ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO

Bài tập lớn 4: Thiết kế hệ thống nhúng

ĐỌC LOADCELL DÙNG HX711 HIỂN THỊ TRÊN MÁY TÌNH VÀ CABLID LOADCELL

Giáo viên hướng dẫn: Nguyễn Vĩnh Hảo

Nhóm 13

Họ và tên thành viên	MSSV
Nguyễn Thanh Trung Kiên	1751050
Huỳnh Văn Thành	2014491
Nguyễn Quang Thịnh	2014600
Trần Trung Tín	2014757

Thành phố Hồ Chí Minh, ngày… tháng 12 năm 2023

MỤC LỤC

I. Chuẩn bị thiết bị và linh kiện

1.1 Cảm biến Loadcell 1kg

Loadcell 1Kg là loại cảm biến khối lượng cân nặng chuyên dùng đo chính xác khối lượng, thường ứng dụng trong làm cân điện tử. Cảm biến thường được dùng chung với module chuyển đổi ADC 24bit HX711 giao tiếp với các vi điều khiển như: PIC, AVR, Arduino…

Thông số kĩ thuật:

Ứng dụng: Cân điện tử

Model: YZC-131

Tải trọng: 1Kg

Điện áp hoạt động: 5VDC

Kích thước: 12.7 x 12.7 x 80mm

Dây đỏ: Nguồn vào (+)

Dây đen: Nguồn vào (-)

Dây xanh: Ngõ ra(+)

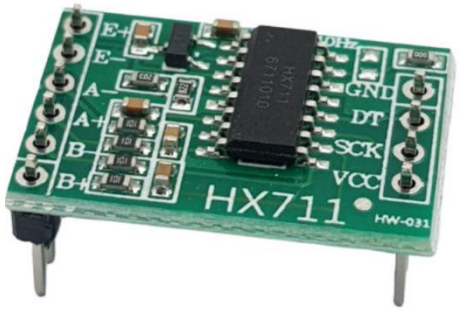
Dây trắng: Ngõ ra (-)

Sơ đồ chân của Loadcell:

Dây đỏ	Ngõ vào (+)
Dây đen	Ngõ vào (-)
Dây xanh Lá	Ngõ ra (+)
Dây trắng	Ngõ ra (-)

1.2 Mạch chuyển đổi ADC 24bit Loadcell HX711

Mạch chuyển đổi ADC 24bit Loadcell HX711 được sử dụng để đọc giá trị điện trở thay đổi từ cảm biến Loadcell (thường rất nhỏ không thể đọc trực tiếp bằng VĐK) với độ phân giải ADC 24bit và chuyển sang giao tiếp 2 dây (Clock và Data) để gửi dữ liệu về Vi điều khiển, thích hợp để sử dụng với Loadcell trong các ứng dụng đo cân nặng.



Thông số kỹ thuật:

Điện áp hoạt động : 2.7~5VDC

Dòng tiêu thụ : < 1.5 mA

Tốc độ lấy mẫu : 10 - 80 SPS (tùy chỉnh)

Độ phân giải : 24 bit ADC

Độ phân giải điện áp : 40mV

Kích thước : 38 x 21 x 10 mm

1.3 USB UART TTL CP2102

Thông số kỹ thuật:

Sử dụng điện áp 5VDC cấp trực tiếp từ cổng USB.

Các chân tín hiệu ngõ ra:

5V: Chân cấp nguồn 5VDC từ cổng USB, tối đa 500mA.

VCC: chân chọn mức tín hiệu TTL cho TX/RX, nếu nối với chân 5V sẽ là 5VDC, nối với 3V3 sẽ là 3.3VDC.

3V3: Chân nguồn 3.3VDC (dòng cấp rất nhỏ tối đa 100mA), không sử dụng để cấp nguồn, thường chỉ sử dụng để thiết đặt mức tín hiệu Logic.

TXD: chân truyền dữ liệu UART TTL, dùng kết nối đến chân nhận RX của các module sử dụng mức tín hiệu TTL.

