April,2023 Semester

ICT 6641
Advanced Embedded System Design

Lecture#4:
Serial (UART) Communication

S. M. Lutful Kabir, *PhD*
Professor, BUET

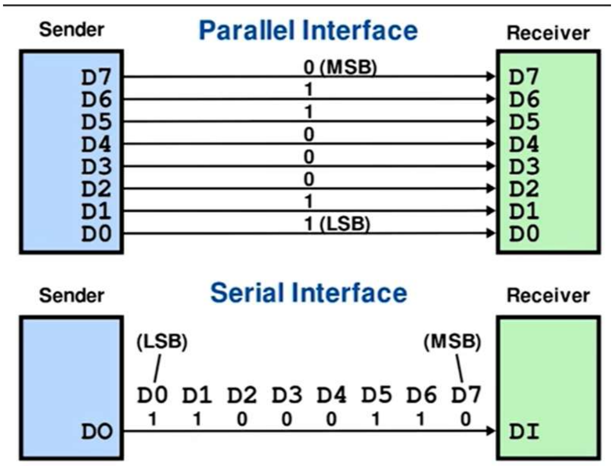
Need for Serial Communication

- Computer transfers data in two ways: Parallel & Serial
- In parallel data transfer, eight or more lines (wire conductors) are used to transfer data to a device.
- Devices that uses parallel data transfer include printers, IDE Hard disk etc.
- Although a lot of data can be transferred in a short amount of time, by using many wires in parallel.
- To transfer many meters away, parallel method is costly, so serial method is used.

Serial Data Transfer

- In serial communication, the data is sent one bit at a time.

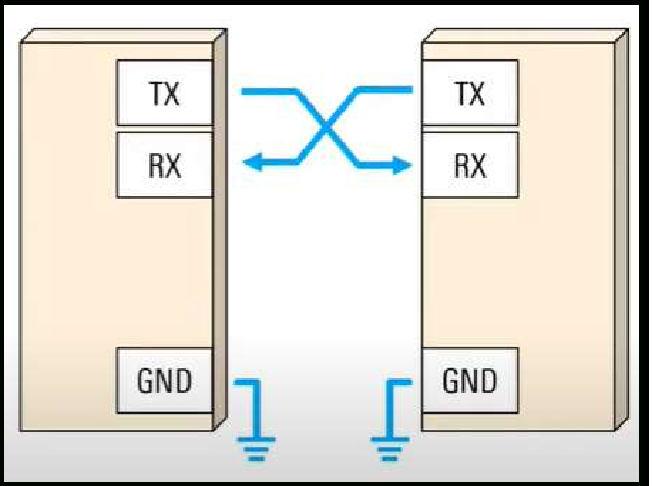
Parallel: expensive - short distance – fast
Serial :cheaper– long distance-slow



Different Protocols of Serial Communication

What is UART?

- Universal Asynchronous Receiver Transmitter
- Protocol (a set of rules) for exchanging serial data between two devices.
- Uses only two wires.
  - Tx to Rx (each direction)
- Can be simplex, half-duplex, duplex.
- Data is transmitted as frames.



