 and PA3, and USART2 which its pins are PA9 and PA10. USART1 was used with Raspberry pi and connected to it as shown in figure 2.

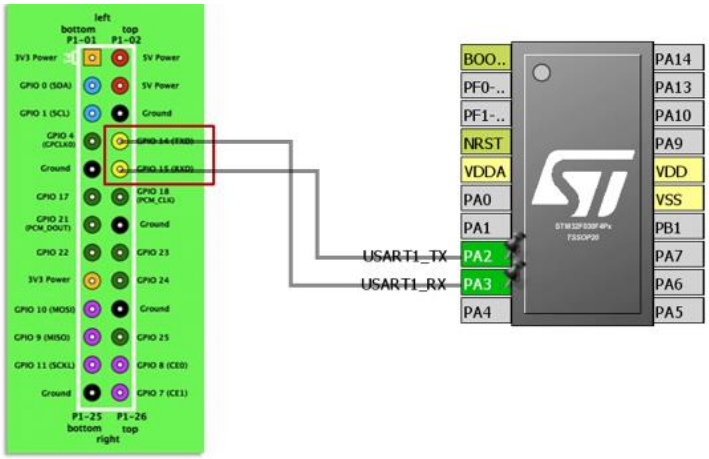


Figure 2 UART connection between Raspberry pi and STM32

UART in HAL Library:

HAL library provides three methods can be used for UART communication:

1. **Polling method:** This method transmits/receives data in blocking mode and it will not allow any other operation to be implemented until the transmission/receive is done or time out occurred.

2. **Interrupt method:** In this method, data transmission takes place in background or in non-blocking mode, When transmission is complete, HAL_UART_TxCpltCallback() function is called(will be discussed later), and the user can write inside this functions the instructions they need.

3. **DMA method**: DMA works somewhat same as interrupt, means that data transfer is in non-blocking mode.

In DMA, when half of the data get transferred, a half transfer get triggered and HAL_UART_TxHalfCpltCallback() function is called.

The idea behind using DMA is when the second half of the data is being transferred, new data can be written in the first half section.

Handle structure:

Handles contain information about related periphery and they are defined as global variables in the main file.

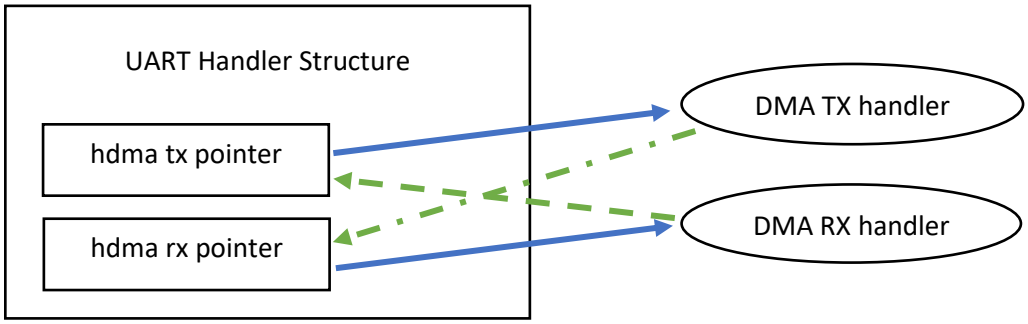
Handle structure consists of some data types, some of them are:

Data Type	Value
Instance	USART1 or USART2
Baud rate	9600 or 115200
Status	Ready, Busy, and Error
Lock	Lock or unlock
Internal Status	TX or RX

*hdma tx pointer	DMA TX handler
*hdma rx pointer	DMA RX handler

i.e., when data is transmitted, first information is stored into the handler (how many bits to send, what is the baud rate, what is the status…etc.), after that they are sent to UART periphery.

Not only the HAL function provides handler for USART1 and USART2, but also it provides handlers when the DMA method is used, and they are linked with UART handler together as shown below.



HAL UART functions:

HAL functions are implemented to help the user to use the UART by the method they chose as well it provides system functions to initialize and set the UART, so the user does not need to initialize the registers and set the values for the UART.

