BÁO CÁO THỰC HÀNH BÀI 6

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

Giảng viên hướng dẫn thực hành: Phạm Minh Quân

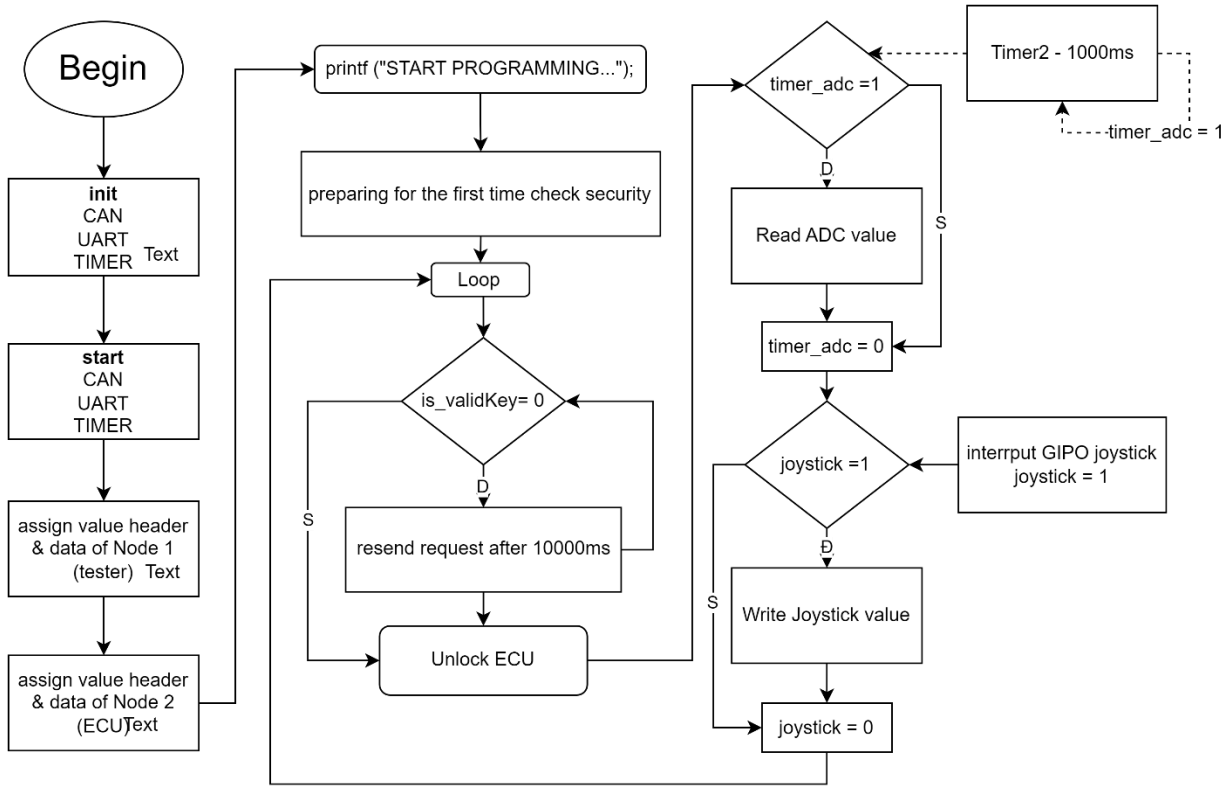
Thông tin các sinh viên	Mã số sinh viên	Họ và tên sinh viên
	19520887	Phạm Trung Quốc
	19521651	Phạm Trọng Huỳnh
	19520928	Viên Minh Tân
	19520036	Phạm Quốc Đăng
Link các tài liệu tham khảo (nếu có)		
Đánh giá của giảng viên: + Nhận xét + Các lỗi trong chương trình + Gợi ý		

MỤC LỤC

Nội dung thực hành

The SecuritySeed and SecurityKey are both 4 bytes (32 bits) number. The security access algorithms of each level would be defined and described in a specific file. The SecuritySeed is a random number except for two value 00000000h and FFFFFFFFh. Only one security level shall be active at any instant of time. If the tester sends an invalid key, the request is rejected by ECU and insert 10s delay before it can receive and process next seed request.