Simplex and Duplex Communication

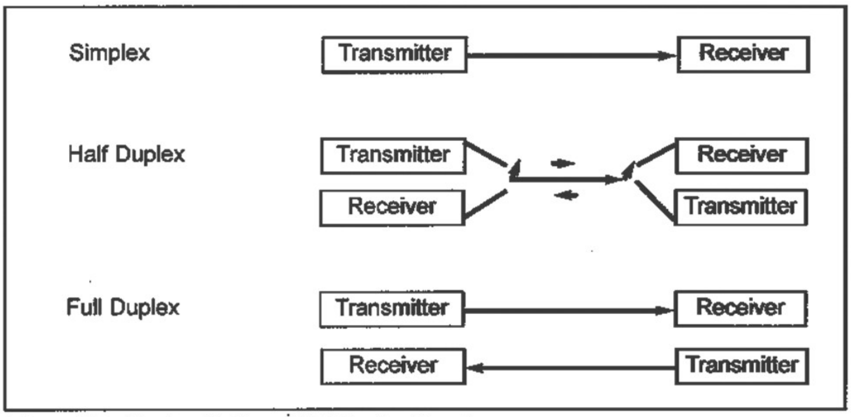


Figure 10-2. Simplex, Half-, and Full-Duplex Transfers

UART versus Other Protocol

- UART is one of the earliest serial protocol – Serial (com) port, RS-232, modem etc.
- One of the most important applications is to display data on the serial console of the computer for debugging or logging important events during program execution on a microcontroller.
- Popularity however is decreasing
  - SPI and I2C between components
  - Ethernet and USB between computers and peripheral
- UART is still important for lower-speed, low-throughput applications.
- Furthermore, many wireless devices such as GSM, GPS, Bluetooth, Xbee, LoRA, and many others provide a serial interface to transfer data between these devices and a microcontroller.

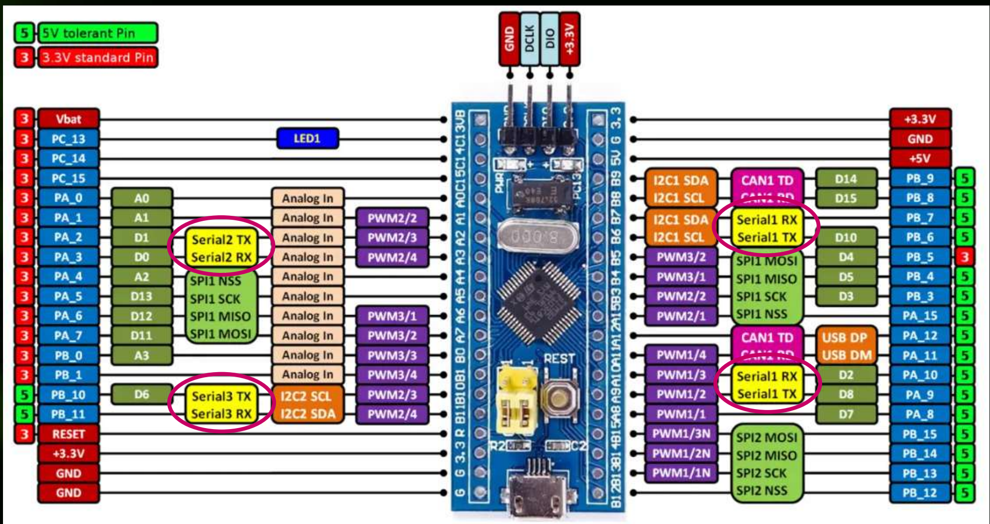
Advantages

- UART is asynchronous – transmitter and receiver do not share the same clock
- The transmitter and the receiver therefore must:
  - Transmit at same speed (baud rate)
  - Use same frame structure/ parameters.

Common UART baud rates
4800
9600
19200
57600
115200

STM32 UART ports

- The internal block diagram of the different modules in STM32F103C8 is shown below. 2 USART1, 1 USRT2 & 1 USART2



Serial Data Transfer

- For serial communication to work, the bytes of data must be converted to serial bits using a parallel-in-serial-out shift register.
- This also means that at the receiving end there must be a serial-in-parallel-out shift register in order to receive the serial data and pack them into a byte.
- For serial transfer, if the distance is short, the 1s and 0s are transferred directly from one device to other.
- For long distance the signal is modulated so that signal does not attenuate.
- This is done by a peripheral device called MODEM (stands for modulation/demodulation).

Methods of Serial Communication

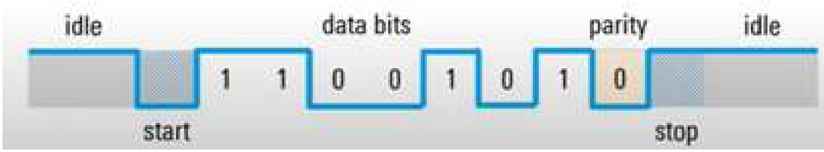
- Serial Communication uses two methods, Synchronous and Asynchronous.
- The Synchronous method transfers a block of data (character) at a time.
- Whereas Asynchronous method transfers a single byte at a time.
- It is possible to write software to use either of the methods, the program can be tedious and long.
- That is why, special IC chips is made by many manufacturers for serial data communication.
- These chips are commonly referred to as Universal Asynchronous Transmitter/Receiver (UART) or Universal Synchronous- Asynchronous Transmitter/ Receiver (USART)
- STM32 has built-in USART .