1. Initialization functions:

a. MX_UART_Init():

This function takes void arguement and returns void, it is called automatically in the code generated from STMCube, its functionalities are:

- Store UART parameters into initialize structure in UART handler

```
handler {
        Instance → USART1 or USART2
        Init {
        Baud rate → X value
        Word length → 8/9 or 10 bits
        Stop bits → One or zero bits
        Parity → One or zero bits
        }
....
}
```

- Call HAL_UART_Init()

b. HAL_UART_Init():

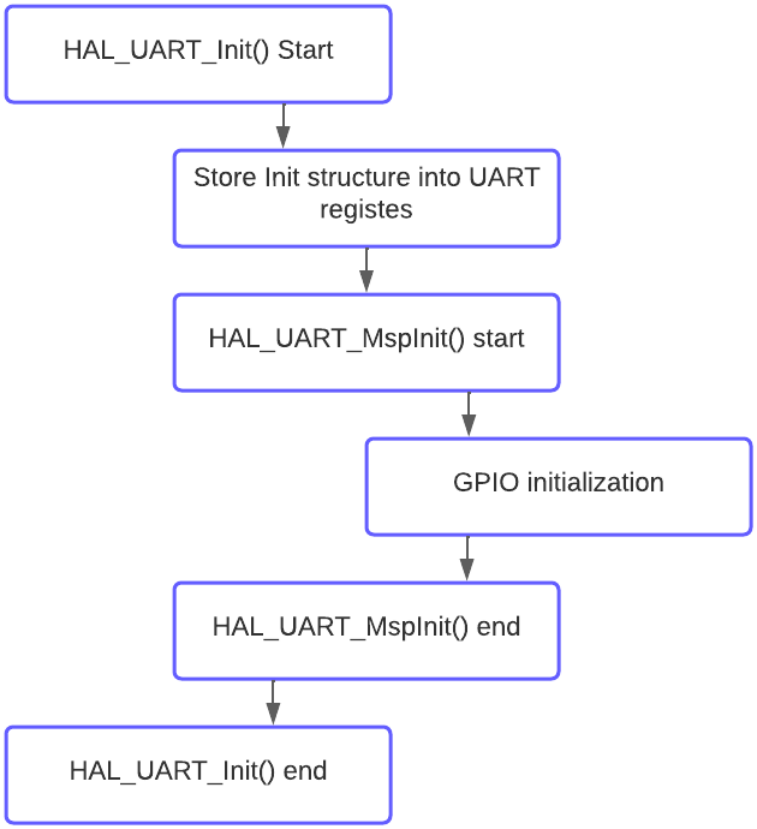
This function takes address of handler structure value and the returns void, it is called inside the MX_UART_Init(), its functionalities are:

- Store initialization structure (handler) into UART registers.
- Call HAL_UART_MspInit()

c. HAL_UART_MspInit():

This function takes handler structure value and the returns void, its function is to initialize the GPIO parameters (pull up, speed, mode, pin state...etc.).

So, the initialization functions can be summarized into the following chart:



2. User Functions:

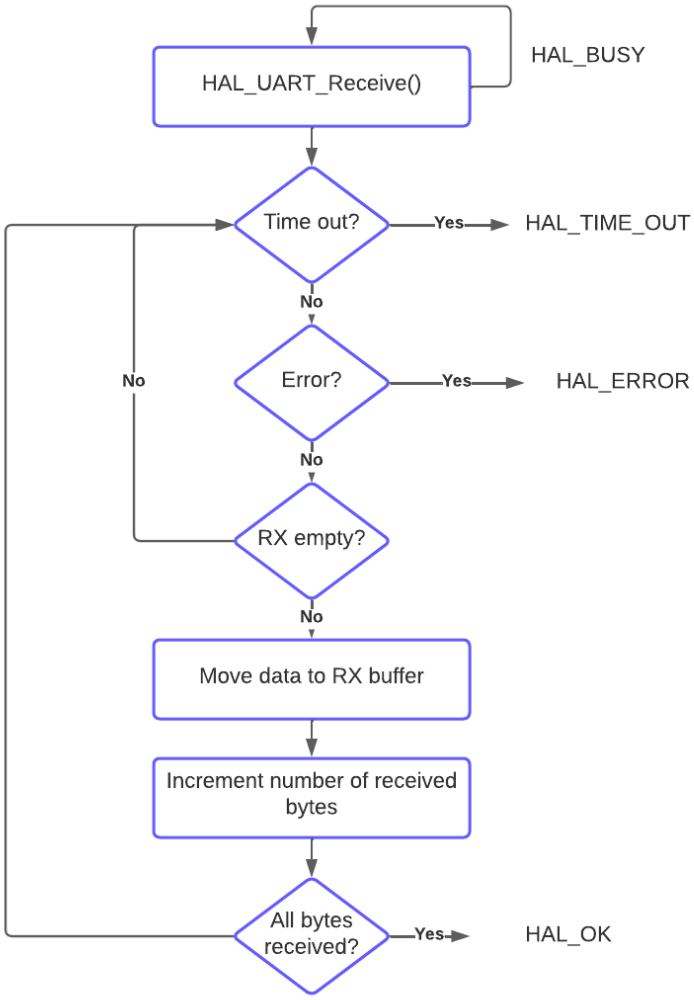
HAL library provides some functions for the three methods which are discussed before, in this subsection, some important functions will be discussed for each method.

