

BÁO CÁO THỰC HÀNH BÀI 4

Môn học: CHUYÊN ĐỀ THIẾT KẾ HỆ THỐNG NHÚNG 1- Mã lớp: CE437.N11

Giảng viên hướng dẫn thực hành: Phạm Minh Quân

Thông tin các sinh viên	Mã số sinh viên	Họ và tên sinh viên
	19520887	Phạm Trung Quốc
	19521651	Phạm Trọng Huỳnh
	19520928	Viên Minh Tân
	19520036	Phạm Quốc Đăng
Link các tài liệu tham khảo (nếu có)		
Đánh giá của giảng viên: + Nhận xét + Các lỗi trong chương trình + Gợi ý		

MỤC LỤC

1. Nội dung thực hành	3
1.1. Phần 1	3
1.1.1. Sơ đồ giải thuật.....	3
1.1.2. Mã nguồn	4
1.1.3. Kết quả.....	7
1.2. Phần 2	8
1.2.1. Lưu đồ thuật toán	9
1.2.2. Giải quyết bài toán.....	9
1.2.3. Mã nguồn	10
1.2.4. Kết quả.....	14

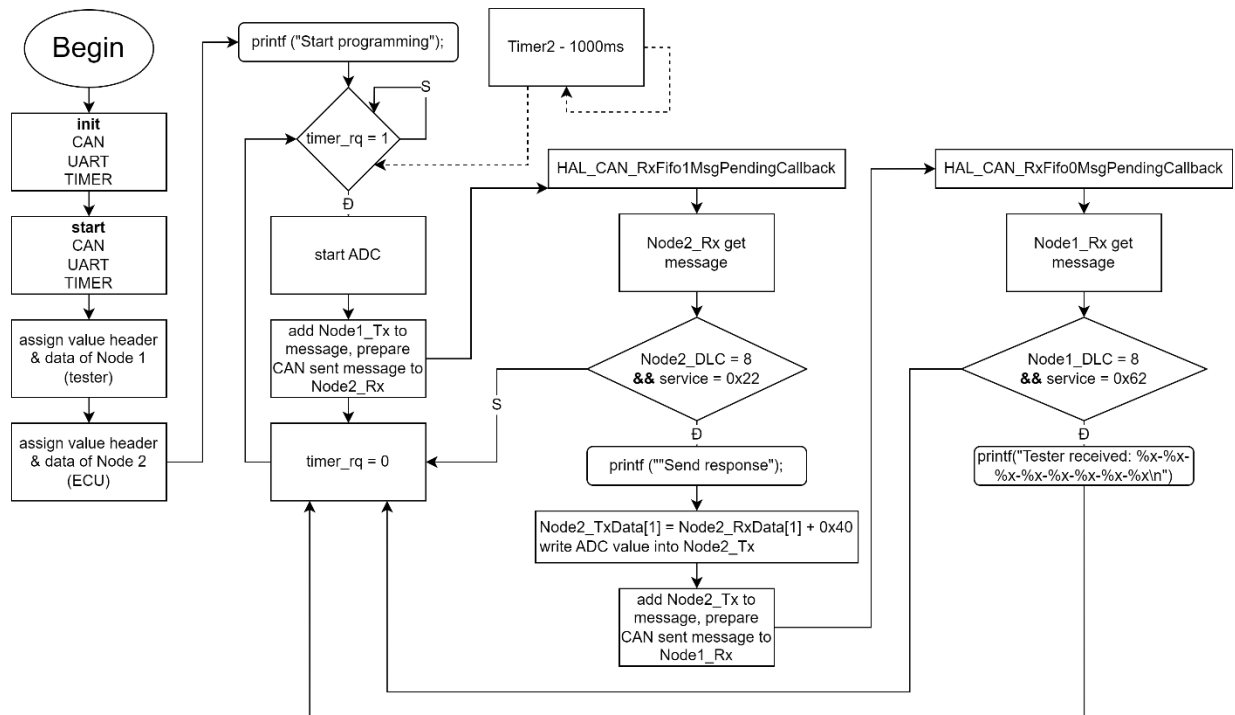
1. Nội dung thực hành

1.1. Phần 1

Trong phần này, sinh viên sẽ hiện thực dịch vụ \$22H – Read Data by Identifier để đọc và hiển thị qua UART giá trị ADC:

- Tester sẽ gửi yêu cầu tới ECU để đọc giá trị ADC.
- ECU sẽ đọc giá trị ADC từ biến trở và gửi lại cho Tester.
- Tester nhận được giá trị ADC từ ECU và hiển thị giá trị qua UART.
- Mỗi chu kỳ 1 giây, Tester sẽ thực hiện gửi lại yêu cầu đọc ADC 1 lần.