Protocol

- The data coming in at the receiving end of the data line in a serial data transfer is all 1s and 0s.
- So, it is difficult to make sense of the data unless the sender and the receiver agree on a set of rules, called Protocol, on
  - how the data is packed,
  - how many bits constitute a character, and
  - how the data begins and ends.

Asynchronous Communication and Data Framing

- UART frames consist of
- Start/Stop bits
- Data bits
- Parity bit (Optional)
- High -> 1 and Low->0
- In the idle state, the line is held high
- Example, transmitting ASCII value of the character “A”.



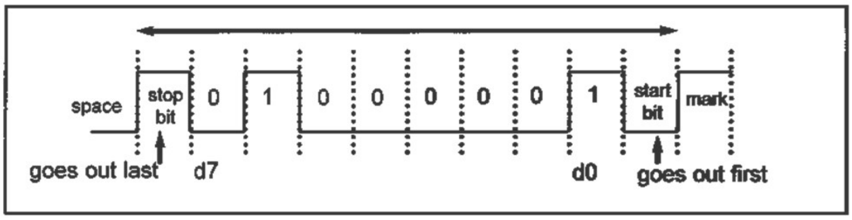


Figure 10-3. Framing ASCII “A” (41H)

Data Frame

- In Asynchronous serial communication, peripheral chips or modem can be programmed for data that is 7 bit or 8 bit wide.
- This is in addition of stop bits, 1 or 2.
- In present days, 10 bits for each character is transmitted: 8 bits for each ASCII code, and 1 bit for each start and stop bits.
- Therefore each 8 bit requires 2 extra bits, which gives 25% overhead.

Parity Bit

- In some systems, the parity bit of the data byte is included in the data frame in order to maintain data integrity.
- This parity bit is odd or even.
- In the case of odd parity number of 1s in the data byte including the parity bit is odd.
- Similarly for even parity, number of 1s in the data byte including the parity bit is even.
- UART/USART chips can be programmed for odd, even or no-parity options.

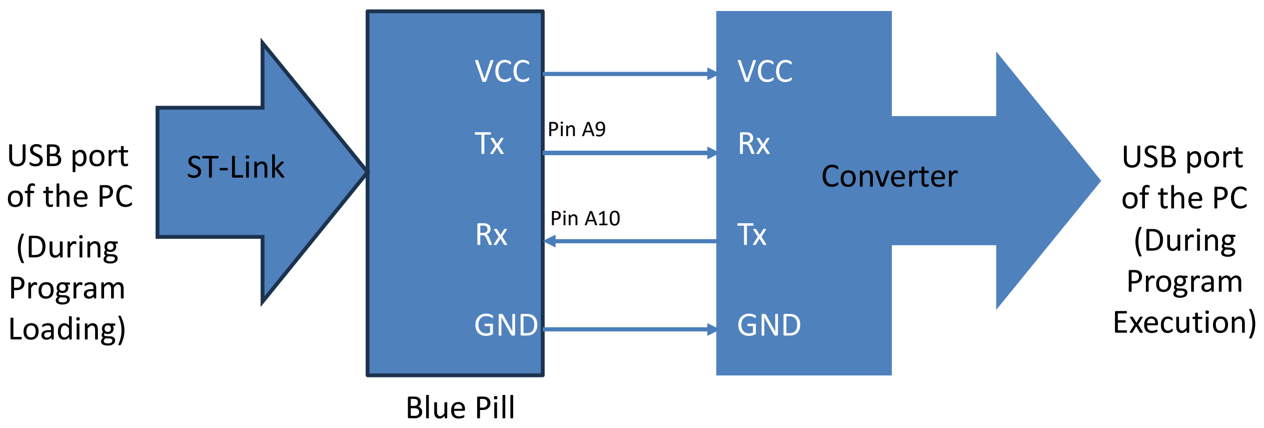
Data Transfer Rate

- The rate of data transfer in serial communication is called bits per second (bps).
- Another widely used terminology is called baud rate.
- However, baud rate and bps rates are not necessarily equal.
- This is because baud rate is used in modem and is defined as the number of signal changes per second.
- A single change of signal may transfer several bits of data
- As far as conductor wire is concerned, the baud rate and bps are the same and for this reason we shall use them interchangeably.