a. Polling Method:

HAL functions handle blocking polling processes, the functions that can be used are:

- **HAL_UART_Transmit():** This function takes four arguments; handler structure, buffer for data to be transmitted, size of buffer, and the time for transmitting and it returns HAL_status prototype which it can be HAL_OK, HAL_BUSY, and HAL_ERROR.
- **HAL UART_Receive():** This function takes four arguments; handler structure, buffer for data to be received, size of buffer, and the time for receiving and it returns HAL_status prototype which it can be HAL_OK, HAL_BUSY, and HAL_ERROR.

Inside the receive function (and the same in transmit function), it happens the following:



b. Interrupt method:

Interrupt method is used to solve the problem of blocking mode to make the process complete executing its processes and UART will be in the background (non-blocking mode).

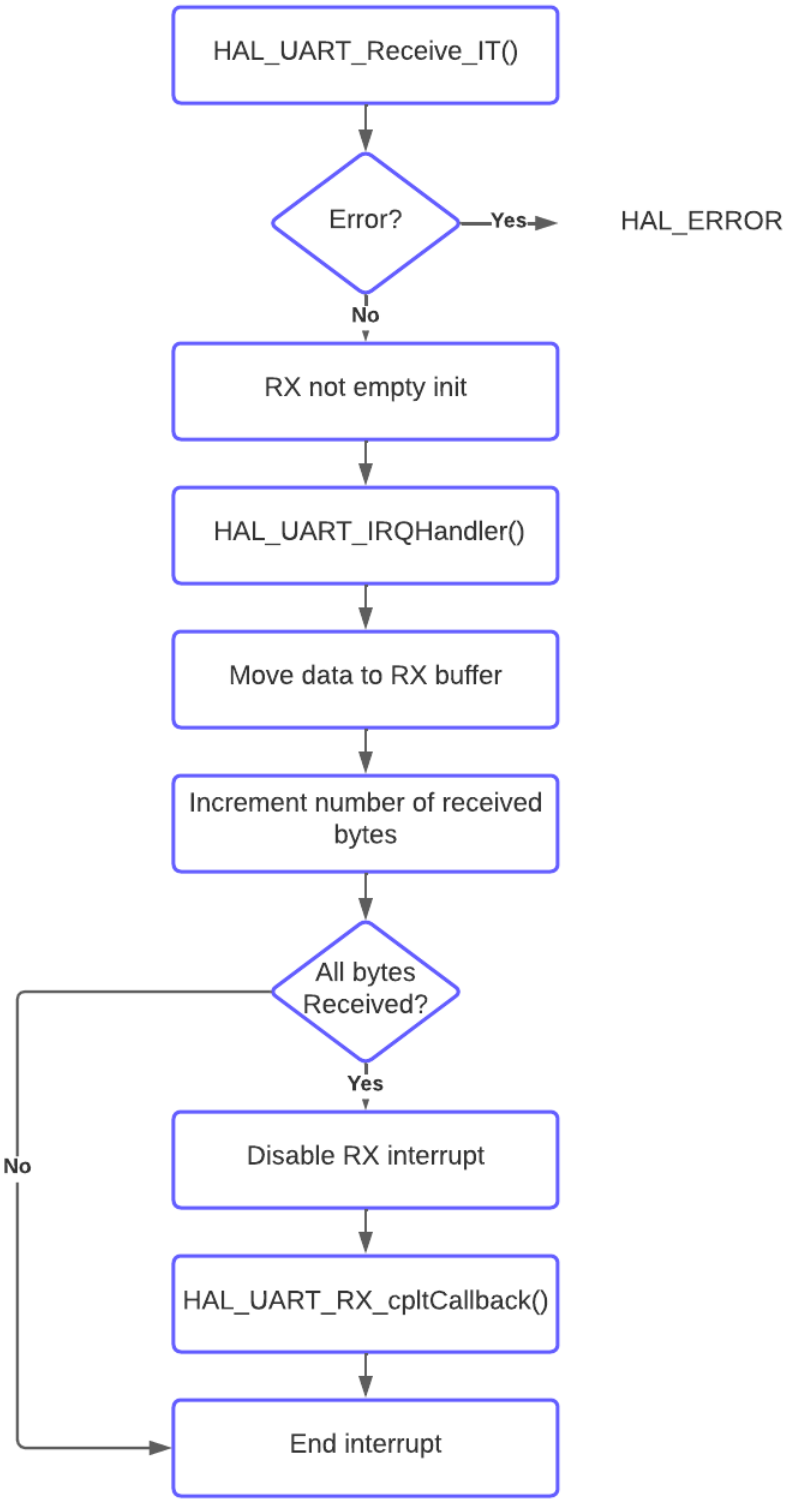
In this method**, HAL_UART_MspInit()** will call two new functions to handle the interrupt, these functions are:

- **HAL_NVIC_SetPriority():** This function is to set the priority of the UART interrupt whether it has a low priority or high. NVIC stands to Nested Vector Interrupt Controller.
- **HAL_NVIC_EnableIRQ():** This function is to enable the handler of the interrupt. IRQ stands for interrupt request.

Also, HAL library provides other functions to be called when performs the receive/transmit functions. HAL functions for the interrupt are:

- **HAL_UART_Transmit_IT():** "IT" stands for interrupt, this function takes three arguments; handler structure, buffer for data to be transmitted, and size of buffer, and it returns HAL_status prototype which it can be HAL_OK or HAL_ERROR.
- **HAL_UART_Receive_IT():** This function takes three arguments; handler structure, buffer for data to be received, and size of buffer, and it returns HAL_status prototype which it can be HAL_OK or HAL_ERROR.
- **HAL_UART_IRQHandler():** This function is to handle the interrupt request of the UART inside the system.
- **HAL_UART_RXcpltCallback():** "cpltCallback" stands for complete callback, this function is automatically called when HAL_UART_Receive_IT() finishes its code, the user can write inside this function since it is used to be such as a flag for the ending of the receiving.
- **HAL_UART_TXcpltCallback():** This function is automatically called when **HAL_UART_Transmit_IT()** finishes its code, the user can write inside this function since it is used to be such as a flag for the ending of the transmission.

Inside the interrupt receive function (and the same in transmit function), it happens the following:



c. DMA method:

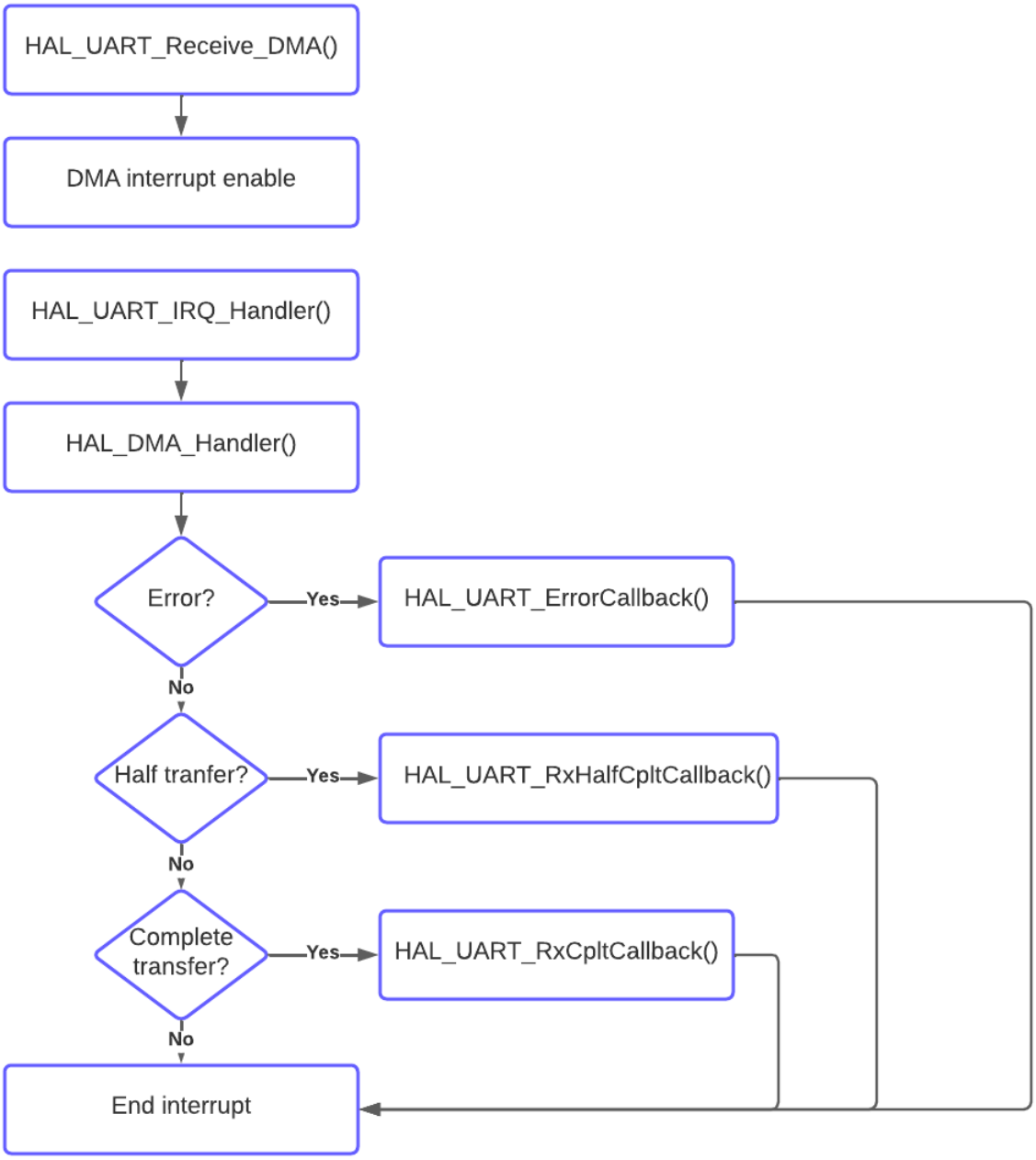
DMA, which stands for direct memory access, is a hardware process that can handle data transfers without processor intervention. Thus, events can happen behind the scene without needing to interrupt the processor. DMA is best suited data transfers since it provides the half call back functions which help the user to change the data during transmitting/receiving process.

Also, HAL library provides other functions to be called when performs the receive/transmit functions. **HAL functions for the DMA are:**

- **HAL_UART_Transmit_DMA():** This function takes three arguments; handler structure, buffer for data to be transmitted, and size of buffer, and it returns HAL_status prototype which it can be HAL_OK or HAL_ERROR.
- **HAL_UART_Receive_DMA():** This function takes three arguments; handler structure, buffer for data to be received, and size of buffer, and it returns HAL_status prototype which it can be HAL_OK or HAL_ERROR.
- **HAL_DMA_Handler():** This function is used to handle the direct memory access for the UART and it is automatically called when HAL_UART_Transmit/Receive_DMA() functions are called.
- **HAL_UART_ErrorCallback():** This function is automatically called when error occurs during transmitting/receiving data and can be edited by the user.
- **HAL_UART_TxHalfCpltCallback():** This function is automatically called when the half of the data is transmitted and can be edited by the user.
- **HAL_UART_RxHalfCpltCallback():** This function is automatically called when the half of the data is received and can be edited by the user.

Also, in DMA mode, the provided functions for the interrupt method are used.

Inside the DMA receive function (and the same in transmit function), it happens the following:



References:

- **Campbell, Scott. “Basics of UART Communication.” Circuit Basics. Keim, Robert.**
- **"STMicroelectronics Reveals Extreme Low-Power STM32U5 Microcontrollers with Advanced Performance and Cybersecurity". www.st.com*. Retrieved Feb 25, 2021*.**
- **STM32 F1 Website; STMicroelectronics.**