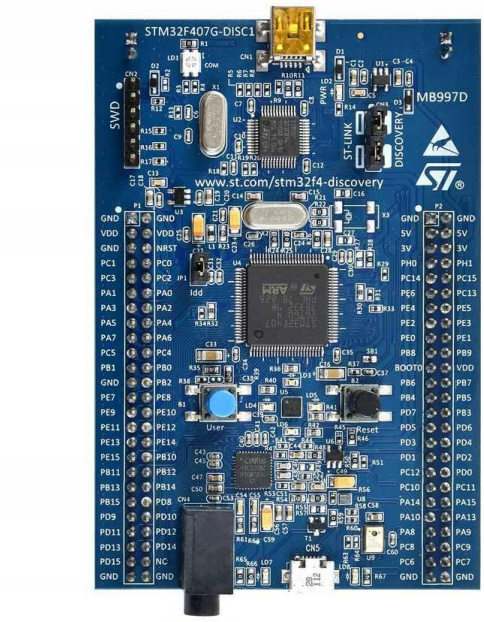
RXD: chân nhận dữ liệu UART TTL, dùng kết nối đến chân nhận TX của các module sử dụng mức tín hiệu TTL.

GND: chân cấp nguồn 0VDC.

Tích hợp led tín hiệu TX, RX.

Tương thích với hầu hết các hệ điều hành hiện nay: Windows, Mac, Linux.