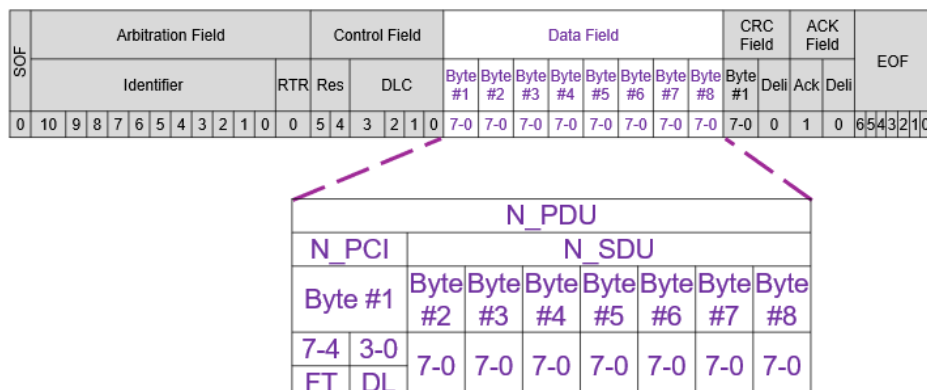
1.1.1. Sơ đồ giải thuật



Hình 1 Sơ đồ giải thuật

Với bài thực hành này, sau mỗi 1000ms Tester gửi đến ECU một yêu cầu để đọc giá trị ADC, Ở đây nhóm sử dụng Timer 2 được thiết lập thời gian tràn là 1000ms. Sau khi Timer 2 tràn biến timer_rq được bật lên 1 để Tester tiếp tục gửi một request khác đến ECU.

Ở bài toán này, vì giá trị ADC là 12 bits (1.5 bytes) nên nhóm sẽ sử dụng Single frame để giải quyết bài toán (vì single frame chứa được 7 bytes dữ liệu), Single frame được cấu hình như bên dưới.



Hình 2 Định nghĩa Single Frame

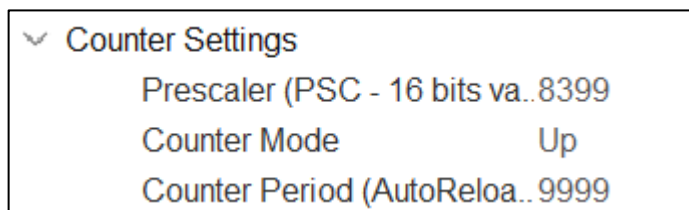
Tuy nhiên bài toán này chúng ta sử dụng thực hiện dịch vụ \$22H (Read Data by Identifier) nên chúng ta chỉ còn 6 bytes chứa dữ liệu, tuy nhiên vẫn đủ để truyền giá trị ADC. Sau khi request được gửi đến ECU, ở ECU sẽ đọc giá trị ADC và gửi lại một gói tin có chứa dữ liệu ADC về Tester. Và người dùng đọc nó thông qua UART.

1.1.2. Mã nguồn

1.1.2.1. Giải thích timer

Cấu hình TIMER 2 với chu kỳ tràn là 1000ms. Sử dụng công thức sau để tính được chu kỳ tràn của 1 TIMER:

$$\frac{\text{Clock system}}{(\text{Prescaler}+1)(\text{Counter Period}+1)} = \frac{1}{\text{Over period}}$$



Hình 3 Cấu hình Timer 2

```

121 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
122 {
123     if(htim->Instance == htim2.Instance)
124     {
125         timer_rq = 1;
126     }
127 }

```

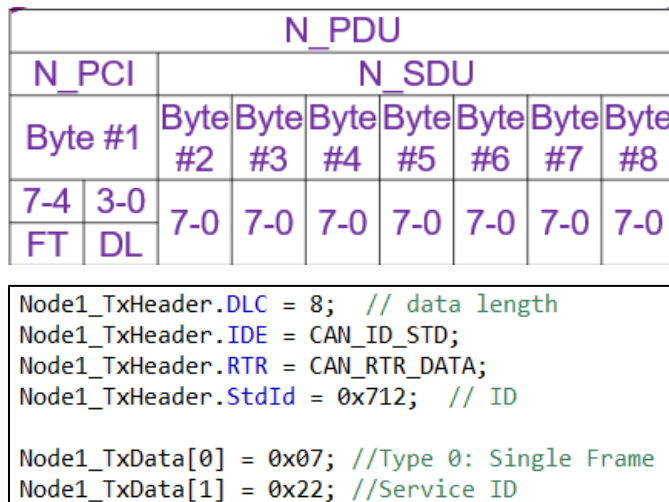
Hình 4 Code interrupt Timer 2

Mỗi khi timer 2 tràn, biến timer_rq được bật lên 1, phục vụ cho việc thông báo đến Tester biết rằng đã đến lúc gửi một request đến ECU để đọc tín hiệu ADC.

1.1.2.2. Cấu hình các gói tin và đọc ADC

a. Gói tin từ Tester request đến ECU để đọc ADC

Với Tester sẽ tương ứng sẽ là Node 1, chúng ta cần gán các giá trị của header cho Tester như sau trước khi gửi đi. Đồng thời ở phần data của Tester ở byte đầu tiên sẽ là 0x07 (bao gồm Frame Type và Data length) và byte thứ 2 là 0x22 là Service ID, xem hình bên dưới.



Hình 5 Cấu hình cho Single Frame cho request từ Tester

b. Gói tin từ ECU gửi lại Tester chứa dữ liệu ADC

Với ECU tương ứng sẽ là Node 2, chúng ta cần gán các giá trị của header cho ECU như sau trước khi gửi đi. Đồng thời ở phần data của ECU ở byte đầu tiên sẽ là 0x07 (bao gồm Frame Type và Data length), byte thứ 2 sẽ là SID + 0x040 (response), var1 là giá trị được đọc về từ ADC. Vì giá trị ADC 12 bit nên cần chứa giá trị này ở byte thứ 3 và 4 của ECU.

