ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN ĐIỆN TỬ
BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

----------o0o----------

BÁO CÁO BÀI TẬP LỚN 1
LẬP TRÌNH STM32 VỚI GIAO THỨC UART

Môn Đo lường điều khiển bằng máy tính

STT	HỌ VÀ TÊN	MSSV
1	Đoàn Huỳnh Quát	1914815

GVHD: Nhóm L02

TP. HỒ CHÍ MINH, THÁNG 1 NĂM 2022

MỤC LỤC

I.STM32F103C8T6

STM32F103C8T6 là vi điều khiển 32bit, thuộc họ F1 của dòng chip STM32 hãng ST.

- Lõi ARM COTEX M3.

- Tốc độ tối đa 72Mhz.

- Bộ nhớ :

64 kbytes bộ nhớ Flash

20 kbytes SRAM

- Clock, reset và quản lý nguồn

Điện áp hoạt động từ 2.0 → 3.6V.

Sử dụng thạch anh ngoài từ 4Mhz → 20Mhz.

Thạch anh nội dùng dao động RC ở mode 8Mhz hoặc 40Khz.

- Chế độ điện áp thấp:

Có các mode: ngủ, ngừng hoạt động hoặc hoạt động ở chế độ chờ.

Cấp nguồn ở chân Vbat bằng pin ngoài để dùng bộ RTC và sử dụng dữ liệu được lưu trữ khi mất nguồn cấp chính.

- 2 bộ ADC 12 bit với 9 kênh cho mỗi bộ

Khoảng giá trị chuyển đổi từ 0 – 3.6 V

Có chế độ lấy mẫu 1 kênh hoặc nhiều kênh.

- DMA:

7 kênh DMA

Có hỗ trợ DMA cho ADC, UART, I2C, SPI.

- 7 bộ Timer:

3 Timer 16 bit hỗ trợ các mode Input Capture/ Output Compare/ PWM.

1 Timer 16 bit hỗ trợ để điều khiển động cơ với các mode bảo vệ ngắt Input, dead-time.

2 Watchdog Timer để bảo vệ và kiểm tra lỗi.

1 Systick Timer 24 bit đếm xuống cho hàm Delay,….

- Có hỗ trợ 9 kênh giao tiếp:

2 bộ I2C.

