

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

□ □ □



BÁO CÁO BÀI TẬP
CƠ SỞ ĐO LƯỜNG VÀ ĐIỀU KHIỂN SỐ

THIẾT KẾ THIẾT BỊ

HIỂN THỊ DẠNG SÓNG (OSCILLOSCOPE) ĐƠN KÊNH

<i>Họ và tên sinh viên</i>	<i>: MSV</i>
Nguyễn Trọng Nam	22022161
Doãn Đức Minh	22022135
Lê Thế Minh	22022215
Cao Song Toàn	22022158

Hà Nội, tháng 3 năm 2025

Nội dung báo cáo

I. Tổng quan	3
1.Nội dung thực hành.....	3
2.Linh kiện chuẩn bị.....	3
II. Nguyên lý hoạt động:	5
1.Ý nghĩa.....	5
2.Sơ lược về phần cứng và cách nối.....	5
3.Cách hoạt động thực tế.....	5
III. Thực hành	6
1. Mô phỏng trên labview.....	6
2. Code và nạp code cho vi điều khiển.....	6
IV. Kết quả thực nghiệm	8

I.Tổng quan

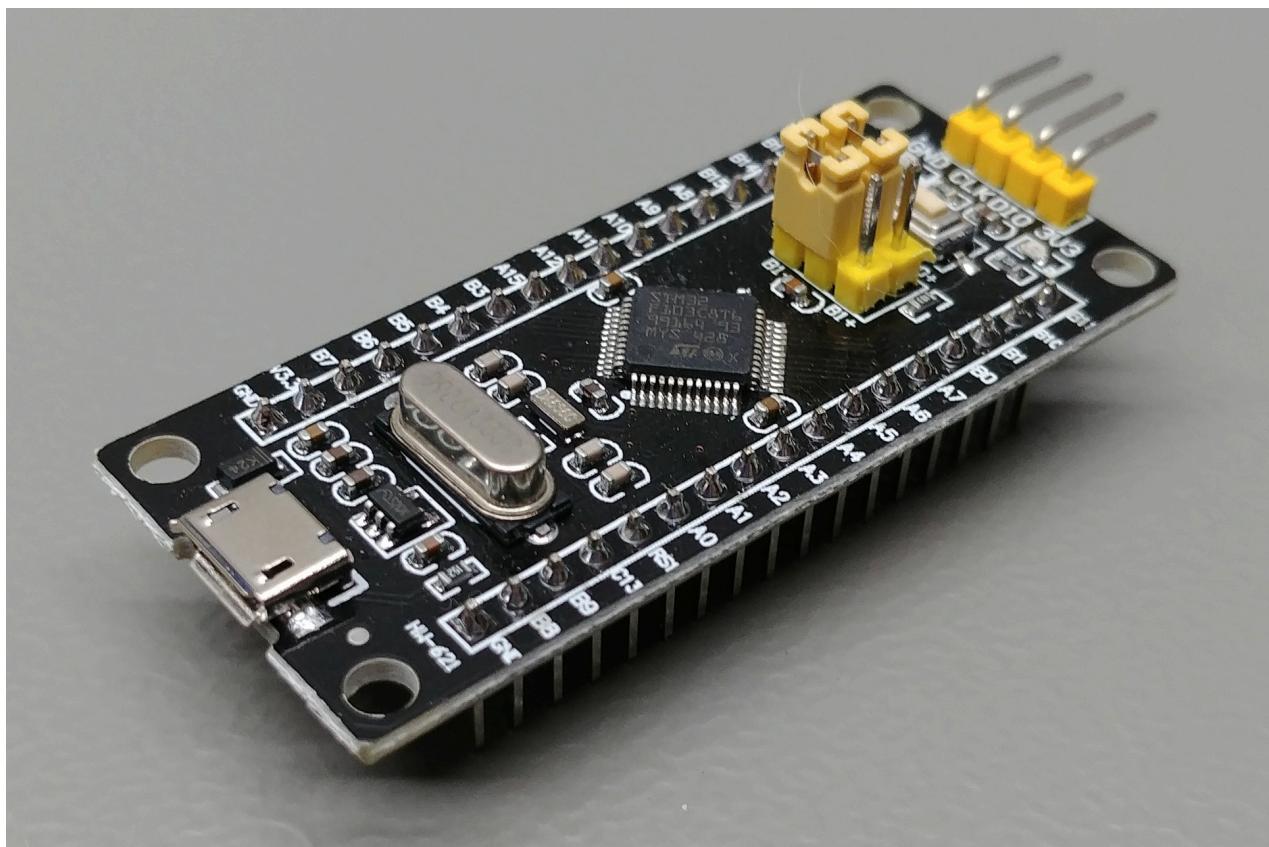
1.Nội dung thực hành

Lắp mạch điều khiển chuyển đổi tín hiệu tương tự - số (ADC), nhận tín hiệu từ nguồn ngoài rồi thu và mô phỏng dạng sóng của tín hiệu. Trong đó:

- Sử dụng vi điều khiển STM32 để mô phỏng tín hiệu vào bằng cách tạo xung PWM có dạng sóng vuông.
- Sử dụng mạch USB to TTL CP2102 để chuyển tín hiệu UART sang USB để máy tính nhận được.
- Lập trình vi điều khiển ở mức thanh ghi.
- Sử dụng LabView để thu tín hiệu, hiện dạng sóng tín hiệu, điều chỉnh Vol/Div, Time/Div và đồng bộ tín hiệu.

2.Linh kiện chuẩn bị

- Sử dụng vi điều khiển STM32f410C8U6.



- Sử dụng mạch USB to TTL CP2102.



- Một số dây nối, ST link.