Positive Response with only Service ID (SID)

SID +
0x40

Opt. Data

```
Node2_TxHeader.DLC = 8; // data length
Node2_TxHeader.IDE = CAN_ID_STD;
Node2_TxHeader.RTR = CAN_RTR_DATA;
Node2_TxHeader.StdId = 0x7A2; // ID

Node2_TxData[0] = 0x07;
```

```
// Send response with ADC, Service id + 0x40
Node2_TxData[1] = Node2_RxData[1] + 0x40;
Node2_TxData[2] = (var1>>8) & 0xff; //high
Node2_TxData[3] = var1 & 0xff; //low
```

Hình 6 Cấu hình cho Single Frame cho response từ ECU

c. Đọc giá trị ADC

Với hàm callback này của interrupt, mỗi khi biến trở thay đổi sẽ được gán vào biến var1 để lưu trữ và sử dụng khi cần thiết, giá trị này sẽ được cập nhật liên tục.

```
543 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
544 {
545     if(hadc->Instance == hadc1.Instance)
546     {
547         var1 = HAL_ADC_GetValue(&hadc1);
548     }
549 }
```

Hình 7 Hàm callback cho việc đọc ADC

1.1.2.3. Luồng xử lý

Trong vòng lặp While sẽ luôn kiểm tra liệu rằng timer 2 có tràn chưa (đồng nghĩa liệu rằng Tester đã đến lúc gửi request cho ECU để đọc ADC chưa). Gọi hàm `HAL_CAN_AddTxMessage` để tiến hành giá trị Tx của Tester lên buffer để sẵn sàng cho việc nhận từ ECU, và trong phần code nhóm đã thiết kế để khi gọi hàm này thì sẽ hoàn tất việc gửi request và nhận luôn cả response từ ECU để đọc ADC. Sau cùng là trả về giá trị cờ cho timer, chuẩn bị cho lần request tiếp theo.

```
209 while (1)
210 {
211     if (timer_rq == 1)
212     {
213         HAL_ADC_Start_IT(&hadc1);
214
215         HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData,
216                             &Node1_TxMailbox);
217         timer_rq = 0; // flag for timer
218     }
219 }
```

Hình 8 Vòng lặp

Sau khi gọi `HAL_CAN_AddTxMessage` trong hàm `while()`, hàm `HAL_CAN_RxFifo1MsgPendingCallback` sẽ được kích hoạt. Bên trong hàm này, chúng ta gọi hàm `HAL_CAN_GetRxMessage` để lấy giá trị mà Tester đã gửi, và ECU nhận. Cần một hàm if để kiểm tra liệu rằng đã bắt đúng được gói có chứa dịch vụ mình cần. Nếu

đúng dịch vụ, cần thực hiện một số tác vụ sau để tiến hành trả về giá trị ADC mong muốn.

- Cộng thêm SID + 0x40 (response)
- Đọc giá trị ADC và chứa trong 2 byte của frame
- Cuối cùng là gọi hàm để response lại từ ECU về Tester.

```
105 void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
106 {
107     HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &Node2_RxHeader, Node2_RxData);
108     if (Node2_RxHeader.DLC == 8 && Node2_RxData[1] == 0x22)
109     {
110         HAL_GPIO_TogglePin(LED1_GPIO_Port, LED2_Pin);
111         printf("Send response\r\n");
112         // Send response with ADC, Service id + 0x40
113         Node2_TxData[1] = Node2_RxData[1] + 0x40;
114         Node2_TxData[2] = (var1 >> 8) & 0xff; //high
115         Node2_TxData[3] = var1 & 0xff; //low
116         HAL_CAN_AddTxMessage(&hcan2, &Node2_TxHeader, Node2_TxData,
117                             &Node2_TxMailbox);
118     }
119 }
```

Hình 9 Call back Fifo1

Đọc gói tin mà Tester nhận được thông qua UART.

```
95 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
96 {
97     HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &Node1_RxHeader, Node1_RxData);
98     if (Node1_RxHeader.DLC == 8 && Node1_RxData[1] == 0x62)
99     {
100         printf("Tester received: %x-%x-%x-%x-%x-%x-%x-%x\n", Node1_RxData[0],
101             HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
102     }
103 }
```