1. Sơ đồ giải thuật



Hình 1 Sơ đồ giải thuật

2. Mã nguồn

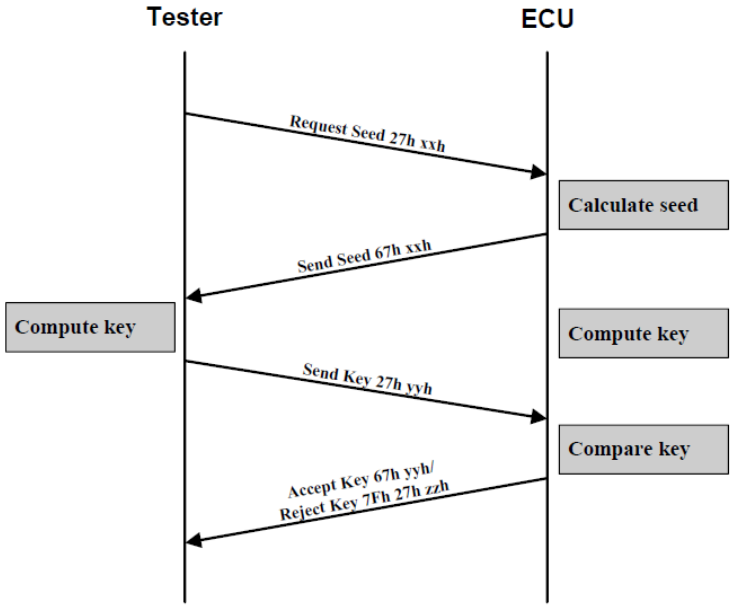
2.1. Giải thích timer

Sử dụng TIMER có chu kỳ tràn 1s cho việc đọc giá trị ADC. Sử dụng TIMER có chu kỳ tràn 10s cho việc gửi lại yêu cầu khi trong việc kiểm tra kết nối trong việc sủ dụng dịch vụ $27H Sercurity. Sử dụng công thức sau để cấu hình cho TIMER.

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

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
        if(htim->Instance == htim2.Instance)
        {
                timer_adc = 1;
        }
        if(htim->Instance == htim3.Instance)
        {
                timer_10s = 1;
        }
}
```

2.2. Phân tích gói tin dịch vụ SecurityAccess service ($27H)



Hình 2 Dịch vụ Security Access

Để có thể thực hiện các dịch khác, chúng ta cần sử dụng dịch vụ Sercurity Access để kiểm tra việc giao tiếp liệu có khả thi trước khi sử dụng các dịch vụ khác. Với mô hình kiểm tra được diễn ra như hình trên, Tester gửi một gói tin yêu cầu một seed từ ECU, ECU sau khi nhận được yêu cầu này sẽ tạo ra một seed và gửi lại cho Tester. Cả hai bên ECU và Tester sẽ giải mã seed này ra thành một key và sau đó kiểm tra liệu rằng hai key này đã giống nhau hoặc khác nhau, ECU sẽ gửi lại cho Tester một lệnh cho phép hoặc từ chối giao tiếp với tester. Các gói tin được phân tích ra như sau.

Request Seed					
Byte 1		Byte 2		Byte 3	
FT	DL	SID		SF	
0	2	2	7	0	1

Send seed							
Byte 1		Byte 2		Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
FT	DL	SID		SF	Security Seed random			
0	6	6	7	0	1			

Seed key								
Byte 1		Byte 2		Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
FT	DL	SID		SF	Security Seed random			
0	6	2	7	0	2	+ 1	+ 1	+ 1	+ 1

Response Access (accept)					
Byte 1		Byte 2		Byte 3	
FT	DL	SID		SF	
0	2	6	7	0	2

Response Access (Reject)							
Byte 1		Byte 2		Byte 3		Byte 4	
FT	DL	SID		SF		NRC	
0	3	7	F	6	7	3	5

Với FT là Frame Type, DL là Data length (số các data theo sau), SID là ID của dịch vụ chúng ta đang sử dụng, SF là Sub Fuction (ở đây với sub function là lẻ khi gửi các gói tin liên quan đến seed, là chẵn khi các gói tin liên quan đến key).

Với Security Seed được tạo ngẫu nhiên ở ECU và gửi về cho Tester (với 4 bytes). ECU được unlock các dịch vụ nếu gói tin phản hồi về không chứa SID là 0x7F (navigative response), và nếu nhận được gói tin có chứa NRC là 0x35 (invalid key).

2.3. Lập trình cho dịch vụ SecurityAccess service ($27H)

Chuẩn bị cho gói tin để Tester Request seed đến ECU. Với việc định nghĩa phần header cho các gói tin. Gọi hàm HAL_CAN_AddTxMessage để kích hoạt hàm HAL_CAN_RxFifo1MsgPendingCallback cho việc nhận ECU có thể nhận được gói tin từ Tester. Phần code này được thực hiện 1 lần trước khi vào hàm loop.