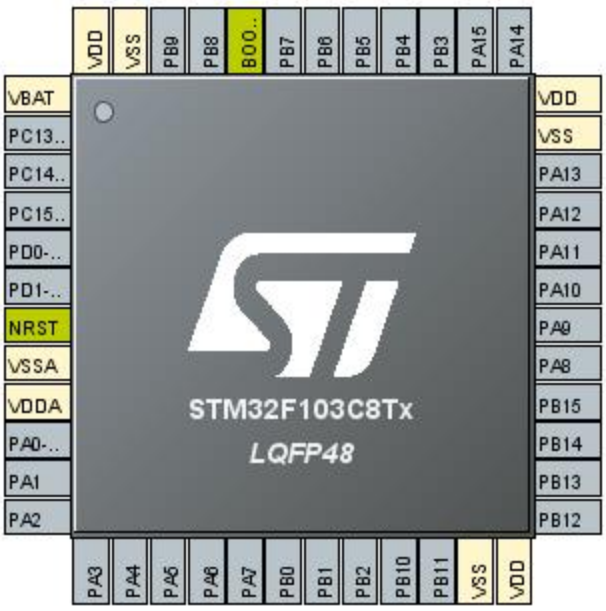
3 bộ USART

2 SPI

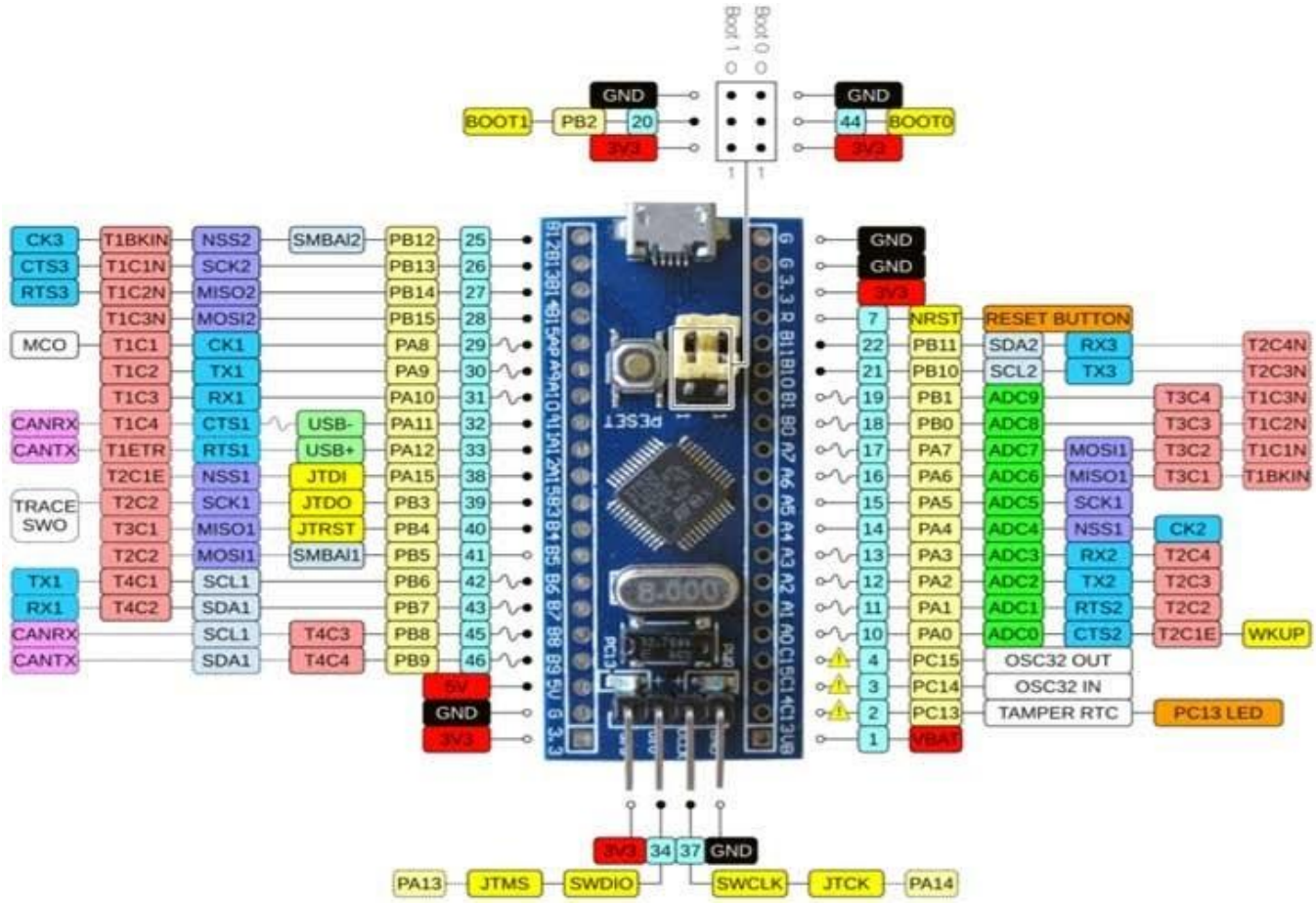
1 CAN

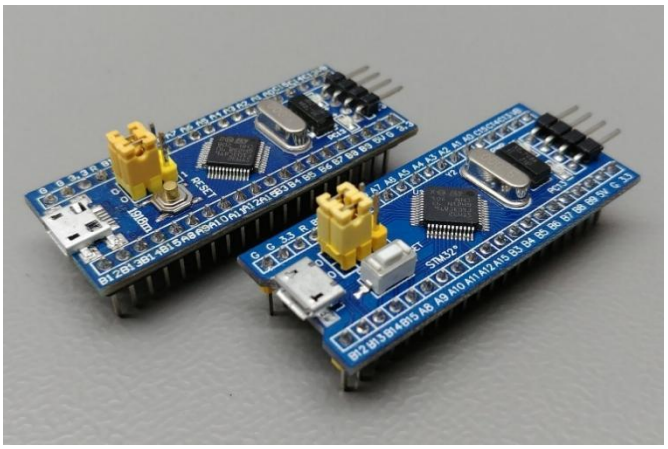
USB 2.0 full-speed interface

- Kiểm tra lỗi CRC và 96-bit ID.



Kit phát triển STM32F103C8T6 Blue Pill ARM Cortex-M3 là loại được sử dụng để nghiên cứu về ARM nhiều nhất hiện nay.





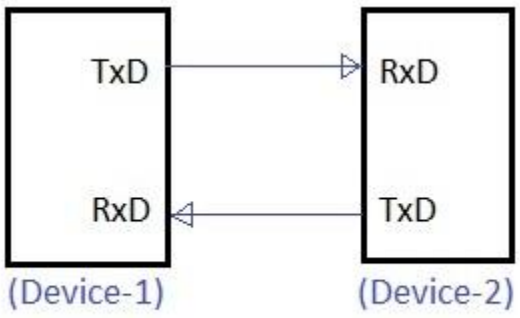
Các thông số kĩ thuật:

- Điện áp cấp 5VDC qua cổng Micro USB sẽ được chuyển đổi thành 3.3VDC qua IC nguồn và cấp cho Vi điều khiển chính.
- Tích hợp sẵn thạch anh 8Mhz.
- Tích hợp sẵn thạnh anh 32Khz cho các ứng dụng RTC.
- Ra chân đầy đủ tất cả các GPIO và giao tiếp: CAN, I2C, SPI, UART, USB,...
- Tích hợp Led trạng thái nguồn, Led PC13, Nút Reset.
- Kích thước: 53.34 x 15.24mm.

II.Giao thức Uart

UART tiếng anh là Universal Asynchronous Reciver/Transmister một chuẩn giao tiếp không đồng bộ cho MCU và các thiết bị ngoại vi. là một ngoại vi cơ bản của STM32 sử dụng 2 chân Rx và Tx để nhận và truyền dữ liệu.

Chuẩn UART là chuẩn giao tiếp điểm và điểm, nghĩa là trong mạng chỉ có hai thiết bị đóng vai trò là transmister hoặc reciver.



UART Interface Diagram

Cách hoạt động của giao thức UART

UART là giao thức truyền thông không đồng bộ, nghĩa là không có xung Clock, các thiết bị có thể hiểu được nhau nếu các Setting giống nhau

UART là truyền thông song công(Full duplex) nghĩa là tại một thời điểm có thể truyền và nhận đồng thời. UART truyền dữ liệu không đồng bộ, có nghĩa là không có tín hiệu để đồng bộ hóa đầu ra của các bit từ UART truyền đến việc lấy mẫu các bit bởi UART nhận. Thay vì tín hiệu đồng bộ, UART truyền thêm các bit start và stop vào gói dữ liệu được chuyển. Các bit này xác định điểm bắt đầu và điểm kết thúc của gói dữ liệu để UART nhận biết khi nào bắt đầu đọc các bit.

Trong đó quan trọng nhất là Baund rate (tốc độ Baund) là khoảng thời gian dành cho 1 bit được truyền. Phải được cài đặt giống nhau ở gửi và nhận. Một số Baud Rate thông dụng: 9600, 38400, 115200, 230400,….

Sau đó là định dạng gói tin.

Định dạng gói tin như sau:

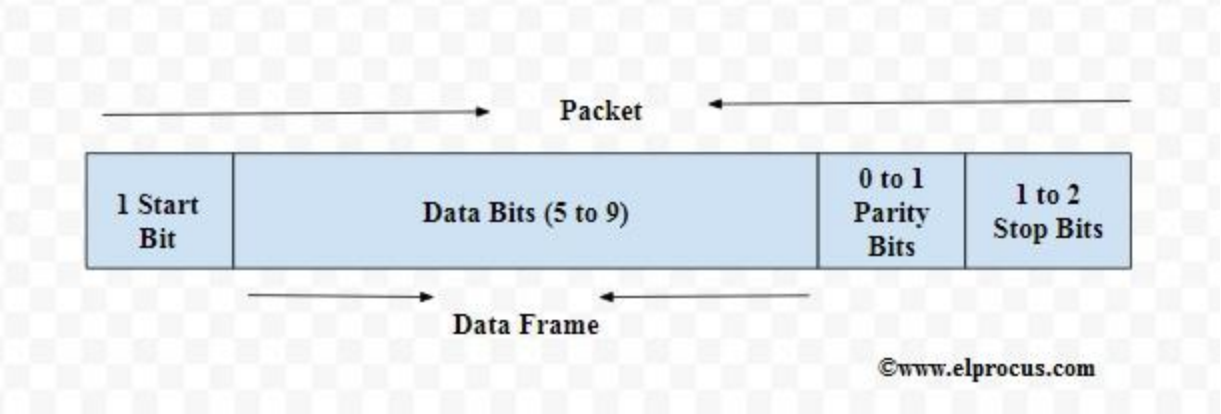
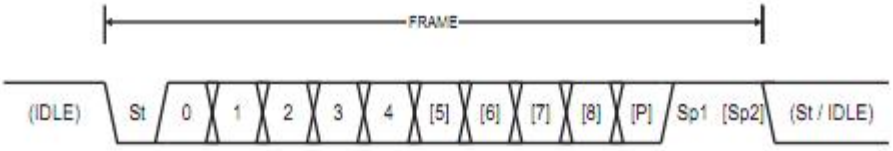


Figure 19-4. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Start – Bit

Start-bit còn được gọi là bit đồng bộ hóa được đặt trước dữ liệu thực tế. Nói chung, một đường truyền dữ liệu không hoạt động được điều khiển ở mức điện áp cao. Để bắt đầu truyền dữ liệu, truyền UART kéo đường dữ liệu từ mức điện áp cao (1) xuống mức điện áp thấp (0). UART thu được thông báo sự chuyển đổi này từ mức cao sang mức thấp qua đường dữ liệu cũng như bắt đầu hiểu dữ liệu thực. Nói chung, chỉ có một start-bit.