Hình 10 Call back Fifo0

1.1.3. Kết quả

Chương trình bắt đầu với dòng “Start programming ...” dùng để nhận biết chương trình đã bắt đầu thời thời điểm nào, thuận tiện cho việc debug các lỗi. Chúng ta thấy rằng mỗi khi Tester gửi một request đến ECU, dòng “Send response” sẽ được print ra terminal, nhận thấy mỗi khi “Send response” được in ra chúng cách nhau 1000ms (thỏa yêu cầu bài toán).

```

07:51.915 -> Start programming 1...
07:51.915 -> Send response
07:51.915 -> Tester received: 7-62-f-ff-0-0-0-0
07:52.910 -> Send response
07:52.910 -> Tester received: 7-62-f-ff-0-0-0-0
07:53.906 -> Send response
07:53.906 -> Tester received: 7-62-f-ff-0-0-0-0
07:54.900 -> Send response
07:54.900 -> Tester received: 7-62-f-ff-0-0-0-0
07:55.895 -> Send response
07:55.895 -> Tester received: 7-62-f-ff-0-0-0-0
07:56.889 -> Send response
07:56.889 -> Tester received: 7-62-f-ff-0-0-0-0
07:56.922 -> Tester received: 7-62-f-ff-0-0-0-0
07:57.917 -> Send response
07:57.917 -> Tester received: 7-62-f-ff-0-0-0-0
07:58.911 -> Send response
09:41.904 -> Send response
09:41.904 -> Tester received: 7-62-f-ff-0-0-0-0
09:42.899 -> Send response
09:42.899 -> Tester received: 7-62-f-ff-0-0-0-0
09:43.893 -> Send response
09:43.893 -> Tester received: 7-62-c-3d-0-0-0-0
09:44.921 -> Send response
09:44.921 -> Tester received: 7-62-9-23-0-0-0-0
09:45.916 -> Send response
09:45.916 -> Tester received: 7-62-7-35-0-0-0-0
09:46.911 -> Send response
09:46.911 -> Tester received: 7-62-7-36-0-0-0-0
09:47.905 -> Send response
09:47.905 -> Tester received: 7-62-7-36-0-0-0-0
09:48.899 -> Send response
09:48.899 -> Tester received: 7-62-7-37-0-0-0-0

```

Hình 11 Đọc giá trị gói tin qua UART

Ngay sau khi Tester gửi một request đến ECU, ECU đọc giá trị ADC từ biến trở và phản hồi về cho Tester một gói tin, đọc gói tin đó thông qua UART, ta kiểm tra dòng “Tester received: 7-62-f-ff-0-0-0-0”. Với byte thứ 3 và thứ 4 chứa giá trị ADC mà ECU đã đọc, khi tiến hành vặn nút xoay trên biến trở, giá trị này ngay lập tức thay đổi.

1.2. Phần 2

Trong phần này, sinh viên sẽ hiện thực dịch vụ \$2EH – Write Data by Identifier để ghi một lượng dữ liệu xuống ECU. ECU sẽ hồi đáp và hiển thị các giá trị nhận được lên UART.

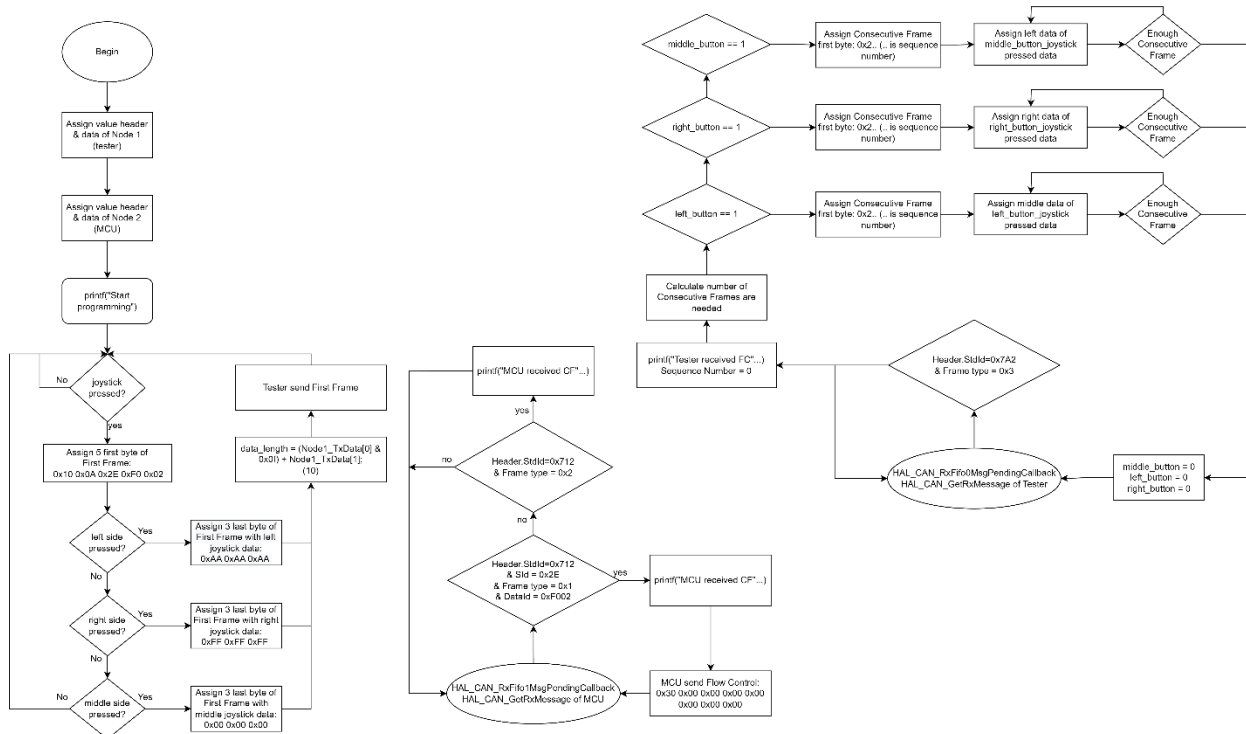
- Tester sẽ đọc vị trí của Joystick khi Joystick được nhấn.
- Sau đó Tester sẽ gửi yêu cầu tới ECU để ghi giá trị dữ liệu tương ứng với vị trí của Joystick. Lưu ý rằng, dữ liệu sẽ không đủ để truyền trong 1 khung gói tin riêng. Sinh viên phải hiện thực quá trình giao tiếp với phương pháp đa gói tin (CAN_TP).
- ECU sẽ nhận yêu cầu và hiển thị toàn bộ gói tin lên UART.
- Các quy định về gói tin: Data ID: 0xF002; DID Length: 10 bytes; Format: Binary