```
Node1_TxHeader.DLC = 8;  // data length
Node1_TxHeader.IDE = CAN_ID_STD;
Node1_TxHeader.RTR = CAN_RTR_DATA;
Node1_TxHeader.StdId = 0x712;  // ID

Node1_TxData[0] = 0x02; //Type 0: Single Frame
Node1_TxData[1] = 0x27; //Service ID
Node1_TxData[2] = 0x01;

Node2_TxHeader.DLC = 8;  // data length
Node2_TxHeader.IDE = CAN_ID_STD;
Node2_TxHeader.RTR = CAN_RTR_DATA;
Node2_TxHeader.StdId = 0x7A2;  // ID
caseFalse = 1;
      HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData,
       &Node1_TxMailbox);
```

Chuẩn bị cho gói tin Send seed, sau khi ECU nhận được request từ Tester, chúng ta cần chuẩn bị một gói tin để ECU gửi Seed về Tester. Ở đây chúng ta có hàm Random giá trị cho Security Seed tên là RandomSeed(), hàm này sẽ ngẫu nhiên ra một số có 4 bytes (32 bits) và lưu vào biến Seed. Biến Seed này được cắt ra và lưu vào byte 4, byte 5, byte 6 và byte 7 của gói tin Send Seed.

Đồng thời tại ECU chúng ta tính Key cho ECU được lưu và KeyECU để so sánh với Key mà Tester giải mã. Sau đó chúng ta gọi hàm HAL_CAN_AddTxMessage để gửi đi gói tin.

```
if (Node2_RxHeader.DLC == 8 && Node2_RxData[1] == 0x27 && Node2_RxData[2] == 0x01)
{
        printf("Request seed\r\n");
        Node2_TxData[0] = 0x06;
        Node2_TxData[1] = Node2_RxData[1] + 0x40;
        Node2_TxData[2] = 0x01;
        RandomSeed();
        printf("Seed: %x\r\n",Seed);
        Node2_TxData[3] = (Seed & 0xff000000UL) >> 24;
        Node2_TxData[4] = (Seed & 0x00ff0000UL) >> 16;
        Node2_TxData[5] = (Seed & 0x0000ff00UL) >>  8;
        Node2_TxData[6] = (Seed & 0x000000ffUL);
        KeyECU = Seed + 0x01010101 + caseFalse;
        printf("KeyECU: %x\r\n",KeyECU);
        HAL_CAN_AddTxMessage(&hcan2, &Node2_TxHeader, Node2_TxData,
                                        &Node2_TxMailbox);
}
```

Chuẩn bị cho gói tin Send key. Sau khi Tester đã nhận được Seed từ ECU, tại Tester sẽ sử dụng giải thuật để giải mã Seed ra key, và tiến hành gửi đi để ECU check liệu rằng có khả thi cho việc tiếp tục giao tiếp.