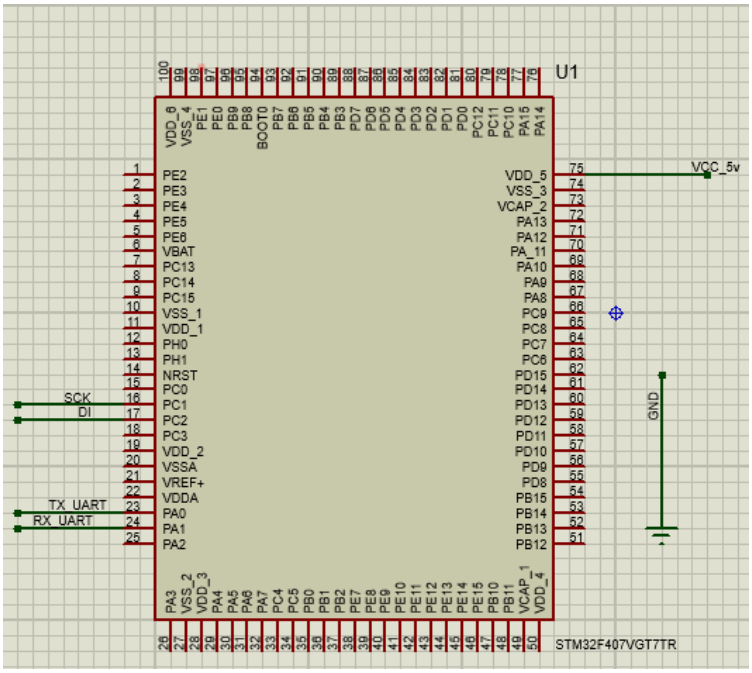
1.4 Kit STM32F407

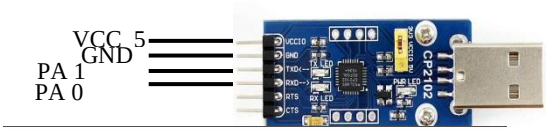


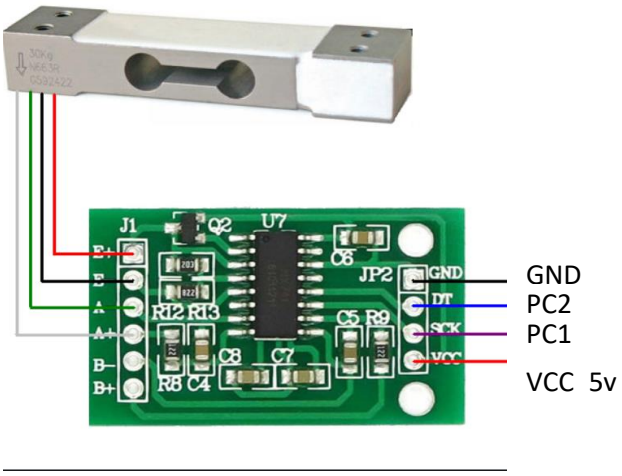
Kit STM32F407 Discovery hiện là loại kit được rất nhiều trường đại học hiện nay sử dụng trong giảng dạy vi điều khiển ARM, nếu so sánh về ngoại vi và sức mạnh của STM32 so với các dòng ARM của các hãng khác thì ở cùng 1 tầm giá, ARM của ST vượt trội về cấu hình và ngoại vi hơn rất nhiều.

Vi điều khiển chính: STM32F407VGT6. Tích hợp sẵn mạch nạp và Debug ST-LINK/V2.

