BÁO CÁO THỰC HÀNH BÀI Lab4_CAN_DiagnosticCommunication

Môn học: **Chuyên đề thiết kế hệ thống nhúng 1** - Mã lớp: **CE437.P11**
Giảng viên hướng dẫn thực hành: **Phạm Minh Quân**

Thông tin sinh viên	**Mã số sinh viên**	**Họ và tên**
	22521472	**Phạm Quốc Tiến**
	22521570	**Trịnh Thành Trung**
	22521564	**Nguyễn Đức Trung**
Link các tài liệu tham khảo *(nếu có)*		
Đánh giá của giảng viên: *+ Nhận xét* *+ Các lỗi trong chương trình* *+ Gợi ý*		

[Báo cáo chi tiết các thao tác, quy trình sinh viên đã thực hiện trong quá trình làm bài thực hành. Chụp lại hình ảnh màn hình hoặc hình ảnh kết quả chạy trên sản phẩm. Mô tả và giải thích chương trình tương ứng để cho ra kết quả như hình ảnh đã trình bày.]

Mục lục

1) Khai báo biến

Code

Trong bài lab này nhóm sẽ coi sensor node như là tester và actuator node là ECU

Các khai báo trong code tester

```
#define CAN_Request_ID 0x712
#define CAN_Response_ID 0x7A2

#define ReadDataByLocalIdentifier_Request_SID 0x22
#define ReadDataByLocalIdentifier_Response_SID 0x62

#define WriteDataByLocalIdentifier_Request_SID 0x2E
#define WriteDataByLocalIdentifier_Response_SID 0x6E

#define SecurityAccess_Request_SID 0x27
#define SecurityAccess_Response_SID 0x67
#define Security_SEED_level 0x01
#define Security_KEY_level 0x02

#define RecordDataIdentifier_High_Byte 0x01
#define RecordDataIdentifier_Low_Byte 0x23

#define Identifier_Negative_Response 0x7F
#define Invalid_length_or_response_format 0x13
#define Invalid_minimum_request_length 0x13
#define DID_not_support 0x31
#define Invalid_Keys 0x35

#define Countinue_State 0x00
#define Wait_State 0x01

#define SF 0b0000
#define FF 0b0001
#define CF 0b0010
#define FC 0b0011
```

```
CAN_RxHeaderTypeDef RxHeader;
uint8_t RxData[8];
CAN_TxHeaderTypeDef TxHeader = {
          .StdId = CAN_Request_ID,
          .IDE = CAN_ID_STD,
          .RTR = CAN_RTR_DATA,
          .DLC = 8
  };
uint8_t TxData[8];
uint32_t TxMailbox;
CAN_FilterTypeDef Filter = {
        .FilterIdHigh = 0x0000,
        .FilterIdLow = 0x0000,
        .FilterMaskIdHigh = 0x0000,
        .FilterMaskIdLow = 0x0000,
        .FilterBank = 0,
        .FilterMode = CAN_FILTERMODE_IDMASK,
        .FilterScale = CAN_FILTERSCALE_32BIT,
        .FilterFIFOAssignment = CAN_FILTER_FIFO1,
        .FilterActivation = CAN_FILTER_ENABLE
};
uint8_t SEED[4];
uint8_t KEY[16];
uint8_t SN = 0x0;
int MAX_KEY = 16;
int NEXT_KEY = 0;
int KEY_REMAIN = 0;
int Access_flag = 0;
```

Các khai báo trong code ECU

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------
/* USER CODE BEGIN PD */
#define CAN_Request_ID 0x712
#define CAN_Response_ID 0x7A2

#define ReadDataByLocalIdentifier_Request_SID 0x22
#define ReadDataByLocalIdentifier_Response_SID 0x62

#define WriteDataByLocalIdentifier_Request_SID 0x2E
#define WriteDataByLocalIdentifier_Response_SID 0x6E
#define SecurityAccess_Request_SID 0x27
#define SecurityAccess_Response_SID 0x67
#define Security_SEED_level 0x01
#define Security_KEY_level 0x02

#define RecordDataIdentifier_High_Byte 0x01
#define RecordDataIdentifier_Low_Byte 0x23

#define Identifier_Negative_Response 0x7F
#define Invalid_length_or_response_format 0x13
#define Invalid_minimum_request_length 0x13
#define DID_not_support 0x31
#define Invalid_Keys 0x35