Stop – Bit

Bit dừng được đặt ở phần cuối của gói dữ liệu. Thông thường, bit này dài 2 bit nhưng thường chỉ sử dụng 1 bit. Để dừng sóng, UART giữ đường dữ liệu ở mức điện áp cao.

Partity Bit

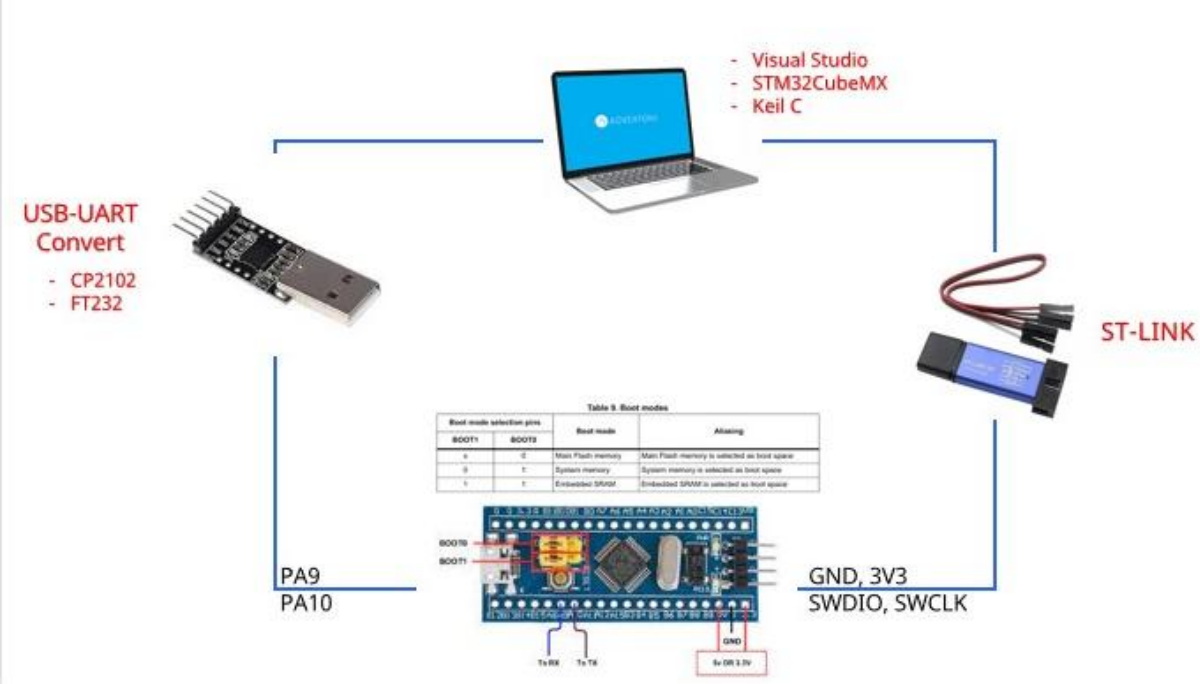
Bit chẵn lẻ cho phép người nhận đảm bảo liệu dữ liệu được thu thập có đúng hay không. Đây là một hệ thống kiểm tra lỗi cấp thấp & bit chẵn lẻ có sẵn trong hai phạm vi như Chẵn lẻ – chẵn lẻ cũng như Chẵn lẻ – lẻ. Trên thực tế, bit này không được sử dụng rộng rãi nên không bắt buộc.

Data frame

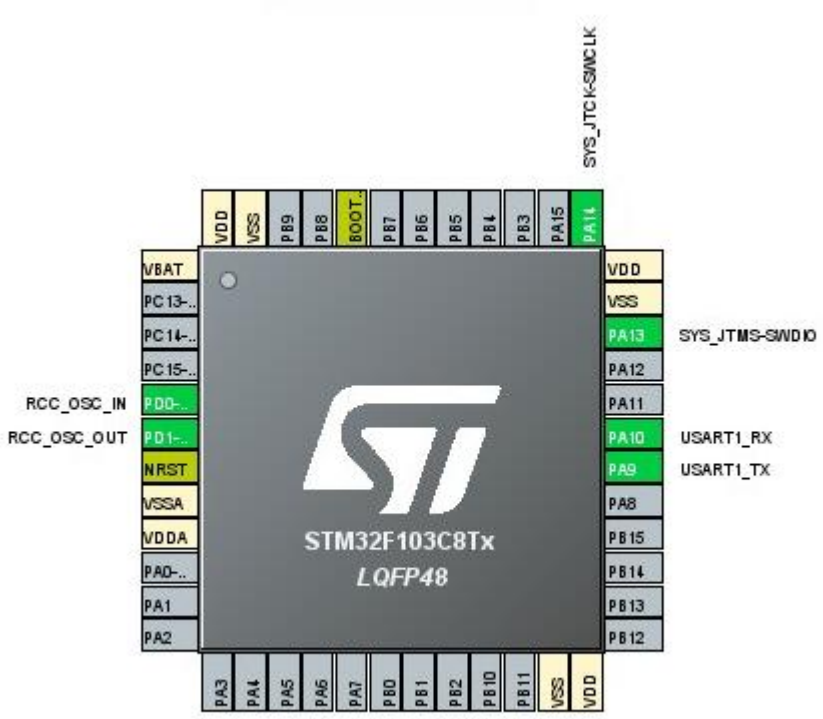
Các bit dữ liệu bao gồm dữ liệu thực được truyền từ người gửi đến người nhận. Độ dài khung dữ liệu có thể nằm trong khoảng 5 & 8. Nếu bit chẵn lẻ không được sử dụng thì chiều dài khung dữ liệu có thể dài 9 bit. Nói chung, LSB của dữ liệu được truyền trước tiên sau đó nó rất hữu ích cho việc truyền.

III.Firmware.