II. Nguyên lý hoạt động:

1. Ý nghĩa:

- **Tạo PWM trên chân PA1** với tần số 1 Hz, duty cycle 50%.
- **Đọc giá trị điện áp (thực chất là mức PWM)** tại chân PA0 (ADC).
- **Gửi dữ liệu điện áp đo được** qua UART2 (chân PA2, PA3) với baudrate 115200.
- **Chuyển tín hiệu UART sang USB** bằng mạch CP2102 để máy tính nhận được dữ liệu.
- **LabVIEW** sẽ đọc và hiển thị các giá trị điện áp (hoặc sóng) từ cổng COM ảo do CP2102 cung cấp, đồng thời có thể hiển thị dạng sóng hoặc giá trị số.

Trong mạch thực tế, **nối PA0 với PA1** để “tự đọc” lại xung PWM vừa tạo ra. Ngoài ra, **nối PA2, PA3 với mạch CP2102** để truyền dữ liệu ADC qua giao tiếp UART.

2. Sơ lược về phần cứng và cách nối:

- STM32F410C6U8

- **PA0**: chân ADC (chế độ analog).
 - **PA1**: chân PWM output (TIM2_CH2).
 - **PA2**: TX của UART2.
 - **PA3**: RX của UART2 (ở đây chủ yếu dùng TX để gửi dữ liệu, RX không nhất thiết phải dùng).
 - **3V3, GND**: cung cấp nguồn cho CP2102 (tùy module CP2102 mà có thể cần 3V3 hoặc 5V, nhưng thường CP2102 có thể hoạt động ở 3.3V logic).
- **Mạch CP2102 (USB to TTL)**
- **TXD** (của CP2102) \leftrightarrow **PA3** (RX của STM32) – để truyền dữ liệu từ máy tính xuống STM32.
 - **RXD** (của CP2102) \leftrightarrow **PA2** (TX của STM32) – để STM32 gửi dữ liệu ADC lên máy tính.
 - **GND** chung với GND của STM32.
 - **3V3** lấy từ STM32 (nếu cần cấp nguồn cho CP2102, tùy module).
 - **Nối PA0 với PA1**: để chân ADC đọc chính tín hiệu PWM. Khi PWM thay đổi mức điện áp (trong thực tế là dao động giữa 0V và 3.3V), ADC sẽ lấy mẫu điện áp đó.
- **Trên LabVIEW:**
- Chọn cổng COM tương ứng với CP2102 (ví dụ COM3, COM4,...).
 - Đọc chuỗi dữ liệu mà STM32 gửi ra (các giá trị điện áp).
 - Giải mã và hiển thị trên đồ thị sóng (hoặc hiển thị giá trị số) tùy người dùng lập trình trong LabVIEW
- ### **3. Cách hoạt động thực tế (luồng tín hiệu)**

- **Tín hiệu PWM:**

- Tạo ra trên **chân PA1** với tần số 1Hz, duty 50%.
- Dạng sóng vuông, 0V → 3.3V, cao 0.5 giây, thấp 0.5 giây.

- **ADC trên PA0:**

- **Nối PA0 với PA1**, nên ADC đọc được mức điện áp PWM (0V hoặc 3.3V, tùy theo thời điểm).
- Vì tần số chỉ 1Hz, nếu đặt delay ngắn thì vẫn có thể gặp giá trị ADC lúc cao hoặc lúc thấp (hoặc ngưỡng trung gian nếu biên PWM đang chuyển).

- **Gửi dữ liệu qua UART:**

- STM32 gửi giá trị điện áp (theo công thức tính 3.3V) ra dưới dạng ASCII (ví dụ “3.30\n” hoặc “0.00\n”).
- Thông qua **PA2 → RXD của CP2102**.
- Máy tính nhận qua cổng COM ảo.

- **LabVIEW:**

- Mở cổng COM tương ứng.
- Đọc chuỗi (ví dụ “3.30” hoặc “0.00”).
- Chuyển về giá trị số, hiển thị dạng bảng số hoặc vẽ đồ thị dạng “dải xung” (nếu muốn mô phỏng thành sóng vuông, có thể căn theo thời gian đọc).

- **Hiển thị kết quả:**

- Vì chu kỳ PWM là 1 giây, người dùng sẽ thấy điện áp đo được một nửa thời gian ở mức 3.30V (tương đương 50% chu kỳ) và 0.00V (50% chu kỳ).

Nếu tốc độ đọc/hiển thị đủ nhanh điện áp sẽ “nhảy” giữa 0 V và 3.3 V một cách tuần hoàn 1 giây/lần.