#define Countinue_State 0x00
#define Wait_State 0x01

#define Block_Size 8
#define Separation_time 250

#define SF 0b0000
#define FF 0b0001
#define CF 0b0010
#define FC 0b0011
```

```
CAN_RxHeaderTypeDef RxHeader;
uint8_t RxData[8];
CAN_TxHeaderTypeDef TxHeader = {
          .StdId = CAN_Response_ID,
          .IDE = CAN_ID_STD,
          .RTR = CAN_RTR_DATA,
          .DLC = 8
  };
uint8_t TxData[8];
uint32_t TxMailbox;
CAN_FilterTypeDef Filter = {
        .FilterIdHigh = 0x0000,
        .FilterIdLow = 0x0000,
        .FilterMaskIdHigh = 0x0000,
        .FilterMaskIdLow = 0x0000,
        .FilterBank = 0,
        .FilterMode = CAN_FILTERMODE_IDMASK,
        .FilterScale = CAN_FILTERSCALE_32BIT,
        .FilterFIFOAssignment = CAN_FILTER_FIFO1,
        .FilterActivation = CAN_FILTER_ENABLE
};
uint8_t SEED[4];
uint8_t KEY[16];
uint8_t KEY_RECV[16];
uint8_t SN = 0x0;
uint8_t CANID = 0x0;
int MAX_KEY = 16;
int NEXT_KEY = 0;
int KEY_REMAIN = 0;
int Access_flag = 1;
```

2) Chu trình hoạt động

Khi hoạt động tester sẽ gửi lần lượt các service $22, $2E và $27 mỗi 2s

```
  while (1)
  {

      Send_SF_Read();
      HAL_Delay(2000);
      Send_SF_Security();
      HAL_Delay(2000);
      Send_SF_Write();
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

3) $22 - Read Data by Identifier

Ở phần practive này nhóm sẽ tiến hành gửi SF chứa SID của service read và RecoedDataIdentifier 0x0123

```
void Send_SF_Read(void)
{
    TxData[0] = (SF << 4) | 0x3;
    TxData[1] = ReadDataByLocalIdentifier_Request_SID;
    TxData[2] = RecordDataIdentifier_High_Byte;
    TxData[3] = RecordDataIdentifier_Low_Byte;
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
}
```

Khi bên ECU nhận được tín hiệu thì chương trình sẽ nhảy vào ngắt CAN rx 1 để xử lý

```
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &RxHeader, RxData) == HAL_OK)
        {
            if ((RxHeader.StdId == CAN_Request_ID))
             {
                switch(RxData[0] >> 4)
                {
                case SF:
                    switch(RxData[1])
                    {
                    case ReadDataByLocalIdentifier_Request_SID:
                        Read_Service_Handler();
                        break;
                    case SecurityAccess_Request_SID:
                        Security_Handler();
                        break;
                    case WriteDataByLocalIdentifier_Request_SID:
                        Write_Service_Handler();
                        break;
                    }
                    break;
                case FF:
                    switch(RxData[2])
                    {
                    case SecurityAccess_Request_SID:
                        Security_Handler();
                        break;
                    }
                    break;
                case CF:
                    Security_CF_Handler();
                    break;
                case FC:
                    break;
                }
             }
        }
}
```

Gói tin mà nhóm vừa gửi thì sẽ có SID của read request nên chương trình sẽ gọi hàm Read_Service_Handler để xử lý. Tại hàm Read_Service_Handler nhóm sẽ kiểm tra xem liệu byte RecordDataIdentifier có match không, nếu có thì sẽ tiến hành gửi lại positive response, nếu không thì sẽ gửi negative response với mã lỗi 0x13

```c
void Send_SF_Error_Code(uint8_t SID, uint8_t error_code)
{
    TxData[0] = (SF << 4) | 0x2;
    TxData[1] = Identifier_Negative_Response;
    TxData[2] = SID;
    TxData[3] = error_code;
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
}

void Read_Service_Handler(void)
{
    if(RxData[2] == RecordDataIdentifier_High_Byte && RxData[3] == RecordDataIdentifier_Low_Byte)
    {
        TxData[0] = (SF << 4) | 0x4;
        TxData[1] = ReadDataByLocalIdentifier_Response_SID;
        TxData[2] = RxData[2];
        TxData[3] = RxData[3];
        TxData[4] = RxHeader.StdId;
        HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
    }
    else
    {
        Send_SF_Error_Code(ReadDataByLocalIdentifier_Request_SID, Invalid_length_or_response_format);
    }
}