Serial Print Using UART communication

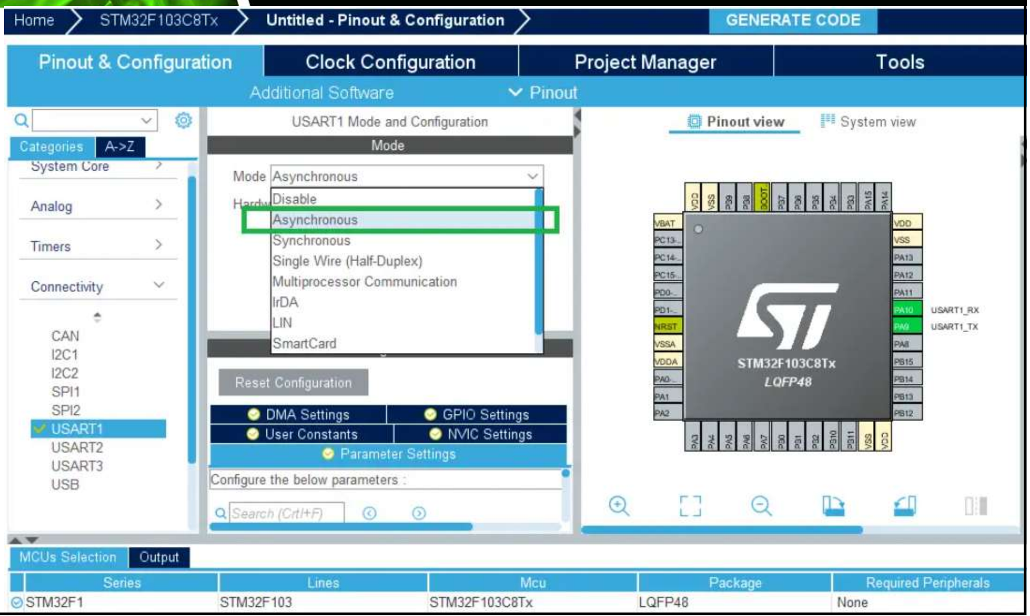
- The UART peripherals in the microcontroller can be used to send serial data to the PC serial COM port and display it on a terminal using a USB-TTL converter board.
- Hence, you're not restricted to use a specific one (UART1, UART2, or UART3).
- We'll use the UART1 module to send the serial data for debugging.
- The STM32F103C8 microcontrollers' pins are not all 5v tolerant.
- You must be careful when receiving input signals from the USB-TTL converter.
- You can send a 3.3v signal from the MCU TX pin to the USB-TTL RX pin and still get the data identified absolutely fine.
- However, it won't work the other way around without shifting the signal's level.
- The pins for UART1 & UART3 are 5v tolerant while UART2 is not.