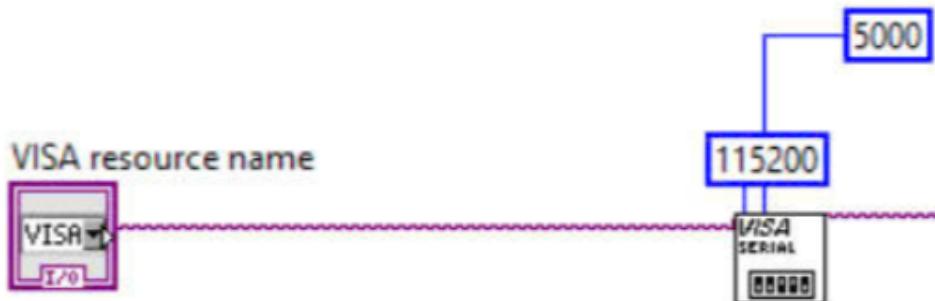
III.Thực hành

1. Mô phỏng trên labview

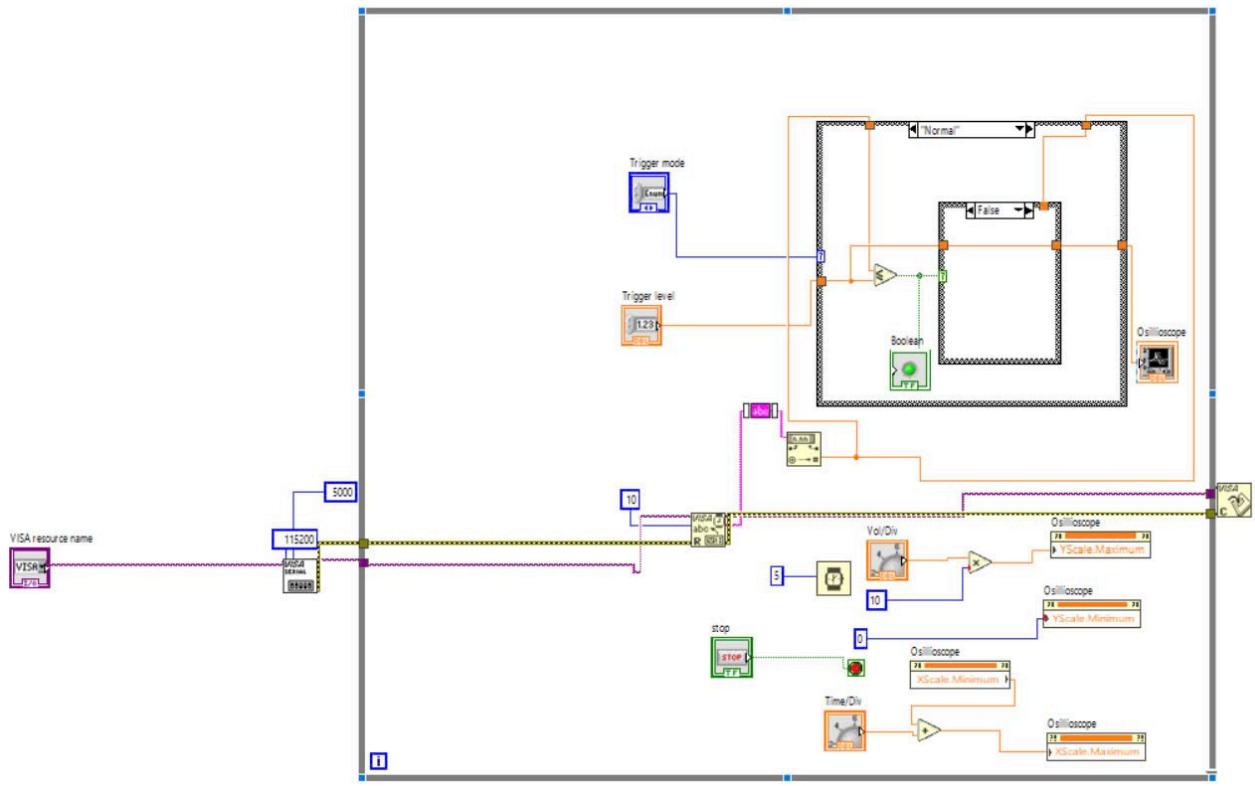
- Phát sóng/thu dữ liệu:

- Ví điều khiển (STM32) được lập trình để đọc giá trị (giá trị ADC) hoặc tạo sóng, sau đó gửi liên tục dữ liệu qua UART với baud rate 115200.
- Module USB-to-Serial (CP2102) chuyển tín hiệu UART sang USB để máy tính nhận được.

- Cấu hình cổng VISA (VISA resource name):

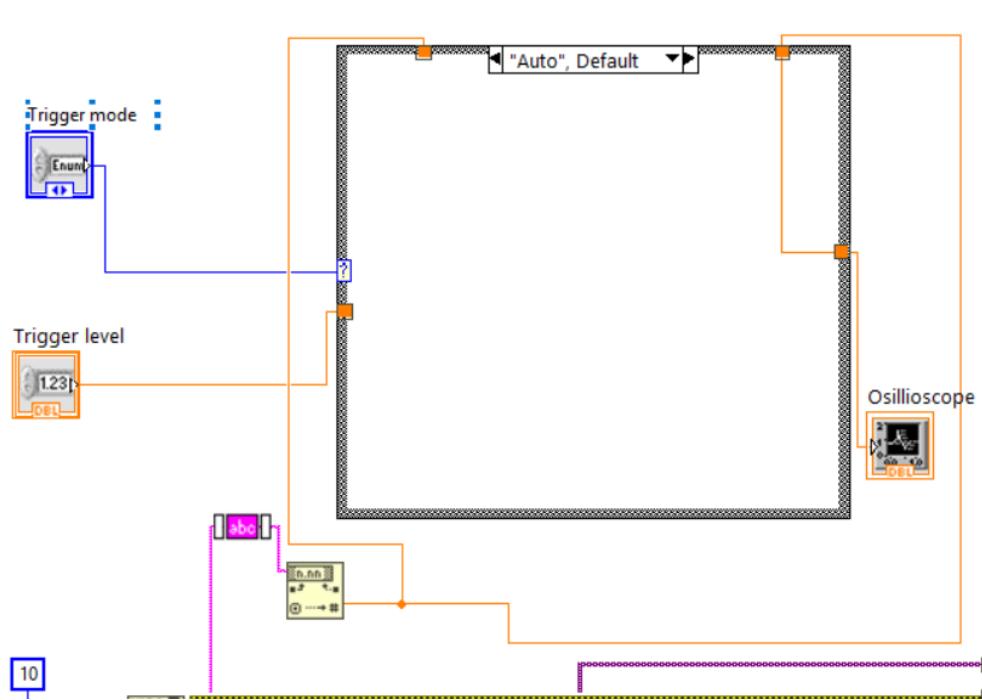


- Trong LabVIEW, block **VISA Configure Serial Port** (hoặc các thiết lập tương tự) được sử dụng để chọn cổng COM tương ứng với module USB-to-Serial, và đặt tốc độ baud 115200.
- Khi chạy chương trình, LabVIEW sẽ mở cổng COM và sẵn sàng nhận dữ liệu.