```

Khi mà nhận được positive response thì test sẽ xuất uart CAN_ID của tester, còn nhận negative response thì sẽ thông báo mã lỗi từ service read

```c
                        case ReadDataByLocalIdentifier_Response_SID:
                            Read_Service_Handler();
                            break;
                        case SecurityAccess_Response_SID:
                            Security_Handler();
                            break;
                        case WriteDataByLocalIdentifier_Response_SID:
                            Write_Service_Handler();
                            break;
                        case Identifier_Negative_Response:
                            CAN_Error_Handler();
                            break;
```

```c
void Read_Service_Handler(void)
{
    if(RxData[2] == RecordDataIdentifier_High_Byte && RxData[3] == RecordDataIdentifier_Low_Byte)
    {
        char text[4];
        sprintf(text, "%u", RxData[4]);
        HAL_UART_Transmit(&huart1, (uint8_t*)text, 5, 100);
    }
}
```

```c
void CAN_Error_Handler(void)
{
    char text[34];
    sprintf(text, "Ma loi 0x%u\n\rtu service 0x%u", RxData[3], RxData[2]);
    HAL_UART_Transmit(&huart1, (uint8_t*)text, 31, 100);
}
```

4) $27 - Security access service và $2E- Write Data by Identifier service

Do trong bài practice 2 có liên quan tới việc sử dụng service security trong bài practice 3 nên nhóm sẽ nhom hai bài lại thành một.

Để có thể sử dụng service Write thì nhóm sẽ tiến hành thực hiện service Security access trước để có quyền truy cập.

Đầu tiên nhóm sẽ tiến hành gửi SF request SEED từ tester

```c
void Send_SF_Security(void)
{
    TxData[0] = (SF << 4) | 0x2;
    TxData[1] = SecurityAccess_Request_SID;
    TxData[2] = Security_SEED_level;
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
}
```

Khi ECU nhận được thì sẽ tiến hành kiểm tra frame. Nếu byte thứ 3 chứa Security_SEED_level thì sẽ tiến hành generate SEED và KEY, sau đó gửi lại SEED cho tester.

```c
                    case SecurityAccess_Request_SID:
                        Security_Handler();
                        break;
```

```c
void Security_Handler(void)
{
    if(RxData[3] == Security_SEED_level)
    {
        srand(time(NULL));
        SEED[0] = (uint8_t)(rand() & 0xFF);
        SEED[1] = (uint8_t)(rand() & 0xFF);
        SEED[2] = (uint8_t)(rand() & 0xFF);
        SEED[3] = (uint8_t)(rand() & 0xFF);
        KEY[0] = SEED[0] ^ SEED[1];
        KEY[1] = SEED[1] + SEED[2];
        KEY[2] = SEED[2] ^ SEED[3];
        KEY[3] = SEED[3] + SEED[0];
        KEY[4] = SEED[0] | SEED[1];
        KEY[5] = SEED[1] + SEED[2];
        KEY[6] = SEED[2] | SEED[3];
        KEY[7] = SEED[3] + SEED[0];
        KEY[8] = SEED[0] & SEED[1];
        KEY[9] = SEED[1] + SEED[2];
        KEY[10] = SEED[2] & SEED[3];
        KEY[11] = SEED[3] + SEED[0];
        KEY[12] = SEED[0] - SEED[1];
        KEY[13] = SEED[1] + SEED[2];
        KEY[14] = SEED[2] - SEED[3];
        KEY[15] = SEED[3] + SEED[0];
        KEY_REMAIN = MAX_KEY;
        Send_SF_Security();
    }
```

```c
void Send_SF_Security(void)
{
    TxData[0] = (SF << 4) | 0x6;
    TxData[1] = SecurityAccess_Response_SID;
    TxData[2] = Security_SEED_level;
    TxData[3] = SEED[0];
    TxData[4] = SEED[1];
    TxData[5] = SEED[2];
    TxData[6] = SEED[3];
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
}
```