Connection of the Converter with the uC module



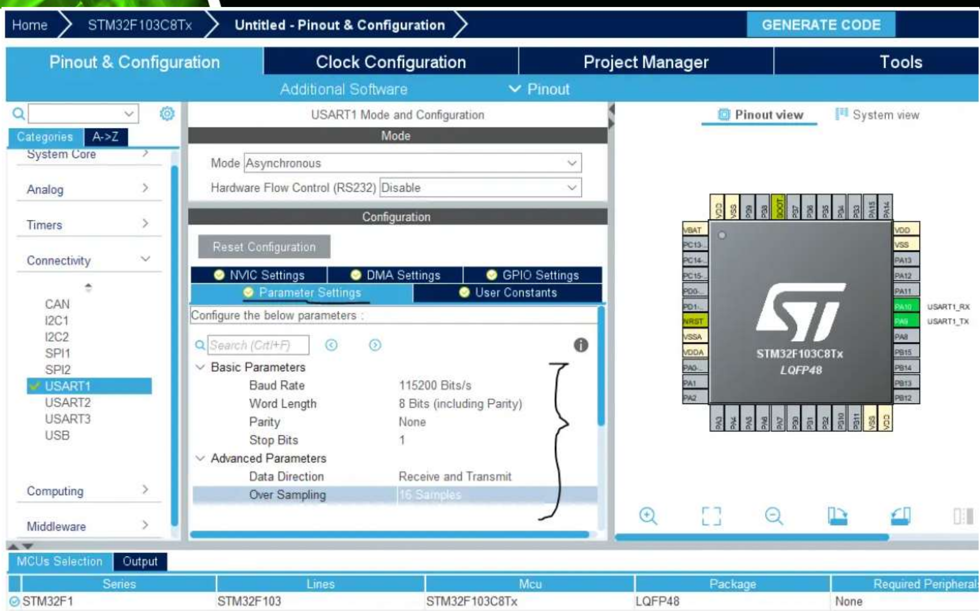
Step#1-#3

- Step1: Open Cube IDE & Create New Project
- Step2: Choose The Target MCU & Double-Click Its Name
- Step3: Enable USART1 Module (Asynchronous Mode)



Step #4

- Step4: Choose The Desired Settings For UART (Baud Rate, Stop Bits, Parity, etc..)



Step #5-#8

- Step5: Goto The RCC Options Tab & Enable External Crystal
- Step6: Go To The Clock Configuration & Set The System Clock To 72MHz
- Step7: Generate The Initialization Code & Open The Code
- Step8: Write The Code For Your Project & Use HAL_UART_Transmit() To Print

```
#include "main.h"

UART_HandleTypeDef huart1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);

int main(void)
{
  uint8_t MSG[35] = {'\0'};
  uint8_t X = 0;
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  while (1)
  {
    sprintf(MSG, "Hello! Tracing X = %d\r\n", X);
    HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    HAL_Delay(500);
    X++;
  }
}
```

Max length of time (mS) until which the function will wait for the completion of transmission.

Step #9 - #11

- Step9: Build & Debug To Flash The Code
- Step10: Go To The Device Manager & Check The USB-TTL COM Port Num.
- Step11: Open The Terminal From CubeIDE

  Window > Show View > Console

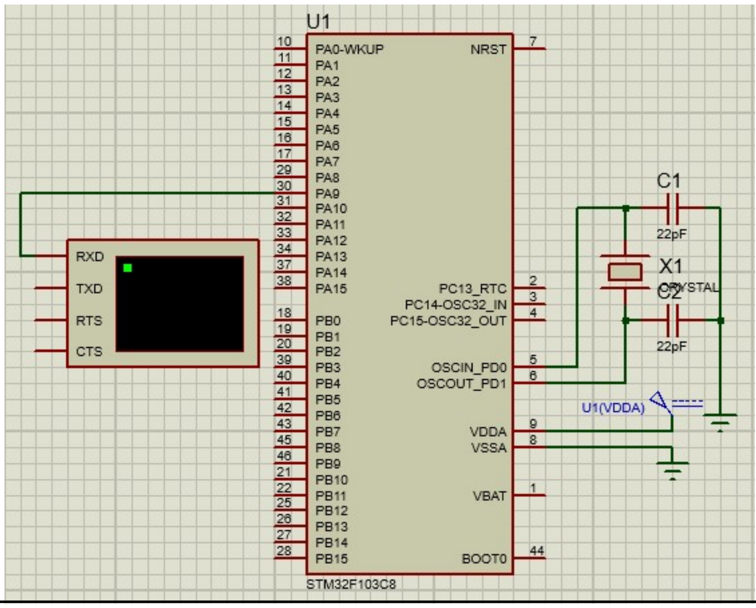
  In Console:

  click on the NEW icon on its menu bar > Command Shell console > Connection type: Serial port > set Baud Rate & Connection Name > Encoding: UTF-8 > And Click OK!
- **Now Download the program in your Blue Pill board and Observe the result.**

Setting the BIOS Serial Console Baud Rate

1. From the System Utilities screen, select System Configuration > BIOS/Platform Configuration (RBSU) > System Options > BIOS Serial Console & EMS > BIOS Serial Console Baud Rate.
2. Select a setting: 9600/ 19200/ 57600/ 115200/ 38400.
3. Save your setting.