Middle: 00 00 00 00 00 00 (hex)

Left: AA AA AA AA AA AA (hex)

Right: FF FF FF FF FF FF (hex)

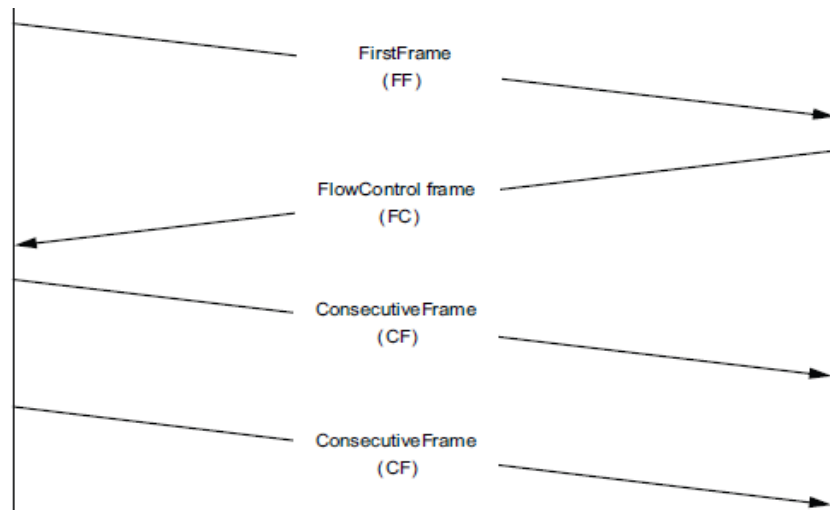
1.2.1. Lưu đồ thuật toán



Hình 12: Lưu đồ giải thuật bài 2

1.2.2. Giải quyết bài toán

Theo đề ta thấy cần gửi đi một gói tin chứa 10 byte dữ liệu, tuy nhiên, single frame mà chúng ta đã làm ở bài thứ 1 chỉ giải quyết được bài toán tối đa 7 bytes dữ liệu. Vì thế cần một giải pháp khác, đó chính là Multi Frame.



Hình 13 Multi frame

Với First frame (FF) chứa tổng chiều dài frame cũng như một số cấu hình khác, Consecutive Frame (CF) với chức năng chứa dữ liệu chính cho Multi frame. Flow control (FC) dùng để điều khiển các gói tin CF. Vậy cần phân tích xem với 10 bytes dữ liệu thì cần gửi bao nhiêu CF.

Với data mà Joystick có 6 byte dữ liệu, vì thế để thỏa như đề yêu cầu là 10 bytes chúng ta thêm 4 byte nữa vào đầu gói tin. Phân tích First frame, Flow control và Consecutive frame ta được các frame như sau

First frame								
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	DL							
1	0	0A	2E	F0	02	0	0	0

Consecutive frame									
Byte #1		Byte #2		Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	DL								
3	000			0	6E	F0	02	0	0

Flow control								
Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8
7-4	3-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0
FT	FS	BS	ST min	N/A	N/A	N/A	N/A	N/A
2	1	0	Data	Data	Data	Data	Data	Data

Qua việc phân tích Multi frame dành cho bài toán này, ta thấy chỉ cần 1 lần gửi CF thì đã hoàn tất một tác vụ cần thiết mà đề đã yêu cầu.

1.2.3. Mã nguồn

1.2.3.1. Giải thích phần interrupt cho Joystick


```

while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
    if ((left_button == 1) || (right_button == 1) || (middle_button == 1))
    {
        //Send first frame
        current_pos = 0;

        Node1_TxData[0] = 0x10;
        Node1_TxData[1] = 0x0A; // 0x019 : 0000 0001 1001
        Node1_TxData[2] = 0x2E;

        //DATA ID
        Node1_TxData[3] = 0xF0;
        Node1_TxData[4] = 0x02;

        if(left_button == 1)
        {
            Node1_TxData[5] = LEFT_DATA[current_pos++];
            Node1_TxData[6] = LEFT_DATA[current_pos++];
            Node1_TxData[7] = LEFT_DATA[current_pos++];
        }
        else if(right_button == 1)
        {
            Node1_TxData[5] = RIGHT_DATA[current_pos++];
            Node1_TxData[6] = RIGHT_DATA[current_pos++];
            Node1_TxData[7] = RIGHT_DATA[current_pos++];
        }
        else if(middle_button == 1)
        {
            Node1_TxData[5] = MIDDLE_DATA[current_pos++];
            Node1_TxData[6] = MIDDLE_DATA[current_pos++];
            Node1_TxData[7] = MIDDLE_DATA[current_pos++];
        }

        data_length = (Node1_TxData[0] & 0x0f) + Node1_TxData[1];

        //printf("\n\n\nTester send FF: %x-%x-%x-%x-%x-%x-%x-%x\r\n", Node1_TxData[0],
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData, &Node1_TxMailbox);
    }
    HAL_Delay(10);
}