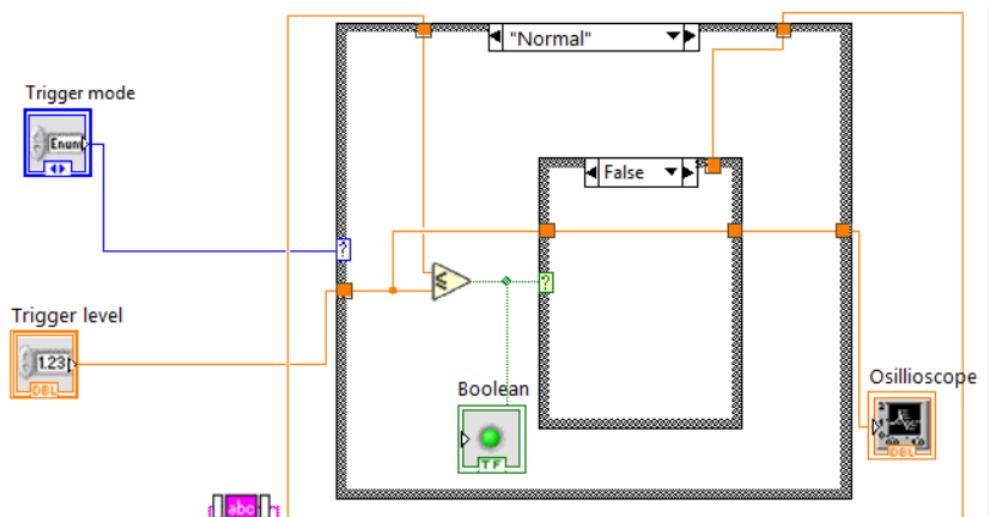


-Vòng lặp đọc dữ liệu (While Loop):

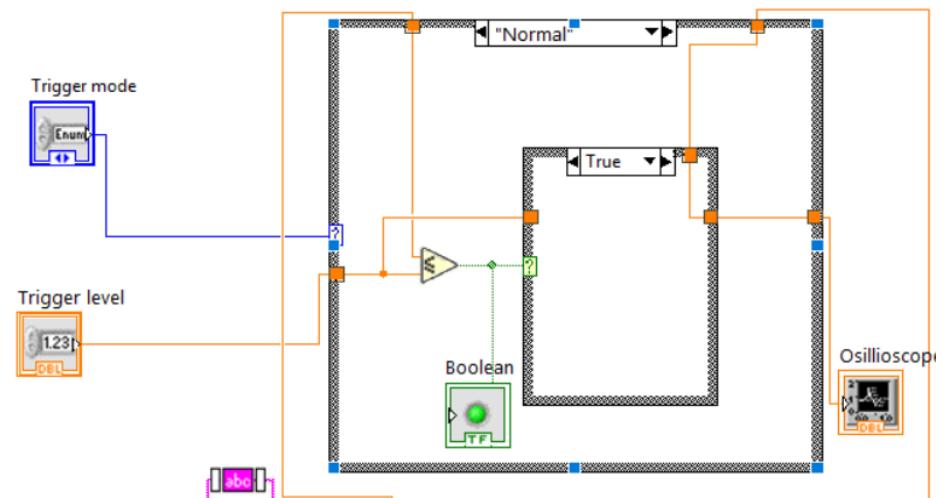
- Bên trong vòng lặp, LabVIEW dùng **VISA Read** để lấy một lượng byte (chuỗi) từ cổng nối tiếp.
- Khỏi “Scan from String” nhận dữ liệu chuỗi (màu hồng) và dùng định dạng “n.nn” để quét, sau đó xuất ra giá trị số (màu cam). Đây là cách LabVIEW chuyển dữ liệu kiểu text (ví dụ đọc được từ UART hoặc file) thành số để xử lý tiếp.
- Mỗi lần đọc, ta sẽ nhận được một (hoặc nhiều) giá trị ADC (dạng ASCII) do vi điều khiển gửi lên, thường kết thúc bằng ký tự xuống dòng \n.
- **Đồng bộ tín hiệu:**
- Khỏi case structure có 2 trạng thái:



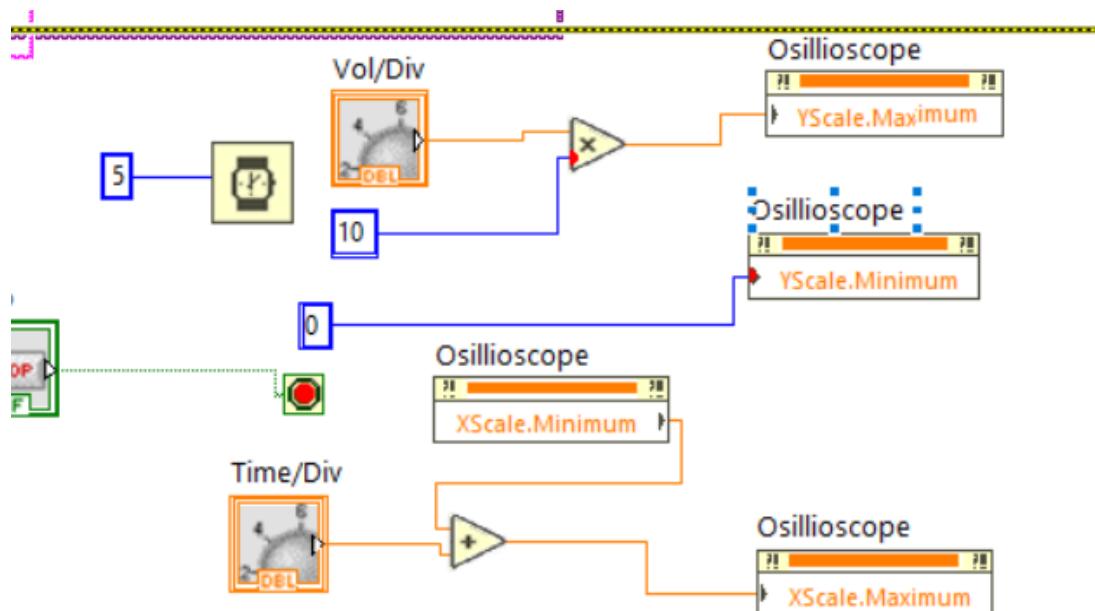
- Auto: Dữ liệu dạng số sau khi chuyển đổi sẽ được đưa trực tiếp vào waveform charchart.
- Normal: Dữ liệu dạng số sẽ được so sánh với giá trị trigger level.



