BÁO CÁO ĐỒ ÁN CUỐI KỲ

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 sinh viên** | Mã số sinh viên: 19520571  Họ và tên: Tô Quang Huấn |
| **Link các tài liệu tham khảo** *(nếu có)* |  |
| **Đánh giá của giảng viên**:  *+ Nhận xét*  *+ Các lỗi trong chương trình*  *+ Gợi ý* |  |

**MỤC LỤC**

[I. Phân tích yêu cầu 3](#_Toc121496863)

[1. Yêu cầu bài toán 3](#_Toc121496864)

[2. Phân tích bài toán 3](#_Toc121496865)

[II. Cấu hình hệ thống 6](#_Toc121496866)

[1. Cấu hình CAN 6](#_Toc121496867)

[2. Cấu hình UART 8](#_Toc121496868)

[3. Cấu hình GPIO 9](#_Toc121496869)

[III. Viết chương trình 11](#_Toc121496870)

[1. Cấu hình ID truyền và nhận của CAN Node 1 và CAN Node 2 11](#_Toc121496871)

[2. Hiện thực Service $27H – Security access 12](#_Toc121496872)

[3. Yêu cầu Security Access đối với các dịch vụ khác 15](#_Toc121496873)

[4. Kết quả hiện thực Service $27H 16](#_Toc121496874)

# Phân tích yêu cầu

## Yêu cầu bài toán

Thực hiện Service $27H (Security Access) của chuẩn giao tiếp CAN để mở khóa ECU với các yêu cầu như sau:

* Tester sẽ tiến hành gởi yêu cầu seed và quá trình mở khóa sẽ được thực hiện mỗi khi người dùng nhấn vào một user button.
* Tester sẽ thực hiện gởi yêu cầu seed và key tới ECU. Từ đó, ECU sẽ cung cấp seed và kiểm tra key. Nếu key trùng khớp thì ECU sẽ gởi positive response đến Tester, chuyển trạng thái từ Locked sang Unlocked. Nếu key không trùng khớp thì tiến hành delay 10 giây trước khi ECU tiến hành nhận một yêu cầu mới từ Tester.
* LED để báo hiệu ECU đang ở trạng thái Unlocked và ngược lại LED sẽ tắt khi ECU ở trạng thái Locked.
* Các Sub-service được sử dụng: 0x01 - Request Seed L1.

0x02 - Send Key L1.

* Độ dài của Seed và Key: 4 bytes Seed (S12 S34 S56 S78)

4 bytes Key (K12 K34 K56 K78)

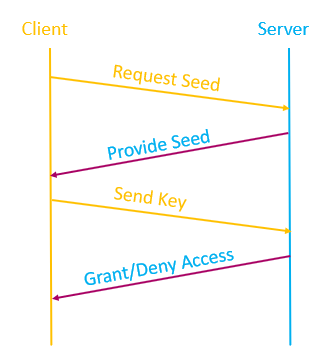
* Thuật toán sinh ra Seed: Random Number
* Thuật toán kiểm tra Key: K12 = S12 + 1

K34 = S34 + 1

K56 = S56 + 1

K78 = S78 + 1

## Phân tích bài toán



Hình : Sơ đồ quy trình hoạt động của Service $27H

Ý tưởng chính dịch vụ $27H - (Security Access) là sử dụng một giải thuật seed và key gồm các bước hiện thực như mô tả sau:

* Bước đầu tiên, Tester gởi yêu cầu đến ECU để mở khóa bằng cách gởi gói tin RequestSeed.
* Sau khi nhận được gói tin RequestSeed, ECU sẽ thực hiện hồi đáp bằng một gói tin chứa dữ liệu seed sử dụng cấu trúc gói tin RequestSeed positive response message. Dữ liệu seed là đầu vào cho cả Tester và ECU để thực hiện tính toán các giá trị key tương ứng.
* Tiếp theo, Tester sẽ gởi yêu dữ kiệu key trả về cho ECU sử dụng cấu trúc gói tin SendKey message.
* Một khi ECU nhận được gói tin chứa dữ liệu key, nó sẽ so sánh dữ liệu key nhận được với giá trị key mà nó tính được dựa vào dữ liệu seed. Nếu 2 dữ liệu key trùng khớp, ECU cho phép mở khóa, Tester có thể truy cập vào các dịch vụ chỉ định hoặc dữ liệu của ECU, ECU sử dụng gói tin SendKey positive response message để cho Tester biết được ECU đã cho phép mở khóa. Ngược lại nếu 2 dữ liệu không trùng khớp, nghĩa là điều kiện truy cập ECU không hợp lệ, lúc này, yêu cầu truy cập bị ECU từ chối và ECU sẽ delay 10 giây trước khi nó nhận một yêu cầu truy cập mới.