```

Hình 17: Code cho phần vòng lặp

Giải thích: Khi xuất hiện sự kiện Interrupt GPIO (Joystick được nhấn), đặt một biến **current_pos** để lưu vị trí hiện tại của chuỗi data từ joystick vào phần data của CAN

Gán các byte cho first frame theo như đã giải thích ở trên. Với 3 bytes cuối, gán giá trị của data từ joystick với thứ tự tăng dần của **current_pos** và ứng với từng phím joystick. Biến **data_length** dùng để lưu độ dài của chuỗi data mình muốn gửi (cụ thể là 10: 0x00A)

Dùng HAL_CAN_AddTxMessage để gửi dữ liệu đi.

1.2.3.3. Giải thích phần callback FIFO0

Vì cả ba trường hợp phím chỉ thay đổi data joystick nên dưới đây chỉ giải thích cụ thể phần left joystick, các phần còn lại tương tự.

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &Node1_RxHeader, Node1_RxData);
    if (Node1_RxHeader.StdId == 0x7A2 && (Node1_RxData[0] >> 4) == 0x3) //Flow control
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
        //printf("Tester received FC: %x-%x-%x-%x-%x-%x-%x-%x\r\n", Node1_RxData[0],Node1_RxData[1],Node1_RxData[2],Node1_RxData[3],Node1_RxData[4],Node1_RxData[5],Node1_RxData[6],Node1_RxData[7]);

        SNum = 0x00;

        if((data_length-3) % 7 == 0)
            number_CF = (data_length-3)/7;
        else
            number_CF = ((data_length-3)/7) + 1;

        if(left_button == 1 && SNum <= 0x0F)
        {
            for(int i = 0; i < number_CF; i++)
            {
                Node1_TxData[0] = 0x21 + SNum;
                SNum += 0x01;
                for(int j = 1; j<8; j++)
                {
                    if(current_pos <= SIGNAL_LEN)
                        Node1_TxData[j]=LEFT_DATA[current_pos++];
                    else
                        Node1_TxData[j] = 0x00;
                }

                //printf("Tester send CF: %x-%x-%x-%x-%x-%x-%x-%x\r\n", Node1_TxData[0],Node1_TxData[1],Node1_TxData[2],Node1_TxData[3],Node1_TxData[4],Node1_TxData[5],Node1_TxData[6],Node1_TxData[7]);
                HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData, &Node1_TxMailbox);
            }
            left_button = 0;
        }
    }
}
```

Hình 18: Code cho phần callback FIFO0

Khi call back fifo0 được gọi, trước tiên hàm HAL_CAN_GetRxMessage sẽ nhận data gửi từ MCU, sau đó kiểm tra StdId và 4bit đầu của gói để đảm bảo gói nhận được là gói Flow control.

Biến Snum (Sequence number) dùng để điền vào 4bit SN của gói Consecutive Frame

Xác định số lượng Consecutive frame cần sử dụng bằng data_length, trừ ba byte đã được truyền ở first frame, còn lại sẽ chia 7, nếu chia không hết thì sẽ thêm 1 frame cho phần dư.

Number_CF chứa số lượng Consecutive frame

Nếu nút left_joystick đã được nhấn và Sequence number <= 0x0F thì sẽ dùng vòng lặp để gửi số lượng number_CF frame. Với mỗi frame được gửi, byte đầu tiên là 0x21 + Snum. Các byte còn lại nếu current_pos <= số lượng byte data muốn gửi thì gán byte đó vào vị trí của CAN data, nếu không thì gán 0x00.

Dùng HAL_CAN_AddTxMessage để gửi data và trả lại biến left_button về 0.

1.2.3.4. Giải thích phần callback FIFO1

```

void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &Node2_RxHeader, Node2_RxData);
    if (Node2_RxHeader.StdId == 0x712 && Node2_RxData[2] == 0x2E && (Node2_RxData[0] >> 4) == 0x1 && Node2_RxData[3] == 0xF0 && Node2_RxData[4] == 0x02)
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port, LED2_Pin);
        printf("MCU received FF: %x-%x-%x-%x-%x-%x\r\n", Node2_RxData[0], Node2_RxData[1], Node2_RxData[2], Node2_RxData[3], Node2_RxData[4], Node2_RxData[5]);