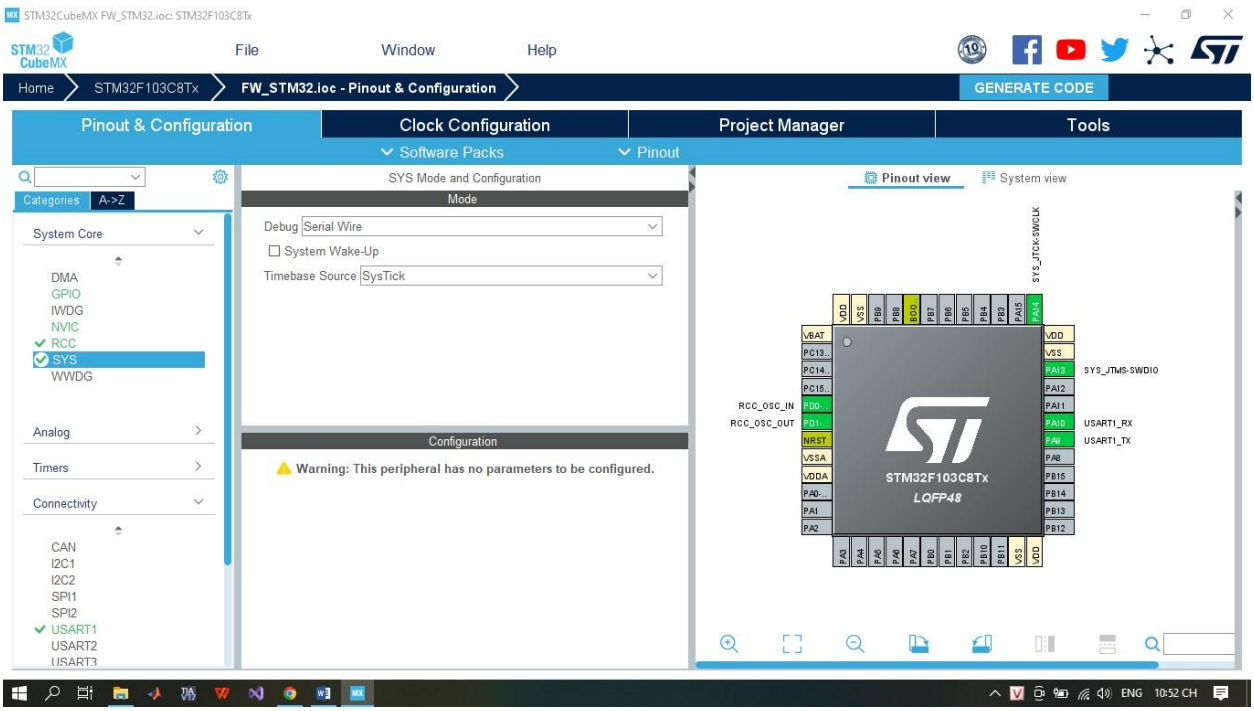
3.1 STM32CUBEMX



Mô hình thực hiện giao tiếp uart

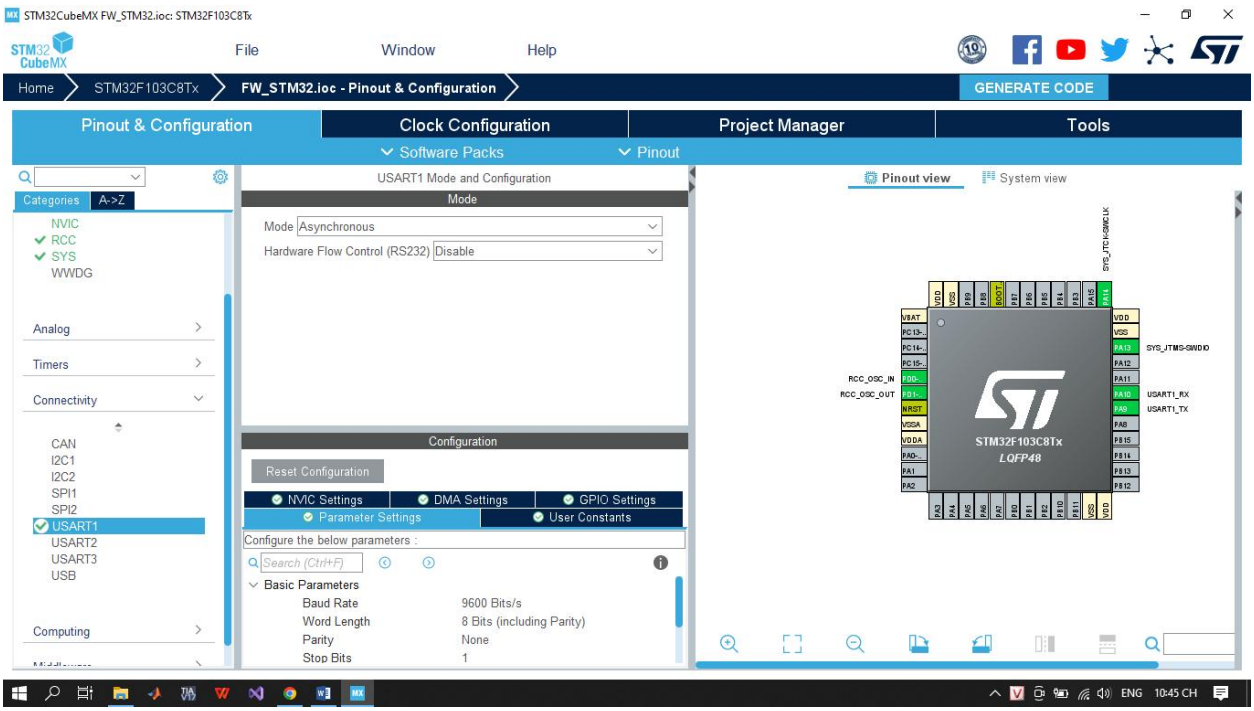


Thực hiện config trên STM32CUBEMX



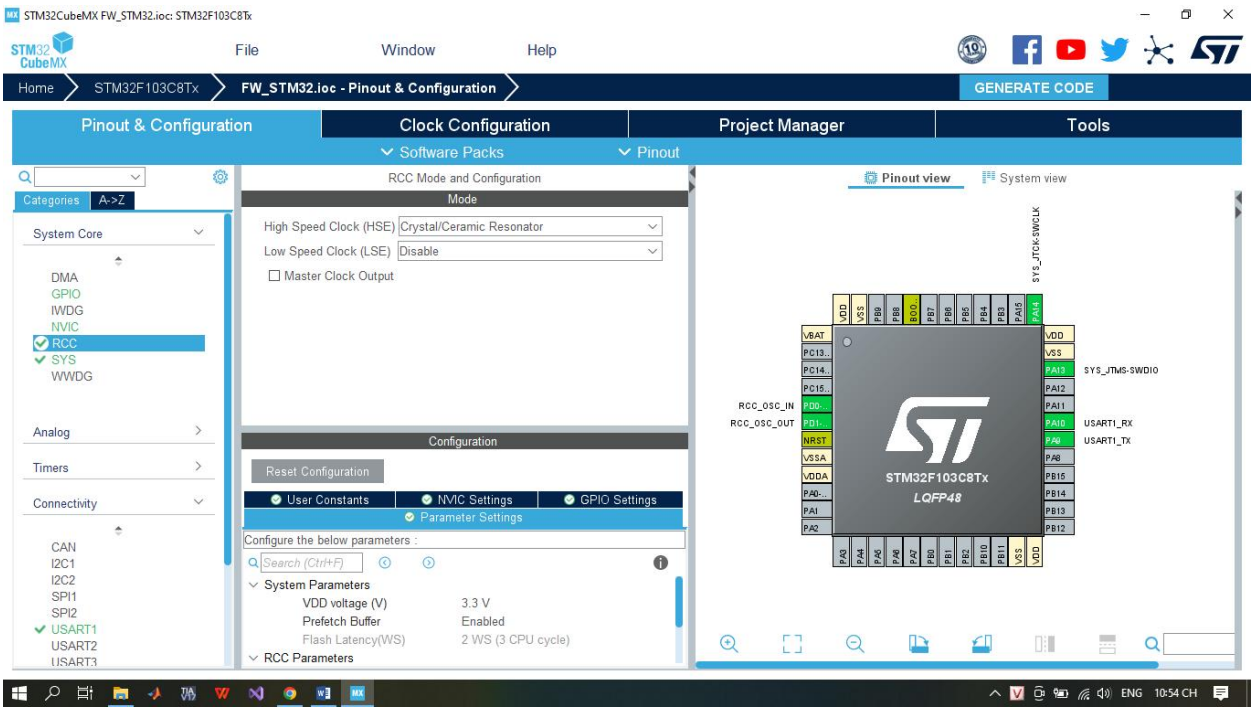
Đầu tiên mình config chân debug là serial wire. Các bạn chọn System Core -> SYS -> Debug: Serial Wire để vi điều khiển được cấu hình nạp code thông qua chân SWDIO và SWCLK (chúng ta sử dụng mạch nạp ST- Link và kết nối với vi điều khiển STM32F103C8T6 thông qua các chân này).