II. Sơ đồ kết nối chân và nguyên lý hoạt động



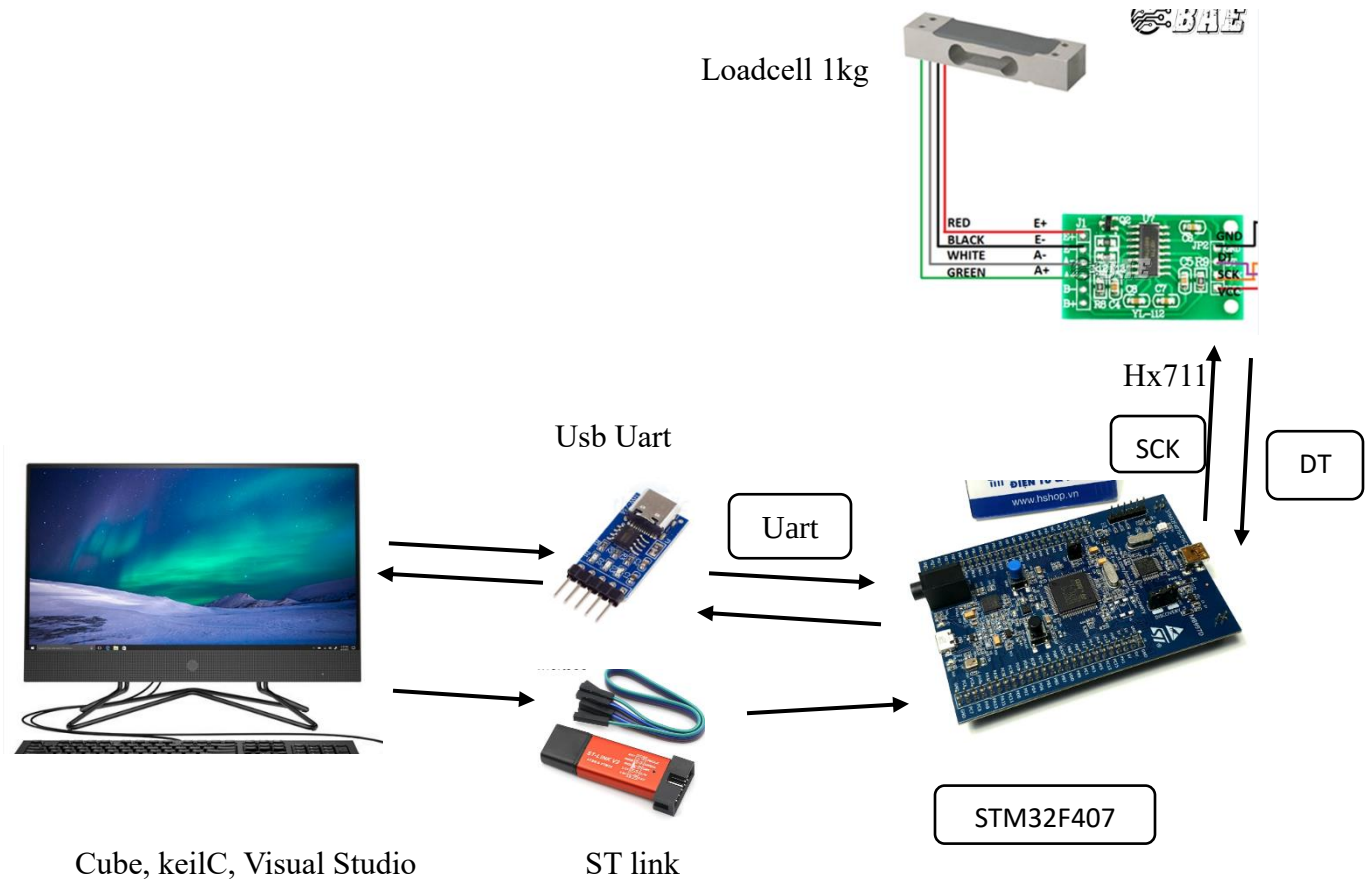




Nhóm sử dụng chân PA0 và PA1 để giao tiếp UART với giao diện, chân PC1 và PC2 để giao tiếp với module HX711.

III. Nguyên lý hoạt động

3.1. Sơ đồ kết nối



3.2. Cách đọc hx711:

Gửi tín hiệu xung đồng hồ (SCK): Gửi một chuỗi xung đồng hồ để kích hoạt HX711 và chuẩn bị cho việc đọc dữ liệu.

Đọc bit dữ liệu (DT): Đọc các bit dữ liệu từ HX711 bằng cách đọc trạng thái của chân DT. Thông thường, HX711 sẽ truyền 24 bit dữ liệu, bắt đầu từ bit cao nhất đến bit thấp nhất. Dữ liệu này có thể được đọc và chuyển đổi thành giá trị số thích hợp.

Hàm đọc loadcell

```
int32_t hx711_value(hx711_t *hx711)
{
  uint32_t data = 0;
  uint32_t  startTime = HAL_GetTick();
```

```
while(HAL_GPIO_ReadPin(hx711->dat_gpio, hx711->dat_pin) == GPIO_PIN_SET)
{
  hx711_delay(1);
  if(HAL_GetTick() - startTime > 150)
    return 0;
}
for(int8_t i=0; i<24 ; i++)
{
  HAL_GPIO_WritePin(hx711->clk_gpio, hx711->clk_pin, GPIO_PIN_SET);
  hx711_delay_us();
  HAL_GPIO_WritePin(hx711->clk_gpio, hx711->clk_pin, GPIO_PIN_RESET);
  hx711_delay_us();
  data = data << 1;
  if(HAL_GPIO_ReadPin(hx711->dat_gpio, hx711->dat_pin) == GPIO_PIN_SET)
    data ++;
}
data = data ^ 0x800000;
HAL_GPIO_WritePin(hx711->clk_gpio, hx711->clk_pin, GPIO_PIN_SET);
hx711_delay_us();
HAL_GPIO_WritePin(hx711->clk_gpio, hx711->clk_pin, GPIO_PIN_RESET);
hx711_delay_us();
return data;
```

Đầu tiên kiểm tra trạng thái của chân dữ liệu (dat_pin) của HX711. Nếu chân này đang ở trạng thái GPIO_PIN_SET (cao), nghĩa là HX711 đang bận thực hiện một đọc giá trị trước đó, vòng lặp sẽ tiếp tục chờ cho đến khi chân dat_pin chuyển sang trạng thái GPIO_PIN_RESET (thấp).

Nếu quá thời gian chờ vượt quá 150ms mà chân dat_pin vẫn không thay đổi trạng thái, hàm sẽ trả về giá trị 0 để báo hiệu lỗi.

Tiếp theo là vòng lặp này đọc 24 bit dữ liệu từ HX711 theo thứ tự từ bit cao nhất đến bit thấp nhất.

Mỗi lần lặp, chân clk_pin của HX711 sẽ được kích hoạt lên (GPIO_PIN_SET) để đọc bit dữ liệu.