```
if (Node1_RxHeader.DLC == 8 && Node1_RxData[1] == 0x67 && Node1_RxData[2] == 0x01)
{
        Node1_TxData[0] = 0x06;
        Node1_TxData[1] = 0x27;
        Node1_TxData[2] = 0x02;
        Node1_TxData[3] = Node1_RxData[3] + 1;
        Node1_TxData[4] = Node1_RxData[4] + 1;
        Node1_TxData[5] = Node1_RxData[5] + 1;
        Node1_TxData[6] = Node1_RxData[6] + 1;
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData,
                &Node1_TxMailbox);
}
```

Chuẩn bị cho gói tin Response Access. Sau quá trình kiểm tra liệu rằng 2 key bên tester giải mã và bên ECU có giống nhau hay không. Nếu giống nhau hoặc khác nhau, ECU sẽ gửi lại một gói tin Response Access để cho Tester biết có khả thi giao tiếp hay không.

```
if (Node2_RxHeader.DLC == 8 && Node2_RxData[1] == 0x27 && Node2_RxData[2] == 0x02)
{
KeyTester = (Node2_RxData[3] << 24) + (Node2_RxData[4] << 16) + (Node2_RxData[5] << 8) + Node2_RxData[6];
printf("Keytester: %x\r\n\n",KeyTester);
if (KeyTester == KeyECU)
{
        Node2_TxData[0] = 0x02;
        Node2_TxData[1] = Node2_RxData[1] + 0x40;
        Node2_TxData[2] = 0x02;
}
else
{
        Node2_TxData[0] = 0x03;
        Node2_TxData[1] = 0x7F;
        Node2_TxData[2] = Node2_RxData[1] + 0x40;
        Node2_TxData[3] = 0x35;
}
HAL_CAN_AddTxMessage(&hcan2, &Node2_TxHeader, Node2_TxData,
                              &Node2_TxMailbox);
}
```

Sử dụng cờ để quyết định đã được unlock ECU chưa. Sau khi Tester nhận được gói tin Response Access, một số cờ sẽ được bật hoặc tắt để xác định quyền tiếp tục giao tiếp với ECU.

Nếu nhận được gói tin cho phép, cờ is_valiedKey được bật lên 1 để những phần code trong while có thể bắt đầu được kích hoạt, cũng như cờ is_start lên 1 để cho các dòng code bên trong while phải chờ sau khi cờ bật mới được thục hiện. Tại đây timer 2 cũng bắt đầu đếm đến hết 1s, phục vụ cho việc đọc giá trị ADC được bắt đầu đếm từ đây.

Ngược lại nếu không nhận được gói tin cho phép, các cờ cũng tắt để bắt chương trình phải chờ 10s sau đó mới được gửi 1 gói tin request khác.

```
if (Node1_RxHeader.DLC == 8 && Node1_RxData[1] == 0x67 && Node1_RxData[2] == 0x02)
{
        printf("unlock ECU\r\n\n");
        is_validKey = 1;
        timer_10s = 0;
        is_start = 1;
        HAL_TIM_Base_Start_IT(&htim2);
}
if (Node1_RxHeader.DLC == 8 && Node1_RxData[1] == 0x7F && Node1_RxData[2] == 0x67)
{
        printf("Nagative response from ECU!\r\n");
        if (Node1_RxData[3] == 0x35)
        {
                printf("Invalid key!\r\n");
                printf("\nPerpare for request new seed...\r\n\n");
                is_validKey = 0;
                is_start = 1;
                HAL_TIM_Base_Start_IT(&htim3);
        }
}
```

Giải quyết việc gửi lại request sau 10s nếu ECU vẫn chưa được unlock. Với hàm if này được đặt trong while, có nghĩa nếu Key nhận vẫn chưa giống nhau, chương trình sẽ phải chờ 10s mới có thể tiếp tục cho đến khi nhận được key giống nhau chương trình mới thực sự bắt đầu cho phép thực hiện các dịch vụ khác.