Simulation in Proteus



Another Example of UART Communication

- In this example, we'll be doing PC interfacing via the serial port using the USB-TTL converter and UART module in the STM32F103C8 microcontroller (Blue Pill Board).
- We'll send and receive asynchronous UART data from and to the PC.
- For the example,
  - Configure GPIO input pin (for push-button) & output pins (for LEDs)
  - Configure UART in asynchronous mode @ 9600 bps + Enable RX interrupts
  - Read the button state and send it to the PC via serial port
  - Read the received characters from the PC and decide which led is to be toggled.

Step #1-#8

- Step1: Open CubeMX & Create New Project
- Step2: Choose The Target MCU & Double-Click Its Name
- Step3: Enable USART1 Module (Asynchronous Mode)
- Step4: Choose The Desired Settings For UART (Baud Rate, Stop Bits, Parity, etc..)
  - Set the baud rate to 9600 bps
  - Enable UART global interrupts in NVIC tab
- Step5: Configure The Required GPIO Pins For This Project
  - PB12, PB13: Output Pins (For LEDs)
  - PB14: Input Pin (For The Push Button)
- Step6: Goto The RCC Options Tab & Enable External Crystal
- Step7: Go To The Clock Configuration & Set The System Clock To 72MHz
- Step8: Generate The Initialization Code & Open The Project In CubeIDE

Step #9: Include the following code within the generated code

HAL_UART_RxCpltCallback(): When data reception is finished, it will be called by interrupt handle function

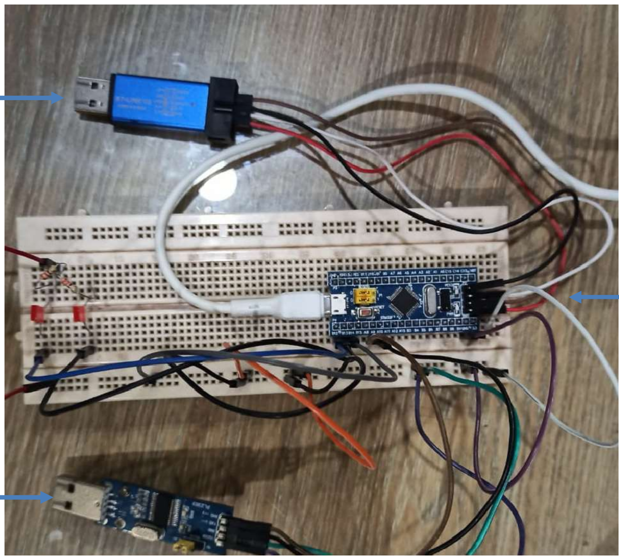
```c
#include "main.h"
UART_HandleTypeDef huart1;
uint8_t RX1_Char = 0x00;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
// [ UART Data Reception Completion CallBackFunc. ]
void HAL_USART_RxCpltCallback(UART_HandleTypeDef *huart) {
   HAL_UART_Receive_IT(&huart1, &RX1_Char, 1);
 }
 int main(void) {
    uint8_t MSG1[] = "Button State: Released\r\n";
    uint8_t MSG2[] = "Button State: Pressed\r\n";
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    HAL_UART_Receive_IT(&huart1, &RX1_Char, 1);
while(1) {
   // Read the button state and sent it via UART
if(HAL_GPIO_ReadPin (GPIOB, GPIO_PIN_14)) {
  HAL_UART_Transmit(&huart1, MSG2, sizeof(MSG2), 100);
 }
else {
  HAL_UART_Transmit(&huart1, MSG1, sizeof(MSG1), 100);
 }
 //--------[ Read The Received Character & Toggle LEDs]----
 if(RX1_Char == '1') {
   HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_12);
   HAL_UART_Receive_IT(&huart1, &RX1_Char, 1);
   RX1_Char = 0x00;
  }
  if(RX1_Char == '2') {
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
    HAL_UART_Receive_IT(&huart1, &RX1_Char, 1);
    RX1_Char = 0x00;
   }
  HAL_Delay(100);
}  ← end of while loop
}  ← end of main()
```