Theo quy trình trên, các cấu trúc gói tin truyền nhận CAN được sử dụng để hiện thực Service $27H như sau:

* Gói tin RequestSeed:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Parameter Name** | **Cvt** | **Value (HEX)** |
| #1 | RequestServiceIdentifier | M | 27 |
| #2 | Sub-Function = [  RequestSeed] | M | 01, 09 |

* Gói tin RequestSeed positive response message:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Parameter Name** | **Cvt** | **Value (HEX)** |
| #1 | PositiveResponseServiceIdentifier | M | 67 |
| #2 | Sub-Function = [  RequestSeed] | M | 01, 09 |
| #3  #4  #5  #6 | SecuritySeed[] = [  seed#1 (high byte)  seed#2  seed#3  seed#4 (low byte)] | M  M M  M | 00-FF  00-FF  00-FF  00-FF |

* Gói tin SendKey:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Parameter Name** | **Cvt** | **Value (HEX)** |
| #1 | RequestServiceIdentifier | M | 27 |
| #2 | Sub-Function = [  SendKey] | M | 02, 0A |
| #3  #4  #5  #6 | SecurityKey[] = [  key#1 (high byte)  key#2  key#3  key#4 (low byte)] | M  M M  M | 00-FF  00-FF  00-FF  00-FF |

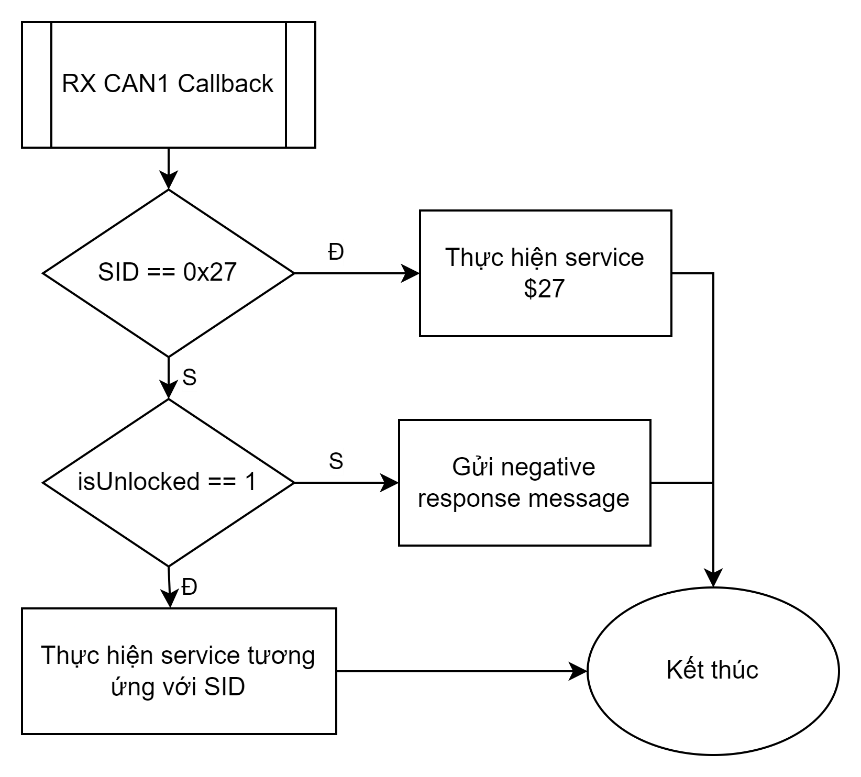
* Gói tin SendKey positive response message:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Parameter Name** | **Cvt** | **Value (HEX)** |
| #1 | PositiveResponseServiceIdentifier | M | 67 |
| #2 | Sub-Function = [  SendKey] | M | 02, 0A |