        Node2_TxData[0] = 0x30;
        Node2_TxData[1] = 0x00;
        Node2_TxData[2] = 0x00;
        Node2_TxData[3] = 0x00; //2E + 40
        Node2_TxData[4] = 0x00;
        Node2_TxData[5] = 0x00;
        Node2_TxData[6] = 0x00;
        Node2_TxData[7] = 0x00;

        printf("MCU send FC: %x-%x-%x-%x-%x-%x\r\n", Node2_TxData[0], Node2_TxData[1], Node2_TxData[2], Node2_TxData[3], Node2_TxData[4], Node2_TxData[5]);
        HAL_CAN_AddTxMessage(&hcan2, &Node2_TxHeader, Node2_TxData, &Node2_TxMailbox);
    }

    if (Node2_RxHeader.StdId == 0x712 && (Node2_RxData[0] >> 4) == 0x2)
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port, LED2_Pin);
        printf("MCU received CF: %x-%x-%x-%x-%x-%x\r\n", Node2_RxData[0], Node2_RxData[1], Node2_RxData[2], Node2_RxData[3], Node2_RxData[4], Node2_RxData[5]);
    }
}

```

Hình 19: Code phần callback FIFO1

Khi call back fifo1 được gọi, trước tiên hàm HAL_CAN_GetRxMessage nhận data được gửi tới từ Tester. Kiểm tra headerStdId = 0x712, IdService = 2E, Frame Type = 0x01, DataID = 0xF002 (xác định là first frame) thì sẽ tiến hành in gói tin ra màn hình và gán lại giá trị cho gói tiếp theo là Flow Control đã được giải thích ở trên. Sau đó tiến hành gửi lại gói vừa cấu hình bằng hàm HAL_CAN_AddTxMessage.

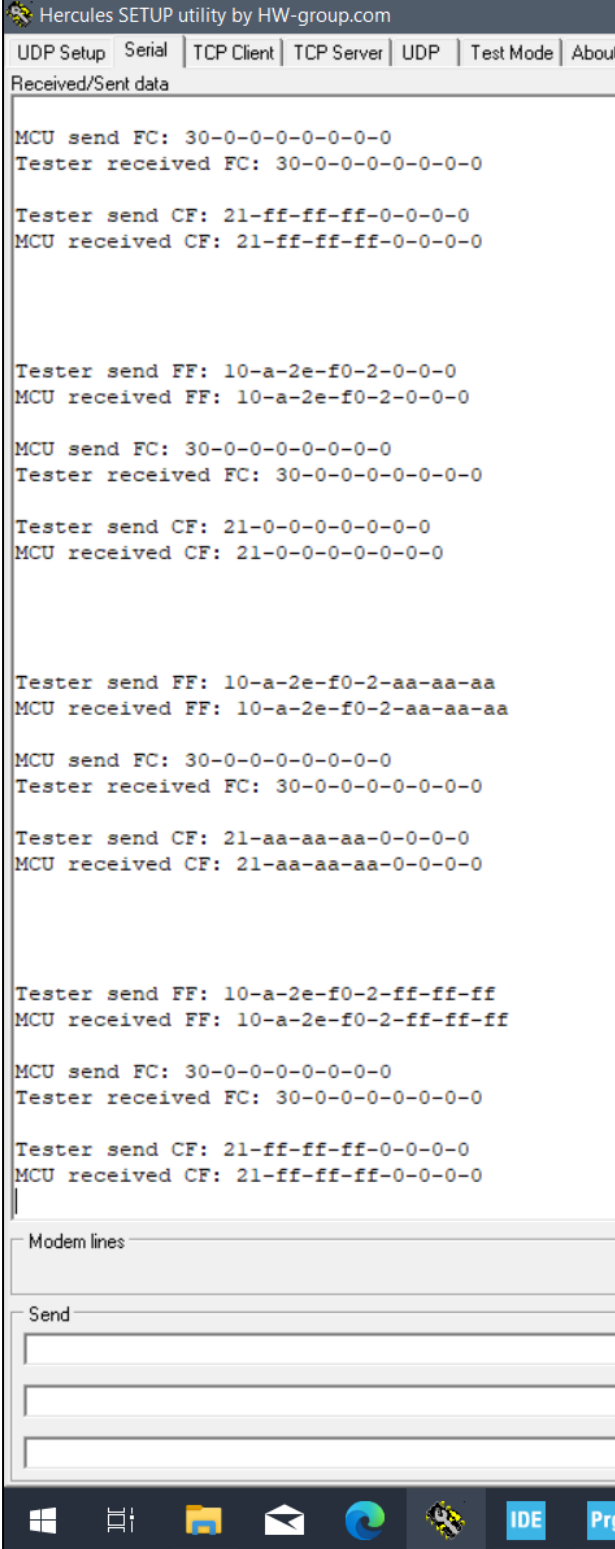
Nếu điều kiện trên không đúng, kiểm tra tiếp điều kiện headerStdId = 0x712 và Frame Type = 0x02 để xác định đây là consecutive frame và lấy dữ liệu in ra màn hình thông qua UART.

1.2.4. Kết quả

Kết quả được đặt ở đường dẫn dưới đây:

https://drive.google.com/file/d/1xAwBOuZMRRcIcRW3pGcV3tdZ6LaOOJiQ/view?usp=share_link

Kết quả với yêu cầu đề bài



UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received/Sent data

```
MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-ff-ff-ff-0-0-0-0
MCU received CF: 21-ff-ff-ff-0-0-0-0

Tester send FF: 10-a-2e-f0-2-0-0-0
MCU received FF: 10-a-2e-f0-2-0-0-0

MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-0-0-0-0-0-0-0
MCU received CF: 21-0-0-0-0-0-0-0

Tester send FF: 10-a-2e-f0-2-aa-aa-aa
MCU received FF: 10-a-2e-f0-2-aa-aa-aa

MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-aa-aa-aa-0-0-0-0
MCU received CF: 21-aa-aa-aa-0-0-0-0