Config uarrt

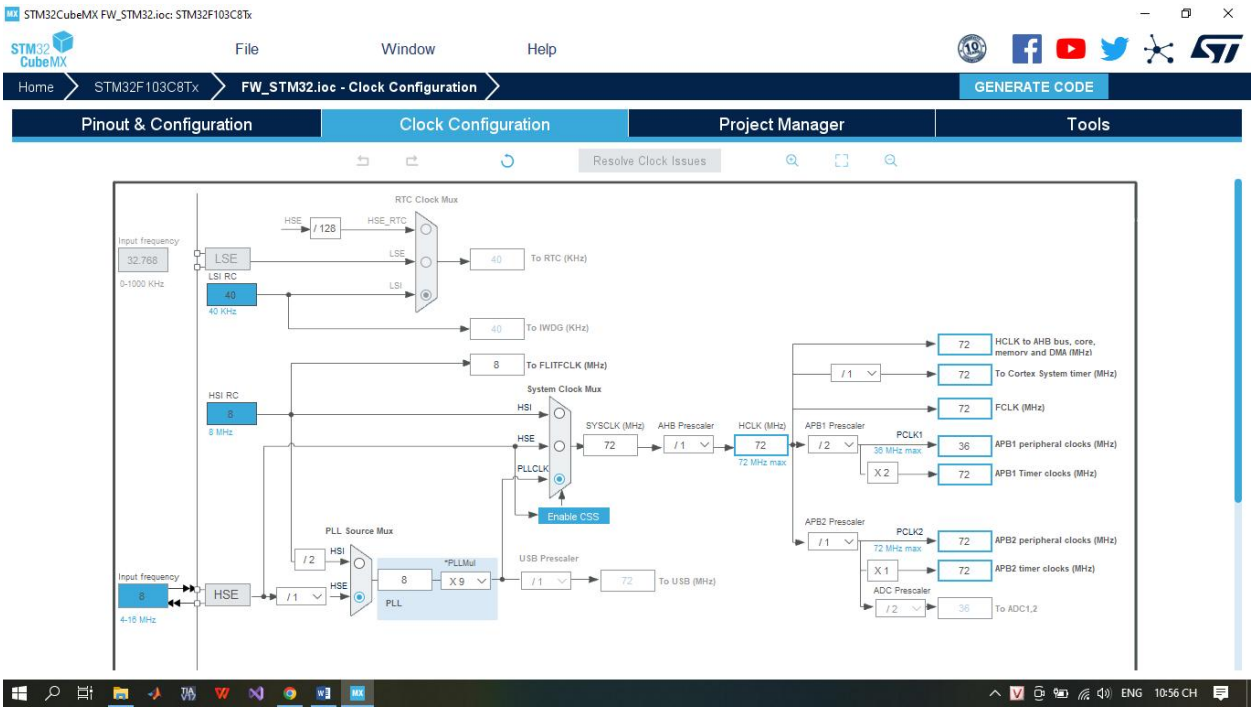


Chọn mode Asynchronous, tốc độ baud là 9600 bits/s, PA9 là chân transmit, PA10 là chân nhận.

Vào NVIC kích hoạt Uart enable interrupt để chương trình sử dụng interrupt đó và nhận dữ liệu bên ngoài truyền vào thông qua interrupt.



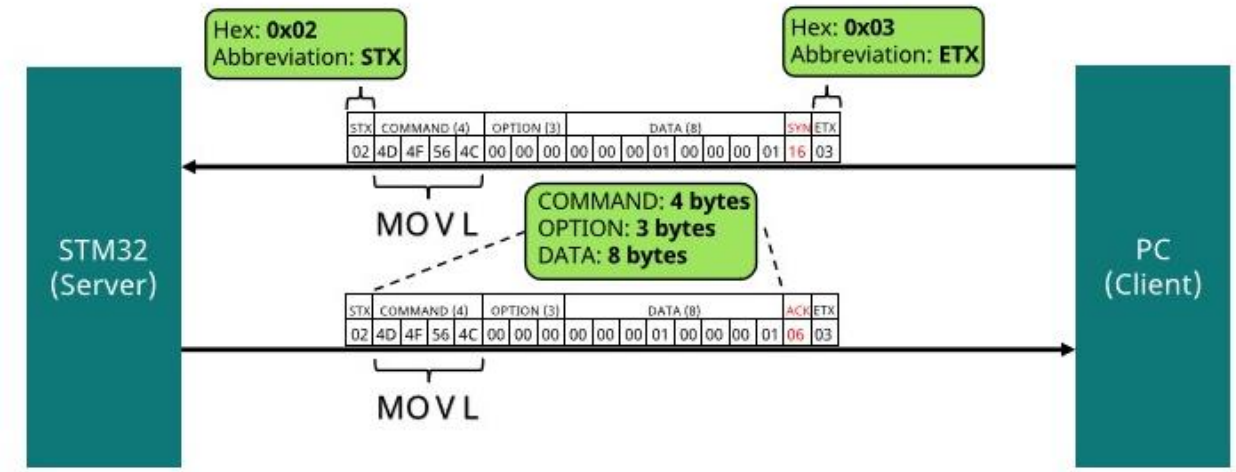
RCC: HSEsử dụng clock ngoài vì clock ngoài chính xác hơn mặc dù có thạch anh bên trong vxl.