* Gói tin SendKey negative response message:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Parameter Name** | **Cvt** | **Value (HEX)** |
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | SID | M | 27 |
| #3 | NegativeResponseCode | M | 35 |

Sau khi đã phân tích xong các bước hoạt động và các gói tin của service $27, ta tiến hành áp dụng chúng vào hệ thống chung để có được sự bảo mật và an toàn khi thực hiện các dịch dụ khác. Lưu đồ phía dưới sẽ khái quát hóa quy trình hoạt động của chúng:



*Hình 2: Sơ đồ hiện thực Security Access*

Đầu tiên, ta cấu hình rằng CAN1 sẽ là chuẩn giao tiếp tại ECU, cờ isUnlocked đại diện cho trang thái của ECU và được khởi tạo là không, SID là mã dịch vụ được yêu cầu. Bất cứ khi nào CAN1 nhận được gói tin dịch vụ, hàm callback tương ứng sẽ được gọi. Lúc này, ECU sẽ kiểm tra xem SID của dịch vụ vừa nhận được. Lúc này ta có các trường hợp sau:

* Trong trường hợp SID bằng 0x27 thì ECU sẽ tiến hành thực hiện service $27 theo các bước đã trình bày ở phần trên. Nếu xác nhận rằng Key nhận được từ Tester khớp với Key đã tính toán ở ECU thì isUnlocked sẽ được gán bằng một.
* Nếu SID khác 0x27 thì sẽ tiến hành kiểm tra isUnlocked có bằng một hay không. Nếu cờ isUnlocked bằng một thì sẽ thực hiện service tương ứng với SID trong gói tin nhận được. Nếu isUnlocked bằng không thì ta hiểu rằng ECU vẫn chưa được mở khóa.

# Cấu hình hệ thống

Đầu tiên chúng ta thực hiện các thao tác để cấu hình GPIO, UART và CAN.

Graphical user interface

Description automatically generated with medium confidence

Hình : Sơ đồ chân được cấu hình

## Cấu hình CAN

Tại phần Connectivity, chọn CAN1 và tick vào Activated để kích hoạt CAN1 và chọn các chân tương ứng cho CAN1 theo như sơ đồ mạch của nhà sản xuất để cấu hình các chức năng cho CAN1. Thực hiện các thao tác tương tự đối với CAN2.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Schematic các chân kết nối CAN

Tiếp đó, ta tiến hành cài đặt các thông số cho CAN1 theo các thông số yêu cầu như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Các thông số cấu hình cho CAN1

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình : Cấu hình NVIC Interrupt cho CAN1

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Các thông số cấu hình cho CAN2

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình : Cấu hình NVIC Interrupt cho CAN2

## Cấu hình UART

Khởi tạo UART để thực hiện truyền nhận và hiển thị dữ liệu, ở đây sử dụng USART1 tại 2 chân là PA9 cho USART1\_TX và PA10 cho USART1\_RX với baud rate là 115200bit/s. Ở đây chúng ta sử dụng mode Asynchronous cho UART bất đồng bộ.

Graphical user interface, text, application

Description automatically generated

Hình : Các config của USART1

## Cấu hình GPIO

Tiến hành khởi tạo 3 GPIO là PC4, PC7 và PC13 lần lượt tương ứng với 3 vị trí Left, Middle và Right của joystick theo thiết kế được sử dụng trong schematic. Bên cạnh đó, khởi tạo GPIO PC6 cho vị trí down của joystick, được sử dụng để yêu cầu giá trị ADC.