Sau đó, chân clk_pin được đặt lại (GPIO_PIN_RESET) để chuẩn bị cho việc đọc bit tiếp theo.

Giá trị bit đọc được được dịch trái một lần (data = data << 1), sau đó kiểm tra trạng thái chân dat_pin. Nếu dat_pin là GPIO_PIN_SET, tức là bit đọc được là 1, ta tăng giá trị data lên 1.

Dữ liệu được đọc từ HX711 có dạng 24 bit, và bit cao nhất (bit 23) là bit dấu. Để chuyển đổi dữ liệu thành số nguyên có dấu, ta thực hiện phép XOR (data = data ^ 0x800000) để đảo bit dấu.

Cuối cùng, chân clk_pin của HX711 được kích hoạt lên một lần nữa để hoàn thành chu kỳ đọc giá trị từ HX711.

Hàm trả về giá trị đọc được (data).

Hàm tính toán chuyển đổi giá trị đọc.

```
float hx711_weight(hx711_t *hx711, uint16_t sample)
{
  hx711_lock(hx711);
  int64_t  ave = 0;
  for(uint16_t i=0 ; i<sample ; i++)
  {
    ave += hx711_value(hx711);
    hx711_delay(5);
  }
  int32_t data = (int32_t)(ave / sample);
  float answer =  (data - hx711->offset) / hx711->coef;
  hx711_unlock(hx711);
```

```
  return answer;
```

Vòng lặp này tính tổng giá trị đọc từ HX711 trong sample lần đọc.

Mỗi lần lặp, giá trị đọc từ HX711 được thêm vào biến ave.

Sau mỗi lần đọc, hàm hx711_delay() được gọi để tạo độ trễ 5ms trước khi tiếp tục đọc giá trị tiếp theo.

Sau khi hoàn thành vòng lặp, giá trị trung bình được tính bằng cách chia tổng (ave) cho số lần đọc (sample).

Giá trị trung bình (data) sau đó được sử dụng để tính toán giá trị trọng lượng chuẩn hóa.

Giá trị trọng lượng chuẩn hóa (answer) được tính bằng cách trừ giá trị hiệu chuẩn (offset) của HX711 từ giá trị đọc (data) và chia cho hệ số hiệu chuẩn (coef) của HX711.

Hàm này để tính giá trị offset khi không có tải bằng cách tính trung bình sample lần đọc.

```
int32_t hx711_value_ave(hx711_t *hx711, uint16_t sample)
{
  hx711_lock(hx711);
  int64_t  ave = 0;
  for(uint16_t i=0 ; i<sample ; i++)
  {
    ave += hx711_value(hx711);
    hx711_delay(5);
  }
  int32_t answer = (int32_t)(ave / sample);
  hx711_unlock(hx711);
  return answer;
}
```