Cầu hình clk

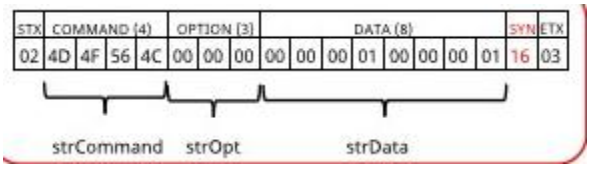
Chọn thạch anh ngoài là 8Mhz, vì thạch anh bên ngoài của vi xử lý là 8Mhz. Chọn các thanh chia để cấu hình HCLK là 72MHz. sau đó chia tiếp để cầu hình clock cho Timer1, timer2 và ngoại vi.

3.2 KEIL C



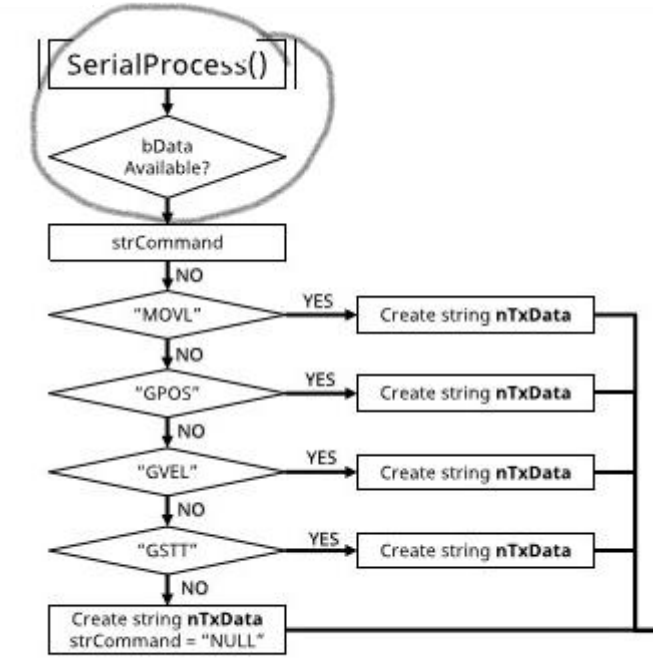
Đây là Frame truyền với bắt đầu với 1 byte 02 và kết thúc 03. 4 byte Command là mã lệnh mà pc muốn slave thực hiện một tác vụ nào đó. Khi STM nhận mã lệnh này nó sẽ biết Pc muốn nó làm gì đó. Option có 3 byte là phần mổ rộng của command. Sau đó là 8 byte dữ liệu của command. Có 1 byte truyền là SYN byte nhận là ACK tưc là nó sẽ thông báo cho server là tôi đã nhận hoặc muốn đưa 1 lệnh.

```
76  bool WriteComm(uint8_t *pBuff, uint8_t nSize)
77  {
78    return HAL_UART_Transmit(&huart1, pBuff, nSize, 1000);
79  }
80
81  bool ReadComm(uint8_t *pBuff, uint8_t nSize)
82  {
83    if ((pBuff[0] == STX[0]) && (pBuff[17] == ETX[0]))
84    {
85      memcpy(strCommand, subString(pBuff, 1, 4), 4);
86      memcpy(strOpt, subString(pBuff, 5, 3), 3);
87      memcpy(strData, subString(pBuff, 8, 8), 8);
88
89      bDataAvailable = true;
90    }
91    else
92    {
93      bDataAvailable = false;
94    }
95
96    return bDataAvailable;
97  }
98
```



Hàm writeComm để ghi dữ liệu ra serial. Readcomm để nhận dữ liệu. Dữ liệu đi vào liên tục sẽ được lưu trong biến con trỏ pBuff, nó sẽ chép từng byte vào. Và để nhận dữ liệu cho đúng thì có câu lệnh điều kiện nếu bit đầu của pBuff bằng STX[0] là cái byte 02 và PBuff[17] là byte 03 (ETX[0]) thì tức là nếu chuỗi dữ liệu bắt đầu từ 02 và kết thúc là 03 thì frame truyền đã nhận thành công. Sau đó phân tách protocol này ra bằng lệnh substring.

Đưa cái cờ bDataAvailable lên true.