```
if (is_validKey == 0 && timer_10s == 1 && is_start == 1)  // timer10s = 1 (c
        {
                Node1_TxHeader.DLC = 8;  // data length
                Node1_TxHeader.IDE = CAN_ID_STD;
                Node1_TxHeader.RTR = CAN_RTR_DATA;
                Node1_TxHeader.StdId = 0x712;  // ID

                Node1_TxData[0] = 0x02; //Type 0: Single Frame
                Node1_TxData[1] = 0x27; //Service ID
                Node1_TxData[2] = 0x01;
                caseFalse = 0;
                HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData,
                                &Node1_TxMailbox);
        timer_10s = (timer_10s == 0) ? 1 : 0;
        HAL_TIM_Base_Stop_IT(&htim3);
        }
```

2.4. Dịch vụ 22H Read data by Identifier

Sau khi nhận được sự cho phép giao tiếp, dịch vụ Read data by Identifier sẽ tiến hành đọc dữ liệu ADC với mô hình đọc được thực hiện tương tự như ở bài 1 của bài thực hành số 4.

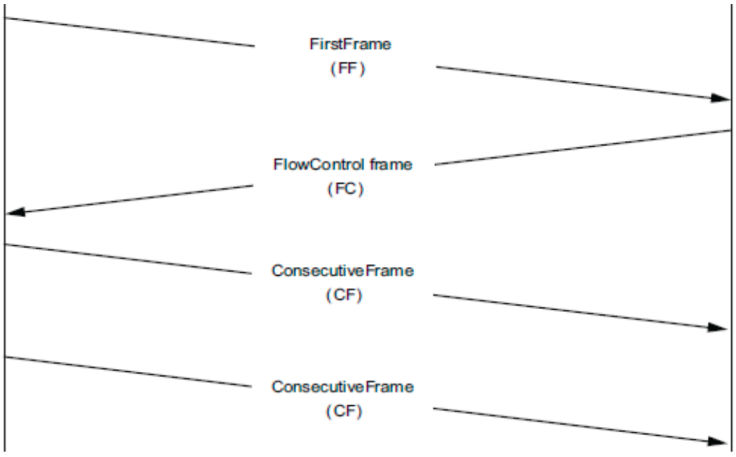
```
if (is_validKey == 1 && timer_adc == 1 && timer_10s == 0 && is_start == 1)
{
        Node1_TxData[0] = 0x07; //Type 0: Single Frame
        Node1_TxData[1] = 0x22; //Service ID
        Node2_TxData[0] = 0x07;
        Node2_TxData[4] = 0x00;
        Node2_TxData[5] = 0x00;
        Node2_TxData[6] = 0x00;
        Node2_TxData[7] = 0x00;
        HAL_ADC_Start_IT(&hadc1);
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData,
                &Node1_TxMailbox);
        timer_adc = 0; // flag for timer
}
```

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.

Giá trị ADC sẽ được đọc và gửi về Tester và đọc thông qua UART cứ sau mỗi 1000ms. Mỗi khi vặn núm xuay trên biến trở, giá trị ADC đọc về sẽ thay đổi theo vị trí vặn.

2.5. Dịch vụ 2EH Write data by Identifier

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 3 Dịch vụ Write data by Identifier

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.

```
if ((is_validKey == 1) && ((left_button == 1) || (right_button == 1) || (middle_button == 1))
                                && (timer_10s == 0) && (is_start == 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;

        Node2_TxData[0] = 0x00;
        Node2_TxData[1] = 0x00;
        Node2_TxData[2] = 0x00;
        Node2_TxData[3] = 0x00;
        Node2_TxData[4] = 0x00;
        Node2_TxData[5] = 0x00;
        Node2_TxData[6] = 0x00;
        Node2_TxData[7] = 0x00;

        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];
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, Node1_TxData, &Node1_TxMailbox);
}
HAL_Delay(10);
}
```

3. Kết quả

Chúng biến quan trọng để giúp giả lập trường hợp lỗi trong quá trình gửi và nhận Seed và Key qua CAN, đó là biến caseFalse, khi bật lên 1, có nghĩa chúng ta đang thiết lập trường hợp lỗi, và bằng 0 nếu chúng ta cho chương trình nhận đúng key. Lí do Key này có thể quyết định vì biến này sẽ được cộng thêm vào giải thuật giải mã của ECU (từ Seed ra Key). Với dòng code này, cúng ta sẽ thiết lập được 2 trường hợp, tiện cho việc debug trong thực tế.

KeyECU = Seed + 0x01010101 + caseFalse;

3.1. Trường hợp 1: Cho phép giao tiếp ngay lần đầu tiên

Với việc xuất ra UART dòng "START PROGRAMMING…" giúp chúng ta nhận biết chương trình bắt đầu tại đâu trong các dòng dữ liệu in ra UART.

Đầu tiên hệ thống sử dụng dịch vụ kiểm tra có quyền giao tiếp giữa ECU và Tester không. Hiển thị seed được tạo, Key được tính toán ở ECU và Tester để tiện theo dõi quá trình tính toán. Sau khi nhận thấy giống Key ở Tester và ECU, ECU unlock cho các dịch vụ để thực hiện.