Diagram, schematic

Description automatically generated Table

Description automatically generated

Hình : Khai báo GPIO pin cho Joystick

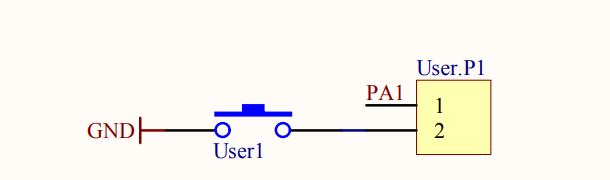
Đồng thời cũng sử dụng các GPIO này làm các interrupt ngoại để định danh những giá trị dữ liệu cụ thể theo để bài toán.

Table, timeline

Description automatically generated

Hình : Cấu hình các interrupt ngoại sử dụng GPIO

Ta cần sử dụng user button để phục vụ cho việc gởi gói tin yêu cầu Security Access, tiến hành khởi tạo thêm GPIO cho user button với chân PA1 và bật interrupt cho GPIO này.



Ảnh có chứa bàn

Mô tả được tạo tự động

Hình : Khai báo GPIO cho user button

# Viết chương trình

## Cấu hình ID truyền và nhận của CAN Node 1 và CAN Node 2

Đầu tiên cần khai báo 2 đối tượng cho 2 node CAN với Node 1 là ECU và Node 2 là Tester. Sau đó khai báo và gán 2 Node này vào struct được định nghĩa sẵn để dễ dàng handle các thông số của CAN.



Tiếp sau đó thực hiện thiết đặt các thông số cho 2 node CAN. Đối với CAN1, các ID cần phải thiết đặt là TX\_ID: 0x7A2 và RX\_ID là 0x712. Thực hiện thiết đặt trong chương trình như sau:

Thiết đặt TX\_ID thông qua dòng lệnh sau:



Thiết đặt RX\_ID thông qua đoạn lệnh sau:



Đối với CAN2, các ID cần phải thiết đặt là TX\_ID: 0x712 và RX\_ID là 0x7A2.

Thiết đặt TX\_ID thông qua dòng lệnh sau:



Thiết đặt RX\_ID thông qua đoạn lệnh sau:



Đồng thời cũng thực hiện thiết lập các thông số phục vụ cho xây dựng các gói tin sau này.



## Hiện thực Service $27H – Security access

Đầu tiên ta khởi tạo struct sendMessage để lưu trữ những thông tin cần thiết của gói tin được gửi đi. Trường serviceInfo lưu trữ SID, SF và DID, bufferLength là độ là buffer được dùng để xây dựng gói tin gửi đi, datalength là độ dài của dữ liệu cần được gửi và cuối cùng là con trỏ data là mảng động lưu trữ dữ liệu để cần được gửi hoặc nhận.

**typedef** **struct** {

uint8\_t SID; // Service ID - SID

uint8\_t SF; // Sub-Function

uint8\_t DID[2]; // Data ID - DID

} CANTPService;

**typedef** **struct** {

CANTPService serviceInfo; // Store SID, MSB\_DID, LSB\_DID

uint16\_t bufferLength; // Length of packet to be transmitted/received

uint16\_t dataLength; // Length of data to be transmitted/received

uint8\_t offset; // Offset of data array

uint8\_t \*data; // Array data transmitted/received

} CANTPMessage;

Ta quy định rằng khi nhấn user button thì Tester sẽ thực hiện security access đến ECU. Để làm được như vậy, ta cần sử dụng hàm callback của interrupt user button đã được cấu hình ở phần II. Sau đó tại Tester sẽ bắt đầu gửi gói tin Request Seed đến ECU thông qua hàm **sendSeedRequest.**

**void** **sendSeedRequest**(uint8\_t SubFunction)

{

**printf**("-------------Service $27H---------------\n");

sendMessage.serviceInfo.SID = 0x27;

sendMessage.serviceInfo.SF = SubFunction;

sendMessage.dataLength = 0;

sendMessage.bufferLength = **sizeof**(sendMessage.serviceInfo) + sendMessage.dataLength - 2;

CANTP(&sendMessage, &Tester);

}

Hàm **sendSeedRequest** được gọi với SubFunction được truyền vào hàm là 0x01. Như vậy, gói tin yêu cầu được truyền lúc này sẽ có dạng:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x07 | 0x27 | 0x01 | 0x55 | 0x55 | 0x55 | 0x55 | 0x55 |

Sử dụng hàm **CANTP()** truyền gói tin này đến ECU. Ngay sau đó, ECU sẽ nhận được gói tin Request Seed. Lúc này tại interrupt của CAN tại ECU sẽ kiểm tra gói tin yêu cầu cung cấp seed và gửi lại gói tin Provide Seed đến Tester thông qua hàm **responseSeedRequest()**.