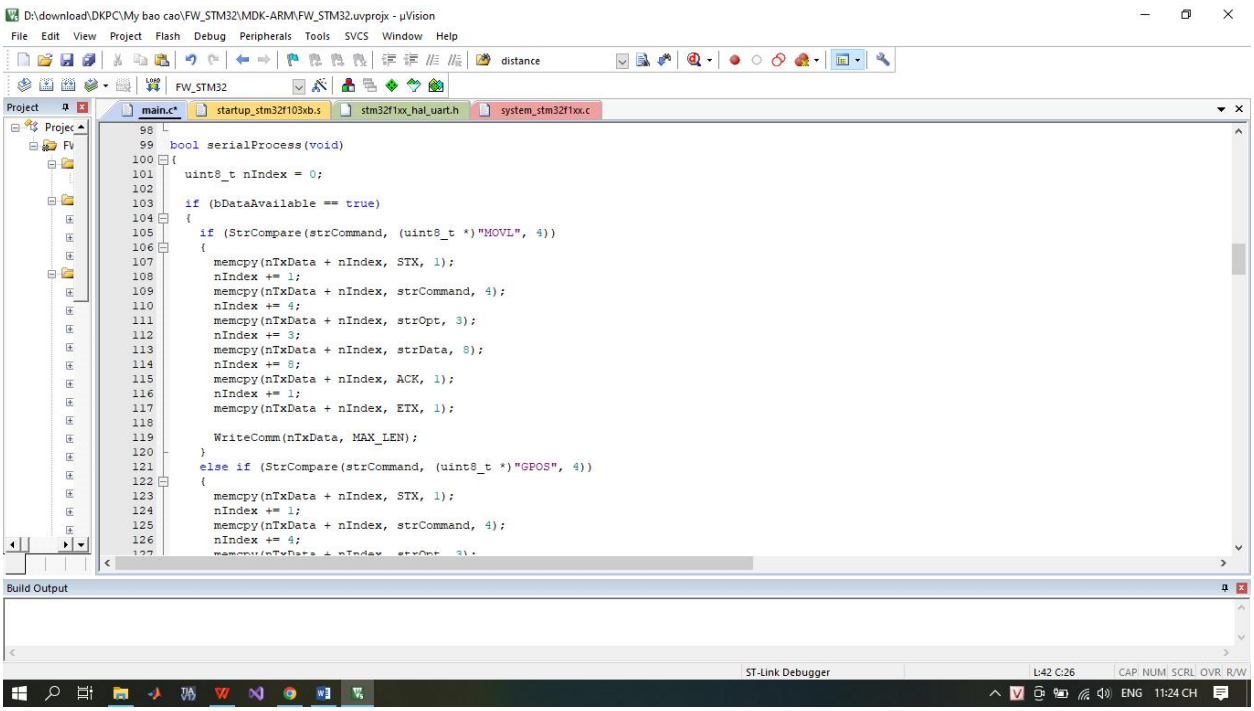


Hàm HAL_UART_RxCpltCallback()

```
194
195  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
196 {
197   if (huart->Instance == huart1.Instance)
198   {
199     ReadComm(nRxData, MAX_LEN);
200
201     HAL_UART_Receive_IT(&huart1, (uint8_t *)nRxData,MAX_LEN);
202   }
203 }
```

Là khai báo Uart nếu điều kiện huart Instance thỏa nó sẽ gọi ReadComm để bắt dữ liệu

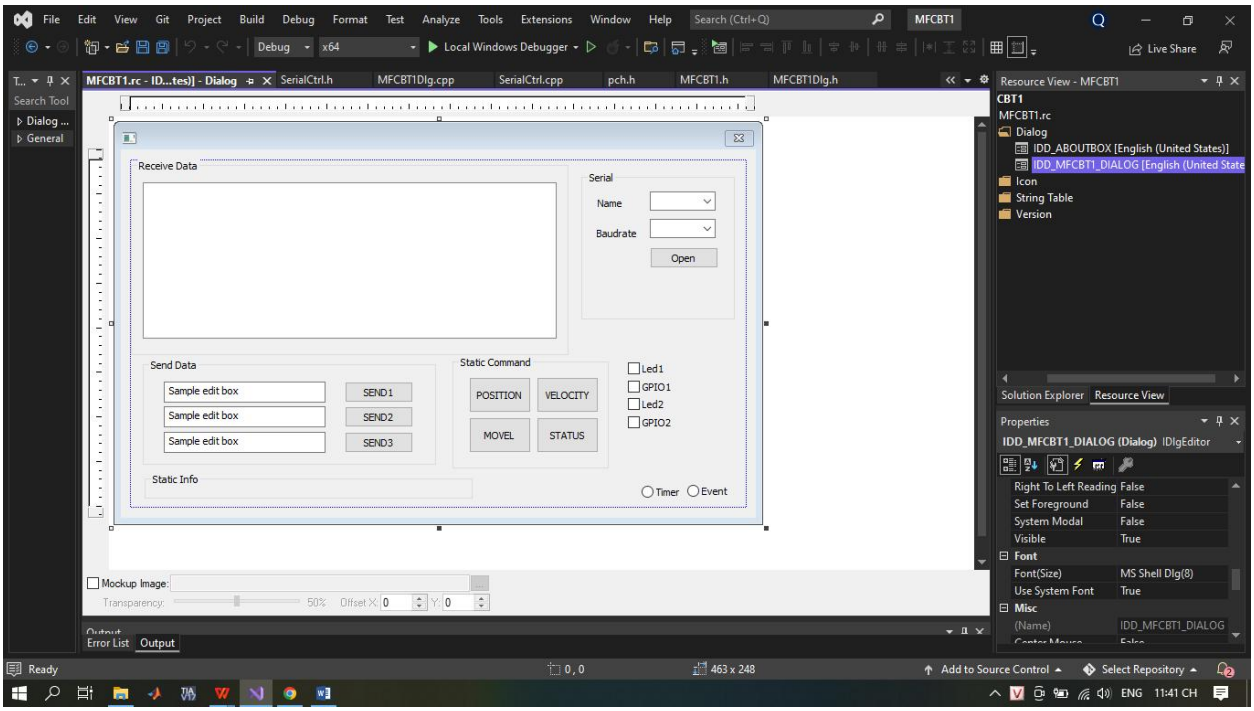
Hàm Serial Process :



Kiểm tra interrupt có nhận đúng chuỗi dữ liệu mình protocol hay không, nếu nó nhận đúng dữ liệu như trCommand bằng MOVL (4 byte) nó sẽ gửi ngược lại cho pc để thông báo nó nhận được frame truyền có lệnh là MOVL .