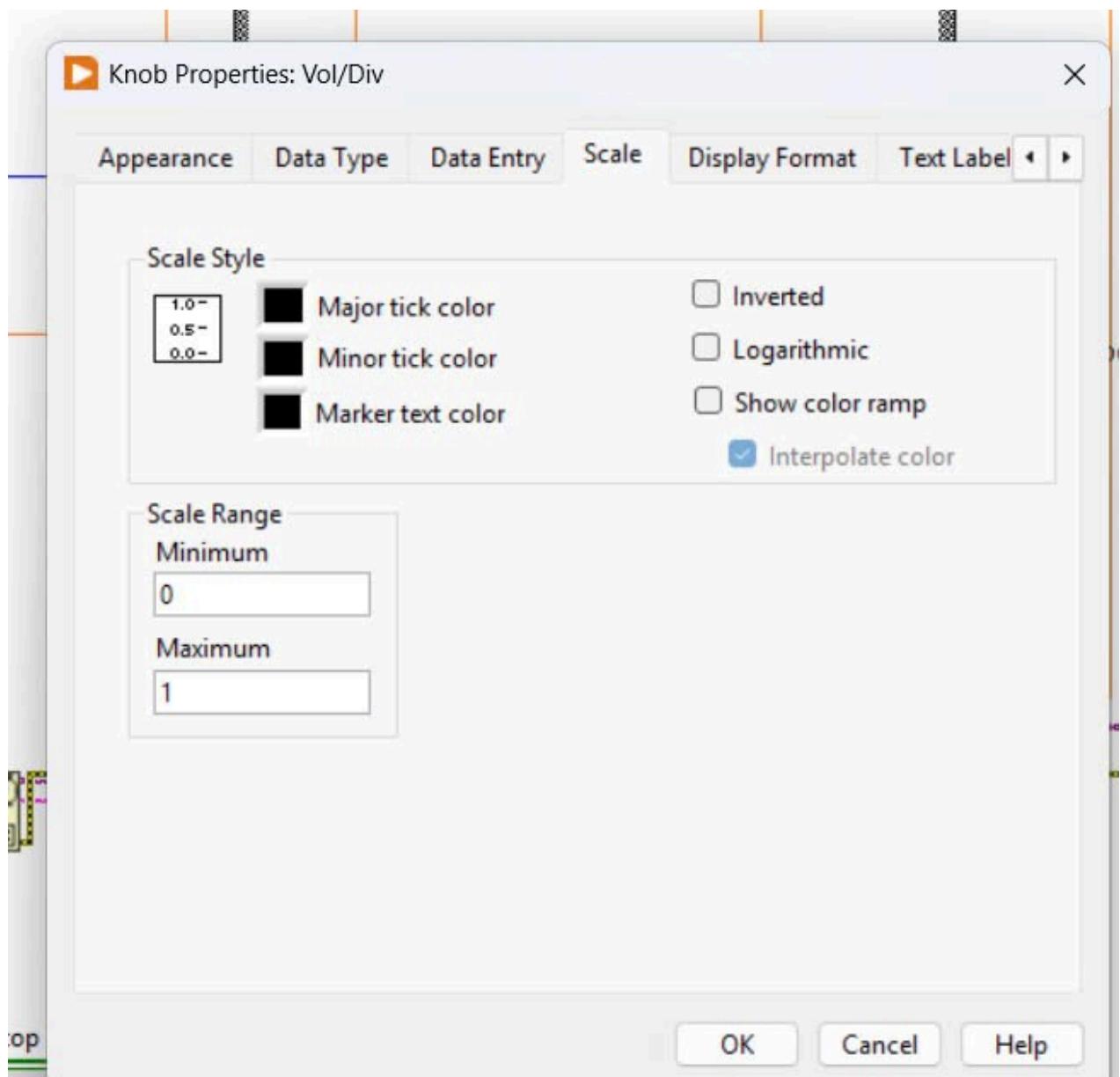
- nếu độ lớn tín hiệu > trigger level: độ lớn tín hiệu sẽ = giá trị trigger level