```
13:54:50.194 -> START PROGRAMMING...
13:54:50.194 ->
13:54:50.194 -> Check security!
13:54:50.194 -> Request seed
13:54:50.194 -> Seed: 5851f42e
13:54:50.194 -> KeyECU: 5952f52f
13:54:50.194 -> Keytester: 5952f52f
13:54:50.194 ->
13:54:50.194 -> unlock ECU
13:54:50.194 ->
13:54:50.194 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:54:51.189 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:54:52.184 -> Tester received ADC value: 7-62-8-df-0-0-0-0
13:54:53.212 -> Tester received ADC value: 7-62-8-de-0-0-0-0
13:54:53.975 ->
13:54:53.975 ->  MCU received FF: 10-a-2e-f0-2-aa-aa-aa
13:54:53.975 -> Tester received FC: 30-0-0-0-0-0-0-0
13:54:54.022 -> MCU received CF: 21-aa-aa-aa-0-0-0-0
13:54:54.022 ->
13:54:54.207 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:54:55.202 -> Tester received ADC value: 7-62-8-dd-0-0-0-0
13:54:56.198 -> Tester received ADC value: 7-62-8-dd-0-0-0-0
13:54:57.193 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:54:57.625 ->
13:54:57.625 ->  MCU received FF: 10-a-2e-f0-2-ff-ff-ff
13:54:57.625 -> Tester received FC: 30-0-0-0-0-0-0-0
13:54:57.658 -> MCU received CF: 21-ff-ff-ff-0-0-0-0
13:54:57.658 ->
13:54:58.188 -> Tester received ADC value: 7-62-8-dd-0-0-0-0
13:54:59.184 -> Tester received ADC value: 7-62-8-dd-0-0-0-0
13:55:00.212 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:55:01.208 -> Tester received ADC value: 7-62-8-dc-0-0-0-0
13:55:02.004 ->
13:55:02.004 ->  MCU received FF: 10-a-2e-f0-2-0-0-0
13:55:02.004 -> Tester received FC: 30-0-0-0-0-0-0-0
13:55:02.004 -> MCU received CF: 21-0-0-0-0-0-0-0
13:55:02.004 ->
```

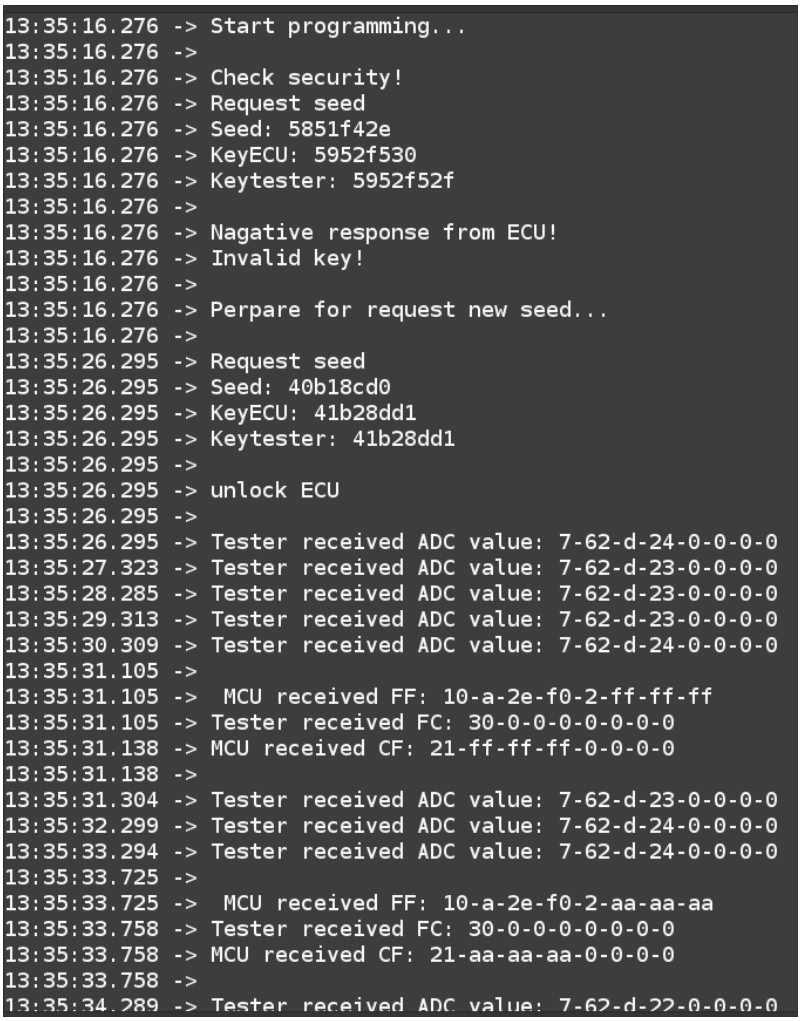
Hình 4 Trường hợp 1 - Cho phép giao tiếp ngay lần đầu tiên