Step #11-#16

- Step11: Build & Debug To Flash The Code
- Step12: Go To The Device Manager & Check The USB-TTL COM Port Num.
- Step13: Open The Terminal From CubeIDE as described in the earlier example.
- Step 14: Compile the project.
- Step 15: Download the hex file into the Blue pill board using ST-Link utility.
- Step 16: Observe the output of the program.

Implementation in Hardware

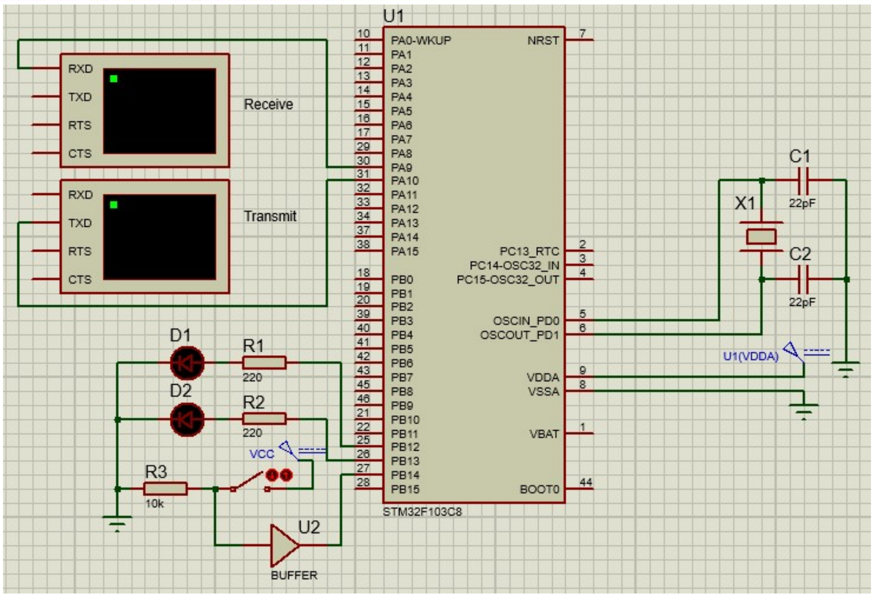
ST-Link



Blue Pill

USB to TTL

Simulation in Proteus



Assignment #3

- Assignment Statement:
  - Develop a project which will posses the following features:
    - PC will send the command to a uC
    - uC will execute the command
    - uC will test whether the execution has worked properly
    - uC will send the result to the computer.
- Connection:
  - For the above project, the uC will have a LED connected to a GPIO pin (pin#1) , and another GPIO pin (pin#2) which will read the state of the previous pin.
- Programming logic:
  - The PC will send '1' for making ON or '0' for making the LED OFF.
  - After receiving the command uC will send send 1/0 to the LED.
  - Then uC will read the pin#1 by pin#2 and send message 'ON' or 'OFF' as the later reads.

Today's Lab Exercises

- 6 groups, each consisting of 3 students. (constituents are given in the next slide).
- Today, there will be 4 experiments to be done.
- Part 1: Expt. on input
- Part 2: Expt. on interrupt
- Part 3a: Expt. on serial print
- Part 3b: Expt. on serial transmit and receive
- Relevant documents are kept in \Desktop\Experiment1

Instructions:

- Create a folder in each PC as **/Desktop/ICT6411/April2023/Projects/Groupn/Exptx**
- Store your works in the folder as indicated above.

Groups

- Group#1: Shafak, Raihan & Kawsar
- Group#2: Riad, Shahria & Tamim
- Group#3: Jahirul, Shahanaz & Amir
- Group#4: Manas, Tousif & Shafikul
- Group#5: Tusher, Samuel & Sourav
- Group#6: Tahmid, Redwan & Shahidul

Thanks