- nếu độ lớn tín hiệu \leq trigger level: độ lớn tín hiệu giữ nguyên

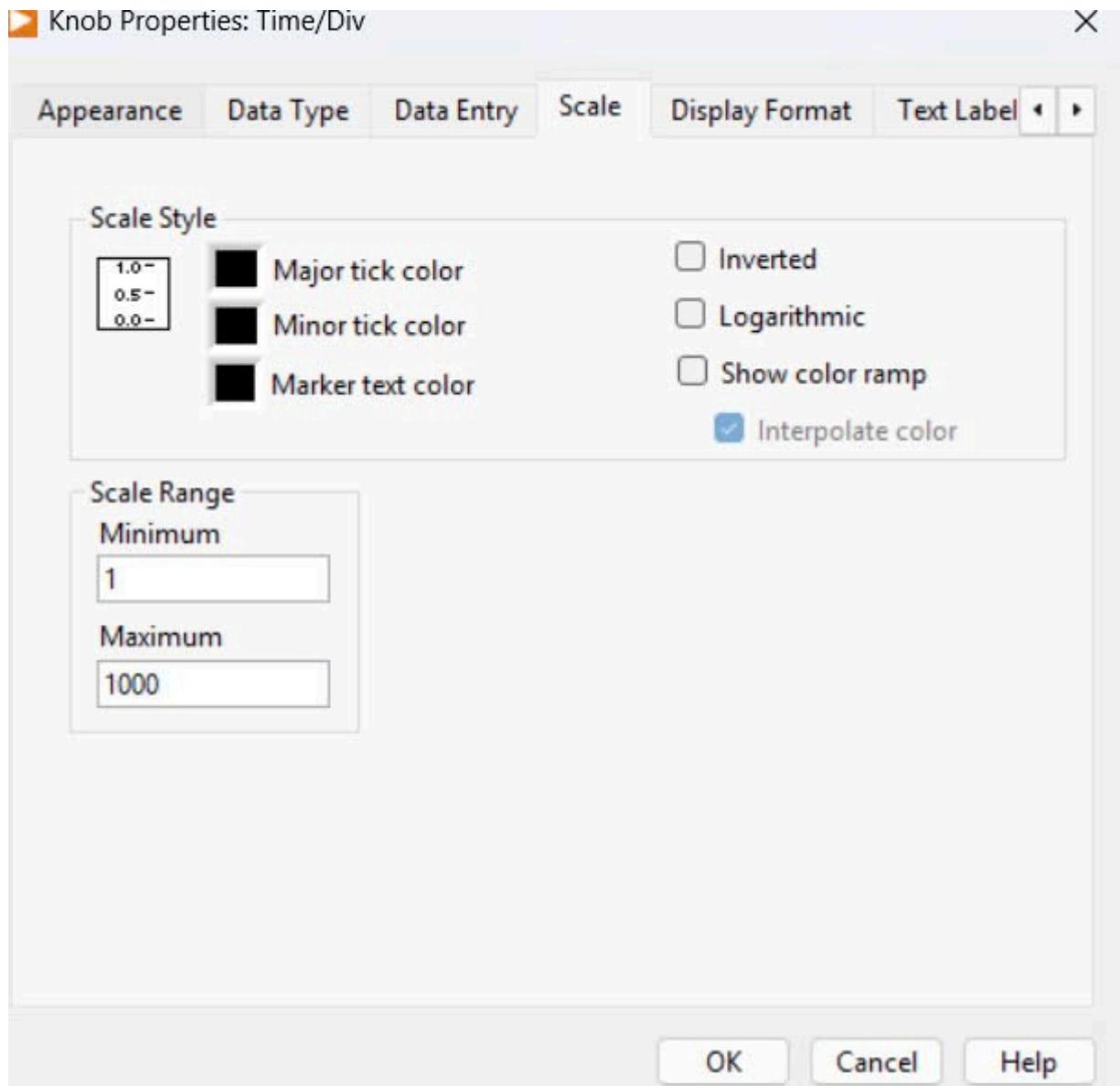


- Vol/Div:



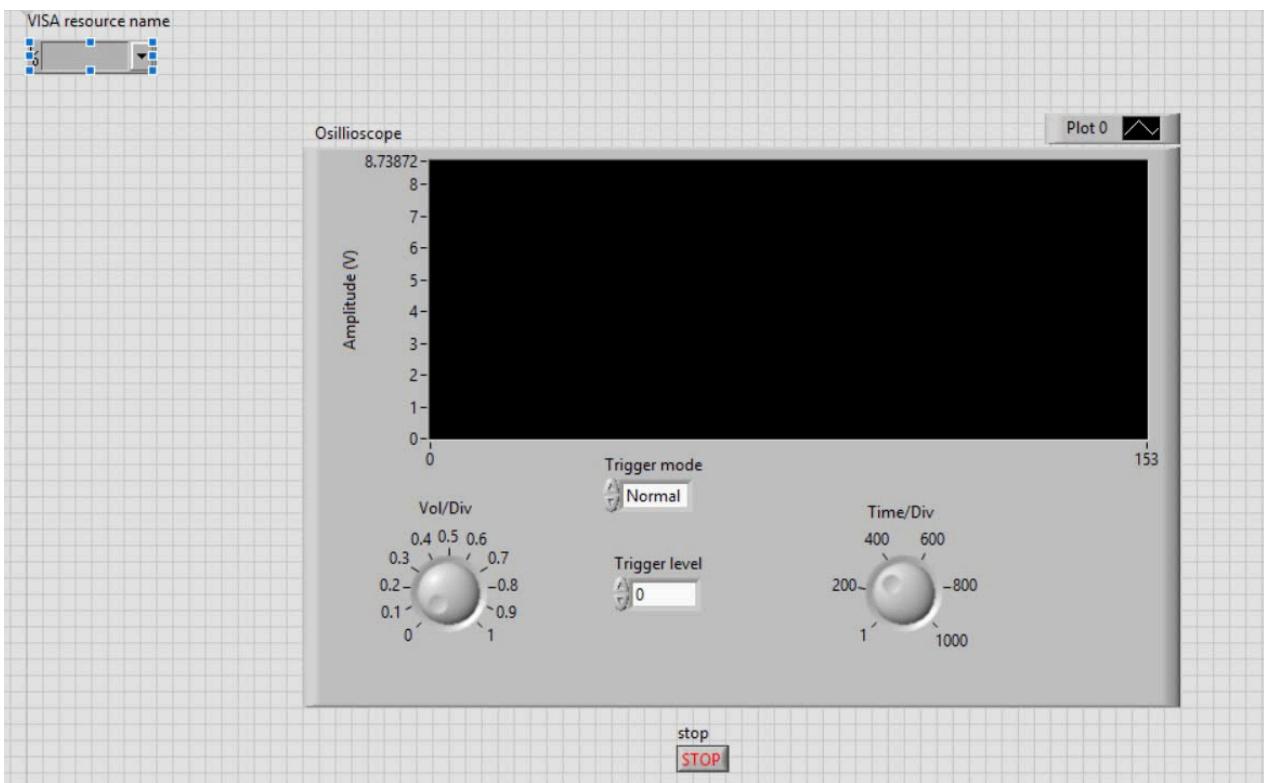
Có định giá trị Y min = 0. Sử dụng khói knob với khoảng từ 0 đến 1 rồi nhân với 10 để chỉnh giá trị Y max từ 0 đến 10. Bằng cách này sẽ scale trực Y như mong muốn.