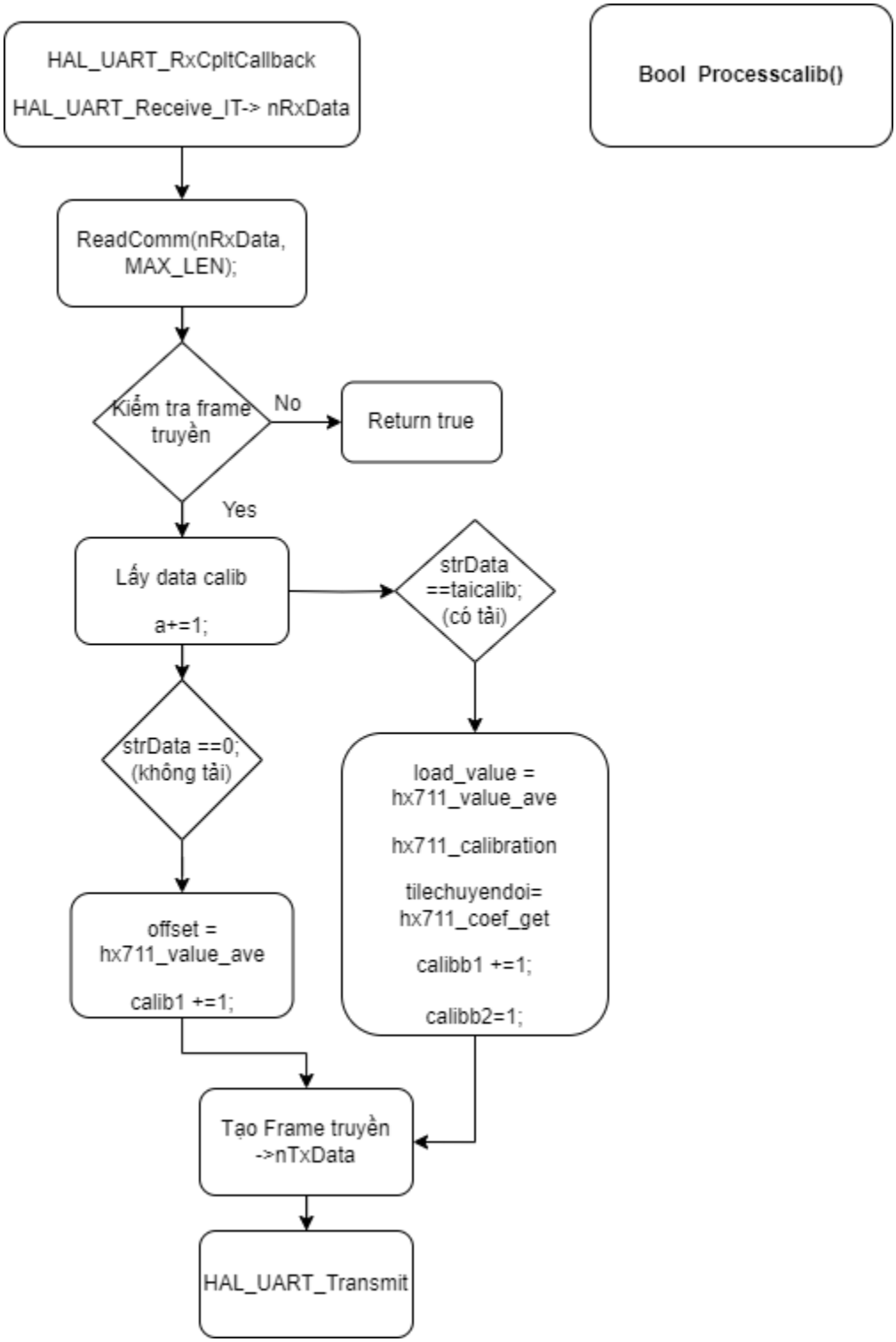
Hàm này tính giá trị chuyển đổi tỉ lệ.

```
void hx711_calibration(hx711_t *hx711, int32_t noload_raw, int32_t load_raw, float scale)
{
```

```
  hx711_lock(hx711);
  hx711->offset = noload_raw;
  hx711->coef = (load_raw - noload_raw) / scale;
  hx711_unlock(hx711);
}
float hx711_coef_get(hx711_t *hx711)
{
  return hx711->coef;
}
```

3.2 Sơ đồ thuật toán:

Trên STM32F407:



Main():

hx711_init()

while(1)

{

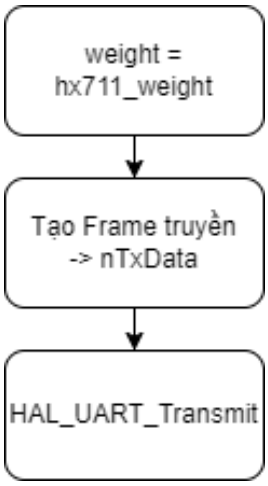
if (calibb1<2) Processcalib()

if(calibb1==2)

if(calibb2==0) Processcalib()