**if**(receiveMessage.serviceInfo.SF == 0x01){

responseSeedRequest(&receiveMessage.serviceInfo, &ECU);

}

Hàm **responseSeedRequest()** thực hiện sinh ra dữ liệu seed bằng giải thuật random số bất kỳ. Ta thiết đặt giá trị random từ 0x00 đến 0xFF và cho random và nhận được 4 byte dữ liệu seed. Sau đó, tiến hành cấu tạo thành một gói tin dựa trên cấu trúc gói tin ***RequestSeed positive response message*** của Service $27H như đã trình bày ở phần I.

**void** **responseSeedRequest**(CANTPService \*ServiceInformation, CANTPCanInit \*can){

uint8\_t minNumber = 0x00;

uint8\_t maxNumber = 0xFF;

**srand**((**int**)**time**(0));

seed[0] = minNumber + **rand**() % (maxNumber + 1 - minNumber);

seed[1] = minNumber + **rand**() % (maxNumber + 1 - minNumber);

seed[2] = minNumber + **rand**() % (maxNumber + 1 - minNumber);

seed[3] = minNumber + **rand**() % (maxNumber + 1 - minNumber);

CANTPSingleFrame singleFrame;

singleFrame.indicate = 0x07;

singleFrame.payload[0] = ServiceInformation->SID;

singleFrame.payload[1] = ServiceInformation->SF;

singleFrame.payload[2] = seed[0];

singleFrame.payload[3] = seed[1];

singleFrame.payload[4] = seed[2];

singleFrame.payload[5] = seed[3];

singleFrame.payload[6] = 0x55;

**memcpy**(&(can->TxData), &singleFrame, **sizeof**(singleFrame));

HAL\_CAN\_AddTxMessage(&(can->hcan), &(can->TxHeader), can->TxData,

&(can->TxMailbox));

}

Như vậy, sau khi thực thi hàm **responseSeedRequest()**, ECU sẽ tiến hành gởi gói tin cung cấp seed đến Tester với cấu trúc như sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x07 | 0x67 | 0x01 | seed[0] | seed[1] | seed[2] | seed[3] | 0x55 |

Gói tin được truyền tới Tester thông qua CAN, Tester nhận được dữ liệu seed và tiến hành tính toán giá trị Key bằng giải thuật tính toán key theo yêu cầu của đề bài và gửi key cho ECU thông qua hàm **sendKey()** như sau:

**void** **sendKey**(CANTPMessage \*receiveMessage, CANTPCanInit \*can){

CANTPSingleFrame singleFrame;

singleFrame.indicate = 0x07;

singleFrame.payload[0] = receiveMessage->serviceInfo.SID - 0x40;

singleFrame.payload[1] = receiveMessage->serviceInfo.SF + 0x01;

singleFrame.payload[2] = receiveMessage->data[0] + 0x01;

singleFrame.payload[3] = receiveMessage->data[1] + 0x01;

singleFrame.payload[4] = receiveMessage->data[2] + 0x01;

singleFrame.payload[5] = receiveMessage->data[3] + 0x01;

singleFrame.payload[6] = 0x55;

**memcpy**(&(can->TxData), &singleFrame, **sizeof**(singleFrame));

HAL\_CAN\_AddTxMessage(&(can->hcan), &(can->TxHeader), can->TxData,

&(can->TxMailbox));

}

Với các giá trị Seed nhận được từ ECU, Tester tiến hành cộng mỗi byte lên 1 đơn vị tạo thành dữ liệu Key có độ dài 4 bytes, cập nhật lại gói tin truyền nhận CAN và gởi đến cho ECU, gói tin chứa dữ liệu Key có cấu tạo như sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x07 | 0x27 | 0x02 | key[0] | key[1] | key[2] | key[3] | 0x55 |

Về phía ECU, khi nhận được gói tin chứa dữ liệu key từ Tester, ECU sẽ gọi hàm **respondKey()** để thực hiện kiểm tra key và trả về gói tin hồi đáp cho Tester về việc Unlock.