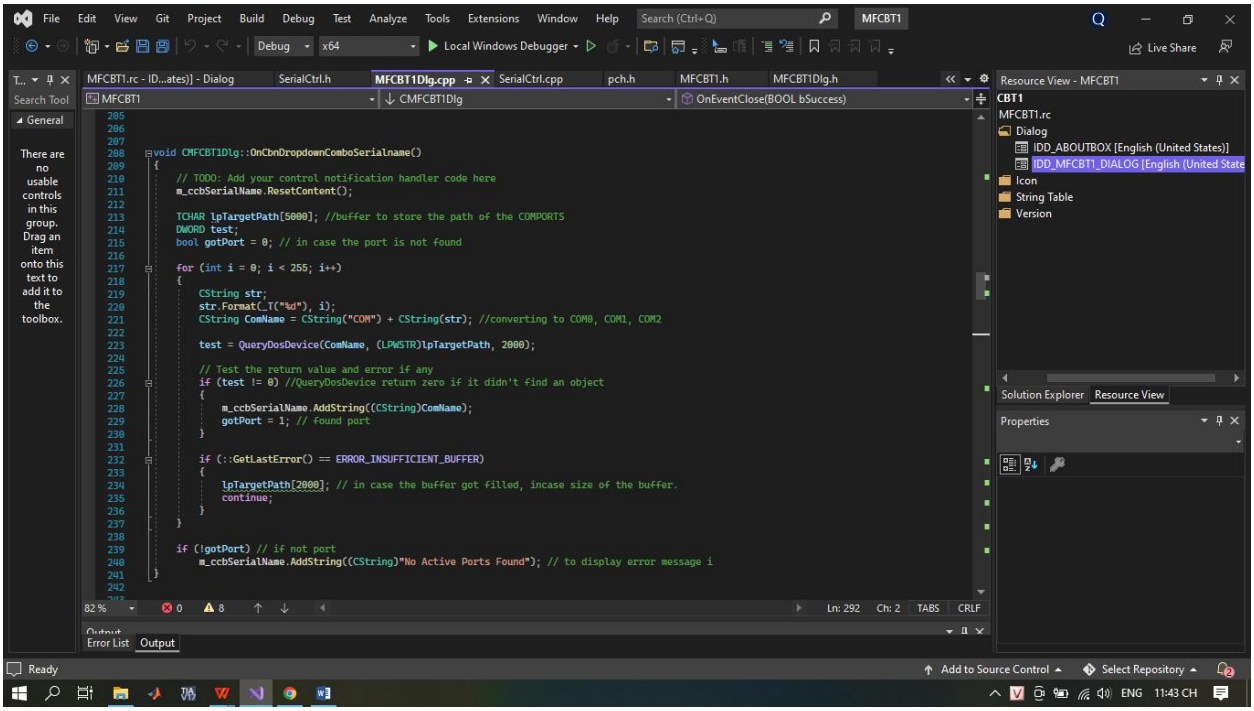
IV. Software

Lập trình gui với C++



Tạo ra giao diện giao tiếp bằng toolbox

Chương trình để điều khiển



Kiểm tra kết nối cổng COM, nếu test là zero thì không tìm thấy đối tượng kết nối

```
void CMFCBT1Dlg::OnBnClickedButtonSend1()
{
    // TODO: Add your control notification handler code here
    CString Cmd;
    if (!GetPortActivateValue()) return;

    m_ceSendCmd1.GetWindowText(Cmd);

    TCHAR* cmd = (LPTSTR)(LPCTSTR)Cmd;
    Write((char*)cmd, Cmd.GetLength());
}
```

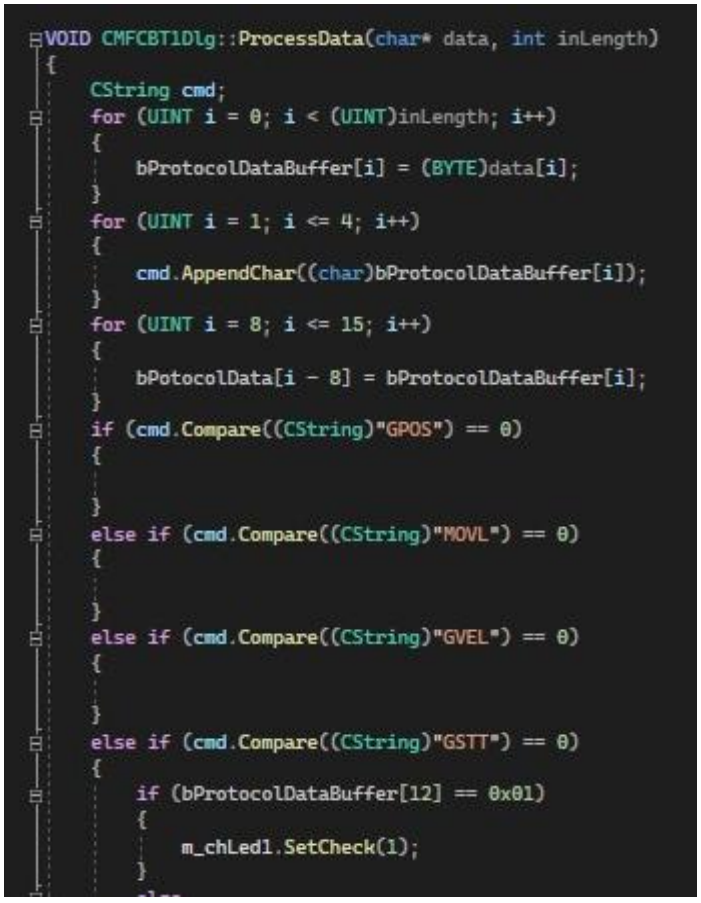
Chuong trình thực hiện gửi cho nút button

```
void CMFCBT1Dlg::OnBnClickedButtonPos()
{
    // TODO: Add your control notification handler code here
    BYTE ProtocolFrame[50] = {};
    UINT index = 0;
    if (!GetPortActivateValue()) return;
    memcpy(ProtocolFrame + index, bSTX, sizeof(bSTX));
    index += sizeof(bSTX);
    memcpy(ProtocolFrame + index, bGPOS, sizeof(bGPOS));
    index += sizeof(bGPOS);
    memcpy(ProtocolFrame + index, bOPT, sizeof(bOPT));
    index += sizeof(bOPT);
    memcpy(ProtocolFrame + index, bDATA, sizeof(bDATA));
    index += sizeof(bDATA);
    memcpy(ProtocolFrame + index, bSYNC, sizeof(bSYNC));
    index += sizeof(bSYNC);
    memcpy(ProtocolFrame + index, bETX, sizeof(bETX));
    index += sizeof(bETX);

    Write((char*)ProtocolFrame, index);

    CString cmd;
    cmd.Format(_T("%s"), (CString)"POS CMD: ");
    m_lsbReadSerial.InsertString(0, cmd);
    cmd.Empty();
    for (UINT i = 0; i < index; i++)
    {
        cmd.AppendFormat((CString)"%02X ", ProtocolFrame[i]);
    }
    m_lsbReadSerial.InsertString(0, cmd);
}
```

Chương trình điều khiển position cho button POS



```
VOID CMFCBT1Dlg::ProcessData(char* data, int inLength)
{
    CString cmd;
    for (UINT i = 0; i < (UINT)inLength; i++)
    {
        bProtocolDataBuffer[i] = (BYTE)data[i];
    }
    for (UINT i = 1; i <= 4; i++)
    {
        cmd.AppendChar((char)bProtocolDataBuffer[i]);
    }
    for (UINT i = 8; i <= 15; i++)
    {
        bPotocolData[i - 8] = bProtocolDataBuffer[i];
    }
    if (cmd.Compare((CString)"GPOS") == 0)
    {

    }
    else if (cmd.Compare((CString)"MOVL") == 0)
    {

    }
    else if (cmd.Compare((CString)"GVEL") == 0)
    {

    }
    else if (cmd.Compare((CString)"GSTT") == 0)
    {
        if (bProtocolDataBuffer[12] == 0x01)
        {
            m_chLed1.SetCheck(1);
        }
```

```
        if (bProtocolDataBuffer[12] == 0x01)
        {
            m_chLed1.SetCheck(1);
        }
        else
        {
            m_chLed1.SetCheck(0);
        }
        if (bProtocolDataBuffer[13] == 0x01)
        {
            m_chGPIO1.SetCheck(1);
        }
        else
        {
            m_chGPIO1.SetCheck(0);
        }
        if (bProtocolDataBuffer[14] == 0x01)
        {
            m_chLed2.SetCheck(1);
        }
        else
        {
            m_chLed2.SetCheck(0);
        }
        if (bProtocolDataBuffer[15] == 0x01)
        {
            m_chGPIO2.SetCheck(1);
        }
        else
        {
            m_chGPIO2.SetCheck(0);
        }
    }
}
```

Chương trình xử lý dữ liệu