Tester send FF: 10-a-2e-f0-2-ff-ff-ff
MCU received FF: 10-a-2e-f0-2-ff-ff-ff

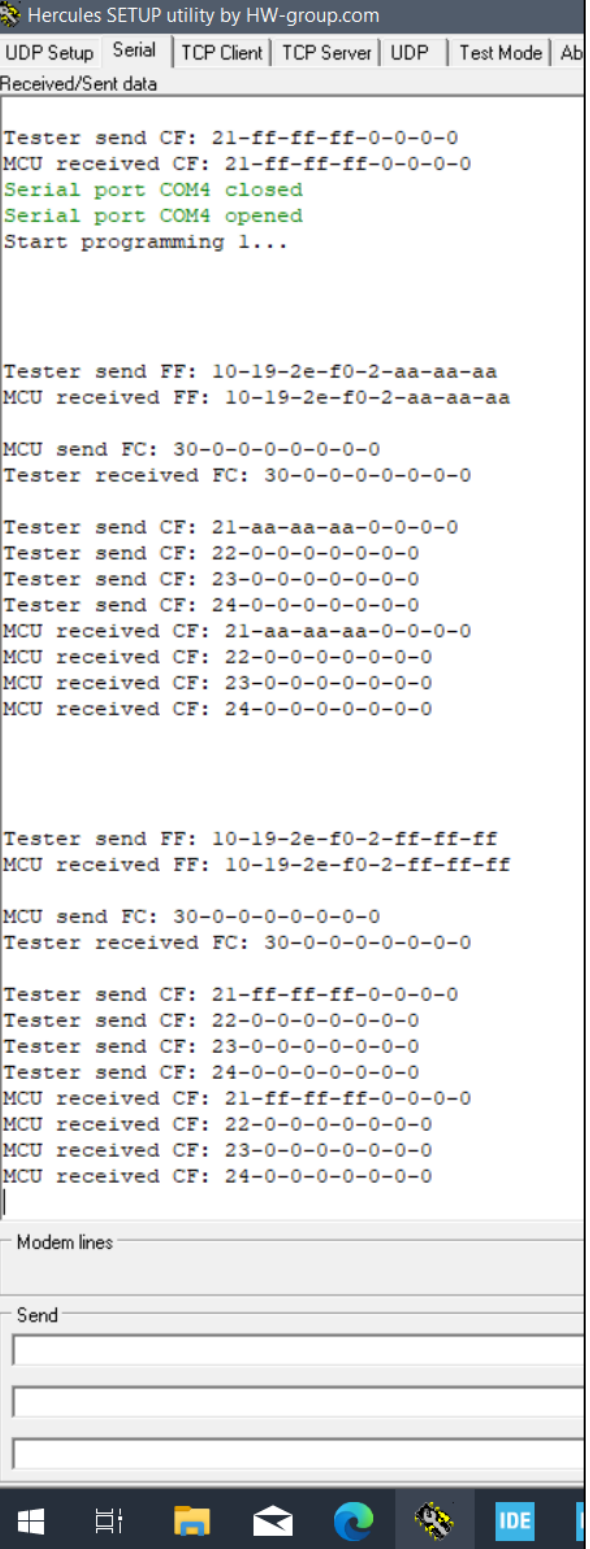
MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-ff-ff-ff-0-0-0-0
MCU received CF: 21-ff-ff-ff-0-0-0-0
```

Modem lines

Send

Kiểm tra khi gửi data với độ dài lớn hơn (dùng nhiều Consecutive frame hơn)



UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received/Sent data

```
Tester send CF: 21-ff-ff-ff-0-0-0-0
MCU received CF: 21-ff-ff-ff-0-0-0-0
Serial port COM4 closed
Serial port COM4 opened
Start programming 1...

Tester send FF: 10-19-2e-f0-2-aa-aa-aa
MCU received FF: 10-19-2e-f0-2-aa-aa-aa

MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-aa-aa-aa-0-0-0-0
Tester send CF: 22-0-0-0-0-0-0-0
Tester send CF: 23-0-0-0-0-0-0-0
Tester send CF: 24-0-0-0-0-0-0-0
MCU received CF: 21-aa-aa-aa-0-0-0-0
MCU received CF: 22-0-0-0-0-0-0-0
MCU received CF: 23-0-0-0-0-0-0-0
MCU received CF: 24-0-0-0-0-0-0-0

Tester send FF: 10-19-2e-f0-2-ff-ff-ff
MCU received FF: 10-19-2e-f0-2-ff-ff-ff

MCU send FC: 30-0-0-0-0-0-0-0
Tester received FC: 30-0-0-0-0-0-0-0

Tester send CF: 21-ff-ff-ff-0-0-0-0
Tester send CF: 22-0-0-0-0-0-0-0
Tester send CF: 23-0-0-0-0-0-0-0
Tester send CF: 24-0-0-0-0-0-0-0
MCU received CF: 21-ff-ff-ff-0-0-0-0
MCU received CF: 22-0-0-0-0-0-0-0
MCU received CF: 23-0-0-0-0-0-0-0
MCU received CF: 24-0-0-0-0-0-0-0
```

Modem lines

Send