Ngày sau khi được unlock cho các dịch vụ khác, ta thấy rằng giá trị ADC đã liên tục được đọc về với chu kỳ 1000ms. Giá trị ADC được lưu tại byte thứ 3 và thứ 4 của gói tin được in ra trên màn hình.

Mỗi khi chúng ta nhấn Joystick, 3 gói tin được in ra màn hình lần lượt là FF, FC, và CF. Với gói tin FF và CF chứa dữ liệu cần thiết là vị trí nút nhấn. Ở hình bên dưới, nhóm đã thử các nút trái, phải giữa, và tất cả đều hoạt động gửi đúng gói tin mong muốn. Trong khi đó ADC vẫn sẽ liên tục được đọc sau mỗi 1000ms.

3.2. Trường hợp 2: Không cho phép giao tiếp lần đầu

Với việc xuất ra UART dòng "START PROGRAMMING…" giúp chúng ta nhận biết chương trình bắt đầu tại đâu trong các dòng dữ liệu in ra UART.



Hình 5 Trường hợp 2 - Không cho phép giao tiếp lần đầu

Đầu tiên hệ thống sử dụng dịch vụ kiểm tra có quyền giao tiếp giữa ECU và Tester không. Hiển thị seed được tạo, Key được tính toán ở ECU và Tester để tiện theo dõi quá trình tính toán. Ở trường hợp này nhóm mô phỏng việc Key không gióng nhau ở ECU và Tester. Chúng ta có thể thấy 10000ms một request khác được gửi lại (ở đây nhóm thiết lập sẽ gửi đúng ở lần thứ 2). Sau khi nhận thấy giống Key ở Tester và ECU, ECU unlock cho các dịch vụ để thực hiện.

Ngày sau khi được unlock cho các dịch vụ khác, ta thấy rằng giá trị ADC đã liên tục được đọc về với chu kỳ 1000ms. Giá trị ADC được lưu tại byte thứ 3 và thứ 4 của gói tin được in ra trên màn hình.

Mỗi khi chúng ta nhấn Joystick, 3 gói tin được in ra màn hình lần lượt là FF, FC, và CF. Với gói tin FF và CF chứa dữ liệu cần thiết là vị trí nút nhấn. Ở hình bên dưới, nhóm đã thử các nút trái, phải giữa, và tất cả đều hoạt động gửi đúng gói tin mong muốn. Trong khi đó ADC vẫn sẽ liên tục được đọc sau mỗi 1000ms.

4. Source code

Link: Lab_STM32F4xx/Lab6_Security at main · PQD-11/Lab_STM32F4xx (github.com)