Calibb1=1;

else



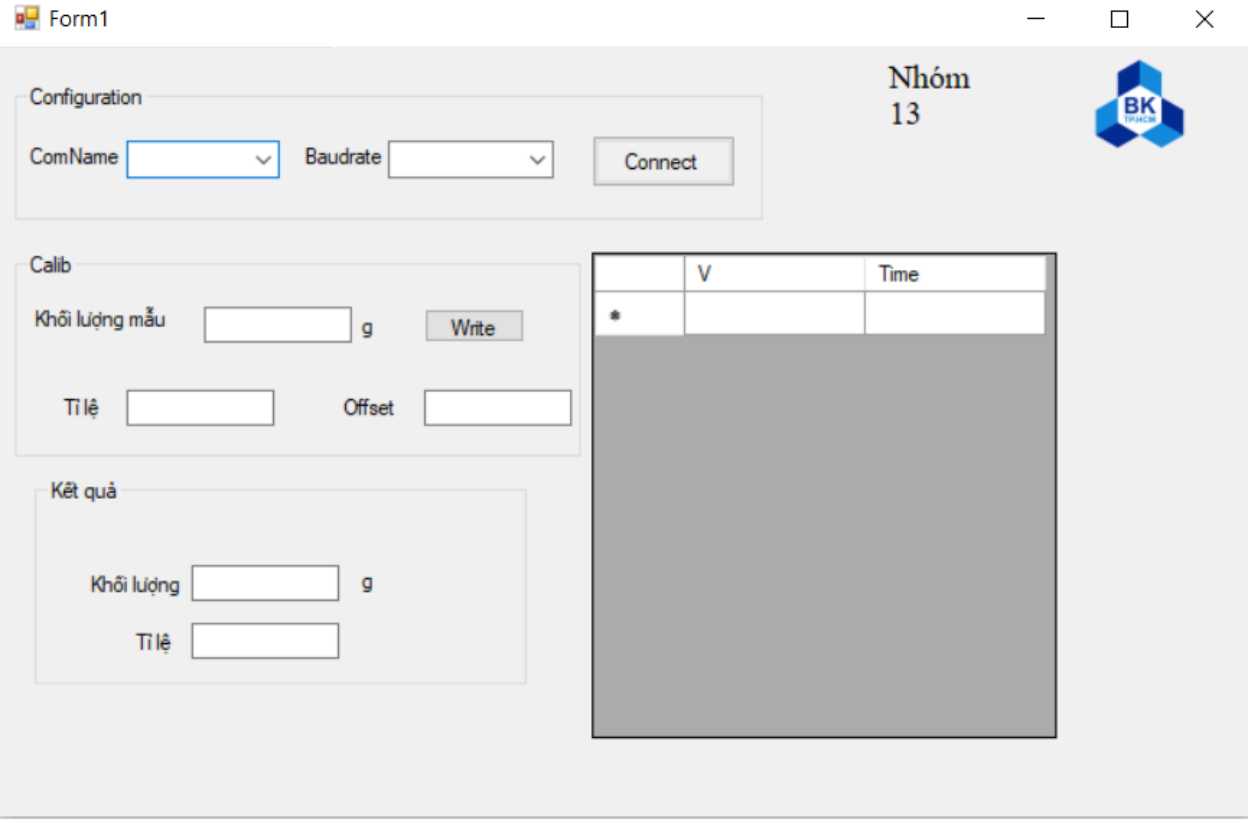
If(a>2) Calibb1=0;

a=1;

}

IV. Lập trình giao diện C# Visual Studio

4.1. Hình ảnh giao diện



4.2. Cấu trúc frame truyền nhận

Cấu trúc frame truyền từ GUI xuống STM32:

Byte	0	1	2	3	4	5..6	7
Thông tin	0xFE	0x00	0x04	0x00	0x01	Mẫu calib	0x40

Cấu trúc frame truyền từ STM32 lên GUI:

Byte	0	1	2	3..4	5..6	7
Thông tin	0xFF	0x00	0x04	Tỉ lệ	Khối lượng	0x40