**else** **if** (receiveMessage.serviceInfo.SF == 0x02)

{

responseKey(&receiveMessage, &ECU);

}

Đối với hàm **responseKey()**, ECU sẽ bắt đầu dựa vào các seed nó tạo ra trước đó để tính ra các giá trị key. Giá trị key này sẽ được đối chứng với dữ liệu nhận được từ gói tin sendKey của Tester.

**void** **responseKey**(CANTPMessage \*receiveMessage, CANTPCanInit \*can){

CANTPSingleFrame singleFrame;

singleFrame.indicate = 0x07;

key[0] = seed[0] + 0x01;

key[1] = seed[1] + 0x01;

key[2] = seed[2] + 0x01;

key[3] = seed[3] + 0x01;

**if** (key[0] == receiveMessage->data[0] && key[1] == receiveMessage->data[1]

&& key[2] == receiveMessage->data[2] && key[3] == receiveMessage->data[3]) {

singleFrame.payload[0] = receiveMessage->serviceInfo.SID;

singleFrame.payload[1] = receiveMessage->serviceInfo.SF;

singleFrame.payload[2] = 0x55;

singleFrame.payload[3] = 0x55;

singleFrame.payload[4] = 0x55;

singleFrame.payload[5] = 0x55;

singleFrame.payload[6] = 0x55;

isUnlocked = 1;

HAL\_GPIO\_WritePin(GPIOB, LED0\_Pin, isUnlocked);

showResponseFrame(&singleFrame);

**memcpy**(&(can->TxData), &singleFrame, **sizeof**(singleFrame));

HAL\_CAN\_AddTxMessage(&(can->hcan), &(can->TxHeader), can->TxData, &(can->TxMailbox));

}

**else** {

receiveMessage->serviceInfo.SID -= 0x40;

sendNegativeResponse(receiveMessage, can, 0x35);

**printf**("---> Invalid key - Delay 10s\n");

HAL\_CAN\_Stop(&hcan1);

HAL\_Delay(10000);

HAL\_CAN\_Start(&hcan1);

**printf**("Continue receiving service request\n");

}

}

Trường hợp kiểm tra Key trùng khớp, ECU sẽ tiến hành bật LED0 báo hiệu ECU đang ở trạng thái Unlocked và gửi positive response dựa trên cấu trúc gói tin ***SendKey positive response message***. Ngược lại, nếu Key không trùng khớp thì ECU sẽ không bật LED0 và gửi negative response đến Tester theo như cấu trúc gói tin ***SendKey negative response message***với Negative Response Code (NRC) là **0x35 – Invalid Key**, đồng thời tiến hành delay 10 giây trước khi nhận một yêu cầu mới.

**void** **sendNegativeResponse**(CANTPMessage \*receiveMessage, CANTPCanInit \*can,

uint8\_t NRC){

CANTPSingleFrame singleFrame;

singleFrame.indicate = 0x07;

singleFrame.payload[0] = 0x7F;

singleFrame.payload[1] = receiveMessage->serviceInfo.SID;

singleFrame.payload[2] = NRC;

singleFrame.payload[3] = 0x55;

singleFrame.payload[4] = 0x55;

singleFrame.payload[5] = 0x55;

singleFrame.payload[6] = 0x55;

**memcpy**(&(can->TxData), &singleFrame, **sizeof**(singleFrame));

HAL\_CAN\_AddTxMessage(&(can->hcan), &(can->TxHeader), can->TxData,

&(can->TxMailbox));

}

Dưới đây là gói tin ***negative response message*** được gửi:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x07 | 0x7F | 0x027 | 0x35 | 0x55 | 0x55 | 0x55 | 0x55 |

Đến đây, quá trình hiện thực Service $27H đã được hoàn tất, các phần sau sẽ là tích hợp dịch vụ $27H vào các dịch vụ chỉ định và kiểm chứng hiện thực.