- **Time/Div:**



Vì tín hiệu được lấy mẫu liên tục nên ta sẽ ko cố định giá trị X min mà ta sẽ thay đổi khoảng thời gian hiển thị trên màn hình để scale bằng cách sử dụng 1 khói knob có giá trị từ 1 đến 1000 rồi cộng với giá trị X min hiện tại để tính giá trị X max, hay $X_{max} = X_{min} + a$ (với a từ 1 đến 1000). Bằng cách này sẽ scale trực X như mong muốn.

- Hiển thị trên “Oscilloscope”:



- Dữ liệu số sau khi chuyển đổi được đưa vào các khối xử lý hoặc khối hiển thị dạng sóng trong LabVIEW (**Waveform Chart**).
- Các nút điều chỉnh như **Volt/Div**, **Time/Div** thực chất là **thay đổi hệ số scale** trên trực tung (điện áp) và trực hoành (thời gian) để người dùng có thể quan sát tín hiệu phù hợp.
- Giá trị “Scale Minimum” và “Scale Maximum” là những tham số để xác định biên độ hiển thị (**Volt/Div**) hoặc tốc độ hiển thị (**Time/Div**).
- Trigger mode dùng để chuyển trạng thái sang **auto** hoặc **normal** như đã nói ở trên và khối Trigger level dùng để ghi giá trị level mong muốn. 2 khối này được dùng cho **đồng bộ tín hiệu**

- Điều khiển tốc độ lấy mẫu / hiển thị:

- Có một block **Wait (ms)**, cấu hình timeout cho VISA Read để điều khiển tốc độ đọc dữ liệu (mỗi 5ms).
- Tránh việc chương trình LabVIEW bị quá tải khi dữ liệu đến quá nhanh, đồng thời tạo nhịp hiển thị mượt mà hơn.

- Dừng chương trình:

- Người dùng có thể nhấn nút “Stop” (hoặc một nút tương tự trong giao diện LabVIEW), lúc này vòng lặp sẽ thoát và block **VISA Close** được gọi để giải phóng cổng COM.

2. Code và nạp code cho vi điều khiển

- Cấu hình GPIO (GPIO_Init):

```

3@ void GPIO_Init(void) {
4    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
5    GPIOA->MODER |= (3 << 0); // PA0: Analog mode (ADC input)
6    GPIOA->MODER |= (2 << 2); // PA1: Alternate function mode (PWM output
7    GPIOA->AFR[0] |= (1 << 4); // AF1: TIM2_CH2 cho PA1
8}

```

- + Bật clock cho cổng GPIOA.
- + Cấu hình chân PA0 ở chế độ analog để sử dụng cho ADC (đo tín hiệu analog).
- + Cấu hình chân PA1 ở chế độ alternate function để dùng cho PWM (đầu ra xung PWM), và gán alternate function AF1 cho chân PA1 để kết nối với kênh 2 của TIM2.

- Cấu hình ADC (ADC_Init & ADC_Read):

```

10@ void ADC_Init(void) {
11    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
12    ADC1->SQR3 = 0; // Kênh 0 (PA0)
13    ADC1->CR2 |= ADC_CR2_ADON;
14 }
15
16@ uint16_t ADC_Read(void) {
17    ADC1->CR2 |= ADC_CR2_SWSTART;
18    while (!(ADC1->SR & ADC_SR_EOC));
19    return ADC1->DR;
20 }

```

- + Bật clock cho ADC1.
- + Trong ADC_Init, thiết lập ADC1 để đọc từ kênh 0 (tương ứng với PA0) và bật ADC bằng cách thiết lập bit ADON.
- + Hàm ADC_Read khởi động quá trình chuyển đổi (bằng cách đặt bit SWSTART), chờ đến khi quá trình chuyển đổi hoàn thành (kiểm tra cờ EOC) và trả về giá trị số đo được từ thanh ghi dữ liệu (DR).

- Cấu hình UART (UART2_Init, UART2_SendChar, UART2_SendString):

```
22@void UART2_Init(void) {
23    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
24    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
25    GPIOA->MODER |= (2 << 4) | (2 << 6); // PA2, PA3: AF mode
26    GPIOA->AFR[0] |= (7 << 8) | (7 << 12); // AF7: USART2
27    USART2->BRR = 0x8B; // 115200 baud (HCLK = 16MHz)
28    USART2->CR1 = USART_CR1_TE | USART_CR1_UE;
29 }
30
31@void UART2_SendChar(uint8_t c) {
32    while (!(USART2->SR & USART_SR_TXE));
33    USART2->DR = c;
34 }
35
36@void UART2_SendString(char *str) {
37    while (*str) UART2_SendChar(*str++);
38 }
--
```

- + Bật clock cho USART2 và cổng GPIOA.
- + Cấu hình chân PA2 và PA3 sang chế độ alternate function (AF7) để sử dụng cho giao tiếp UART.
- + Thiết lập baud rate 115200 cho USART2 (với HCLK = 16MHz).
- + Hàm UART2_SendChar gửi một ký tự qua USART bằng cách đợi cho đến khi thanh ghi truyền (TXE) sẵn sàng, sau đó ghi ký tự vào thanh ghi dữ liệu.
- + Hàm UART2_SendString gửi chuỗi ký tự bằng cách lặp qua từng ký tự của chuỗi và gửi đi.