## Yêu cầu Security Access đối với các dịch vụ khác

Trạng thái của ECU được lưu trữ vào biến boolean **isUnlocked**, biến này sẽ cho biết ECU đang ở trạng thái nào. Nếu giá trị của biến bằng 0 thì cho biết ECU đang ở trạng thái Locked và nếu giá trị của biến bằng 1 thì cho biết ECU đang ở trạng thái Unlocked.

**static** uint8\_t isUnlocked = 0; // locked: 0; unlocked: 1

Tiến hành hiện thực đối với các dịch vụ chỉ định phải được truy cập cần có Security Access, trong bài lab này, nhóm tiến hành hiện thực mô phỏng dựa trên dịch vụ $2EH, dịch vụ cho phép ghi dữ liệu xuống ECU. Mỗi khi có yêu cầu dịch vụ $2EH để ghi dữ liệu xuống ECU, ECU sẽ kiểm tra xem nó đã được Unlock hay chưa. Trường hợp ECU chưa ở trạng thái Unlocked, Tester sẽ bị từ chối yêu cầu ghi dữ liệu xuống ECU và ECU sẽ gửi một gói tin ***negative response message*** đến Tester.

**if** (!isUnlocked) {

**if**(frameType == 0x00)

receiveSingleFrame(&receiveMessage, ECU.RxData);

**else**

receiveFirstFrame(&receiveMessage, ECU.RxData);

sendNegativeResponse(&receiveMessage, &ECU, 0x33);

**printf**("\*\*\* ECU locked \*\*\*\n");

**return**;

}

Ngược lại, nếu ECU đang ở trạng thái Unlocked thì yêu cầu về dịch vụ ghi dữ liệu xuống ECU sẽ được thực hiện theo trình tự của dịch vụ.

## Kết quả hiện thực Service $27H

Sau khi đã hiện thực Service $27H và kết hợp yêu cầu Security Access vào các dịch vụ được chỉ định cần có, ta sẽ kiểm tra kết quả thông qua các trường hợp sau:

* Ghi dữ liệu xuống ECU thông qua Joystick (service $2E) trước khi thực hiện service $27: Tester sẽ ghi dữ liệu thông qua Joystick. Tuy nhiên, lúc này ECU đang ở trạng thái locked. Do đó yêu cầu bị từ chối và ECU sẽ trả về tin negative response như dưới.

Graphical user interface, text, application

Description automatically generated

Hình : Kết quả yêu cầu dịch vụ $2EH khi ECU đang ở trạng thái Locked

* Thực hiện servive $27 nhưng sai mã Key sau đó yêu cầu đọc ghi dữ liệu xuống ECU bằng Joystick tại Tester.

Text

Description automatically generated

Hình : Kết quả trường hợp so sánh Key không trùng khớp khi yêu cầu Unlock

Ở trường hợp này, ta thực hiện service $27. Đầu tiên quá trình yêu cầu Seed của Tester được đáp ứng, do đó nhận được gói tin positive response. Tuy nhiên, mã Key mà Tester gửi cho ECU không khớp cho nên ECU sẽ trả về gói tin negative response và tạm thời khóa 10 giây không nhận bất kỳ truy cập CAN nào. Sau đó, ta tiếp tục thử nghiệm thực hiện service $2E. Kết quả là ECU đang locked và trả về gói tin negative response như trên.

* Thực hiện servive $27 và đúng mã Key sau đó yêu cầu đọc ghi dữ liệu xuống ECU bằng Joystick tại Tester và ADC.

Text, letter

Description automatically generated

Hình : Kết quả yêu cầu các dịch vụ khác thành công khi ECU ở trạng thái Unlocked

Trường hợp cuối cùng, ta thực hiện Service $27 với đúng mã Key giúp ECU đi đến trạng thái Unlocked. Lúc này ta thực hiện các Service $2E (ghi dữ liệu đến ECU qua joystick) và Service $22 (đọc giữ liệu từ ADC) và kết quả là các dịch vụ này được thực hiện thành công.