- Chuyển đổi số sang chuỗi (int_to_string & adc_to_string):

```

40 void int_to_string(uint16_t num, char *str) {
41     int i = 0;
42     if (num == 0) {
43         str[i++] = '0';
44     } else {
45         while (num > 0) {
46             str[i++] = (num % 10) + '0';
47             num /= 10;
48         }
49     }
50     int j = 0;
51     char temp;
52     for (j = 0; j < i / 2; j++) {
53         temp = str[j];
54         str[j] = str[i - 1 - j];
55         str[i - 1 - j] = temp;
56     }
57     str[i] = '\0';
58 }

```

- + Hàm **int_to_string** chuyển một số nguyên (uint16_t) sang chuỗi ký tự. Quá trình này thực hiện bằng cách lấy từng chữ số (từ phần dư của phép chia) và lưu vào mảng sau đó đảo chuỗi để được thứ tự chính xác.

```

60 void adc_to_string(uint16_t adc_value, char *str) {
61     uint32_t voltage_x100 = (adc_value * 330) / 4095;
62     uint16_t integer_part = voltage_x100 / 100;
63     uint16_t decimal_part = voltage_x100 % 100;
64
65     char int_str[5];
66     int_to_string(integer_part, int_str);
67
68     int i = 0;
69     while (int_str[i] != '\0') {
70         str[i] = int_str[i];
71         i++;
72     }
73     str[i++] = '.';
74     if (decimal_part < 10) {
75         str[i++] = '0';
76         str[i++] = decimal_part + '0';
77     } else {
78         str[i++] = (decimal_part / 10) + '0';
79         str[i++] = (decimal_part % 10) + '0';
80     }
81     str[i++] = '\n';
82     str[i] = '\0';
83 }

```

- + Hàm **adc_to_string** nhận giá trị ADC, tính điện áp theo đơn vị 0.01V (giả sử điện áp tham chiếu là 3.3V, chuyển đổi: voltage_x100 = (adc_value * 330) / 4095), sau đó tách phần nguyên và phần thập phân, ghép chúng lại thành chuỗi (ví dụ: "3.30") và thêm ký tự xuống dòng.
- **Cấu hình PWM (PWM_Init):**

```

85@ void PWM_Init(void) {
86    // Bật clock cho TIM2
87    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
88
89    // Cấu hình TIM2: tần số 1Hz, HCLK = 16MHz
90    TIM2->ARR = 63999; // Auto-reload: 16MHz / (250 * 64000) = 1Hz
91    TIM2->PSC = 249;   // Prescaler = 249 (250 sau +1)
92
93    // Cấu hình kênh 2 (PA1) cho PWM mode 1
94    TIM2->CCMR1 |= (6 << 12); // PWM mode 1 (110)
95    TIM2->CCER |= TIM_CCER_CC2E; // Bật kênh 2 output
96
97    // Đặt duty cycle (50%: 32000/64000)
98    TIM2->CCR2 = 32000;
99
100   // Bật timer
101   TIM2->CR1 |= TIM_CR1_CEN;
102 }

```

- + Bật clock cho TIM2.
- + Thiết lập bộ đếm của TIM2 sao cho tần số PWM là 1Hz (dựa trên HCLK 16MHz, sử dụng prescaler và auto-reload register).
- + Cấu hình kênh 2 của TIM2 (gắn với PA1) ở chế độ PWM mode 1.
- + Đặt duty cycle cho PWM là 50% (bằng cách gán giá trị 32000 vào CCR2 so với giá trị auto-reload 64000).
- + Cuối cùng, kích hoạt TIM2 để bắt đầu tạo tín hiệu PWM.
- **Hàm main:**

```

104@ int main(void) {
105    GPIO_Init();
106    ADC_Init();
107    UART2_Init();
108    PWM_Init();
109
110    char buffer[10];
111    while (1) {
112        uint16_t adc_value = ADC_Read(); // Đo tín hiệu tại PA0
113        adc_to_string(adc_value, buffer);
114        UART2_SendString(buffer);
115        for (volatile int i = 0; i < 10000; i++); // Giữ delay để lấy mẫu
116    }
117 }

```

- + **Gọi các hàm khởi tạo: cấu hình GPIO, ADC, UART và PWM.**
- + **Trong vòng lặp vô hạn, liên tục thực hiện:**
 - Đo tín hiệu analog qua ADC từ PA0.
 - Chuyển đổi giá trị ADC sang dạng chuỗi thể hiện điện áp.

- Gửi chuỗi điện áp qua UART.
- Thực hiện delay ngắn để tạo khoảng thời gian giữa các lần lấy mẫu.

IV.Kết quả thực nghiệm

Video thực nghiệm: Video.mp4

Source code: <https://github.com/iGhost1107/BTVN1.git>

1. Dải tần hoạt động của thiết bị là bao nhiêu?

Với code gốc (có delay, gửi chuỗi ASCII qua UART 115200), dải tần hoạt động để “bắt” dạng sóng liên tục chỉ ở **mức vài trăm Hz**.

Nếu tinh chỉnh (loại bỏ delay, tăng baud, tối ưu xử lý), có thể nâng lên **vài kHz đến hàng chục kHz** hoặc cao hơn.

2. Dải tần đó phụ thuộc vào yếu tố nào?

Tốc độ lấy mẫu (sampling rate) của ADC: Phải đủ lớn (tối thiểu gấp đôi tần số tín hiệu theo định lý Nyquist).

Tốc độ xử lý của vi điều khiển: Ảnh hưởng đến việc đọc ADC, xử lý, và truyền dữ liệu.

Băng thông truyền dữ liệu (UART/USB/SPI...): Nếu truyền dữ liệu ra máy tính quá chậm, sẽ không theo kịp tốc độ lấy mẫu cao.

Độ phân giải và kiến trúc ADC: Một số STM32 cho phép lấy mẫu hàng trăm kHz đến vài MHz, nhưng cũng bị giới hạn bởi điều kiện thực tế (nhiều, thời gian chuyển đổi).

Mạch tiền khuếch đại hoặc suy giảm (nếu có): Để đảm bảo tín hiệu vào luôn nằm trong dải điện áp cho phép của ADC, đồng thời không méo hoặc bị cắt.

Cách tạo sóng (nếu là máy phát): Dùng DAC nội (nếu có), DAC ngoài, hay PWM + lọc RC; mỗi phương pháp có tốc độ và chất lượng khác nhau.