Khi tester nhận được SEED thì sẽ tự generate ra KEY, set quyền access là false và gửi KEY lại cho ECU để ECU check. Do có tận 15 KEY nên nhóm sẽ gửi multi frame.

```c
void Security_Handler(void)
{
    if(RxData[2] == Security_SEED_level)
    {
        Access_flag = 0;
        SEED[0] = RxData[3];
        SEED[1] = RxData[4];
        SEED[2] = RxData[5];
        SEED[3] = RxData[6];
        KEY[0] = SEED[0] ^ SEED[1];
        KEY[1] = SEED[1] + SEED[2];
        KEY[2] = SEED[2] ^ SEED[3];
        KEY[3] = SEED[3] + SEED[0];
        KEY[4] = SEED[0] | SEED[1];
        KEY[5] = SEED[1] + SEED[2];
        KEY[6] = SEED[2] | SEED[3];
        KEY[7] = SEED[3] + SEED[0];
        KEY[8] = SEED[0] & SEED[1];
        KEY[9] = SEED[1] + SEED[2];
        KEY[10] = SEED[2] & SEED[3];
        KEY[11] = SEED[3] + SEED[0];
        KEY[12] = SEED[0] - SEED[1];
        KEY[13] = SEED[1] + SEED[2];
        KEY[14] = SEED[2] - SEED[3];
        KEY[15] = SEED[3] + SEED[0];
    }
```

```c
void Send_FF_Security(void)
{
    TxData[0] = (FF << 4) | 0x0;
    TxData[1] = 0x06;
    TxData[2] = SecurityAccess_Request_SID;
    TxData[3] = Security_SEED_level;
    TxData[4] = KEY[0];
    TxData[5] = KEY[1];
    TxData[6] = KEY[2];
    TxData[7] = KEY[3];
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
    NEXT_KEY = 4;
    KEY_REMAIN = MAX_KEY - NEXT_KEY;

}
```

Sau đó bên ECU sẽ kiểm tra 4 KEY đầu và gửi FC

```
249     else if(RxData[3] == Security_KEY_level)
250     {
251         Access_flag = 1;
252         if(KEY[0] != RxData[4]) Access_flag = 0;
253         if(KEY[1] != RxData[5]) Access_flag = 0;
254         if(KEY[2] != RxData[6]) Access_flag = 0;
255         if(KEY[3] != RxData[7]) Access_flag = 0;
256         NEXT_KEY = 4;
257         KEY_REMAIN = MAX_KEY - NEXT_KEY;
258         Send_FC_Security();
259     }
```

```
177 void Send_FC_Security(void)
178 {
179     TxData[0] = (FC << 4) | Countinue_State;
180     TxData[1] = Block_Size;
181     TxData[2] = Separation_time;
182     HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
183 }
```

Khi bên tester nhận được FC thì sẽ tiến hành set DLC header Tx bằng Block Size (8) và set timer để gửi CF mỗi Separation time (25ms).

```
268                 case FC:
269                     if((RxData[0] & 0b00001111) == Countinue_State)
270                     {
271                         TxHeader.DLC = (int)RxData[1];
272                         __HAL_TIM_SET_AUTORELOAD(&htim2, (int)RxData[2]);
273                         HAL_TIM_Base_Start_IT(&htim2);
274                     }
275                     break;
```

```
227 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
228 {
229     if(KEY_REMAIN > 0)
230     {
231         Send_CF_Security();
232     }
```

```
175 void Send_CF_Security(void)
176 {
177     TxData[0] = (CF << 4) | SN;
178     for(int i = 1; i < KEY_REMAIN || i < 8; i++)
179     {
180         TxData[i] = KEY[NEXT_KEY];
181         NEXT_KEY++;
182     }
183     HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
184     SN += 1;
185     KEY_REMAIN = MAX_KEY - NEXT_KEY;
186 }
```

Bên ECU sẽ check KEY mỗi khi nhận được gói CF.

```
297                     case CF:
298                         Security_CF_Handler();
299                         break;
```

```
200 void Security_CF_Handler(void)
201 {
202     for(int i = 1; i < KEY_REMAIN || i < 8; i++)
203     {
204         if(KEY[NEXT_KEY] != RxData[i]) Access_flag = 0;
205         NEXT_KEY++;
206     }
207     KEY_REMAIN = MAX_KEY - NEXT_KEY;
```

Và khi KEY đã gửi hết thì tester sẽ ngưng gửi CF

```
232     }
233     else
234     {
235         HAL_TIM_Base_Stop_IT(&htim2);
236     }
237 }
```

Còn ECU sẽ gửi accept hoặc lỗi invalid key cho tester. Đồng thời khi mà Accept thì ECU sẽ bật LED G và hẹn timer sau 5s sẽ xóa quyền access

```
208     if(KEY_REMAIN == 0)
209     {
210         if(Access_flag)
211         {
212             Security_Accept();
213         }
214         else
215         {
216             Send_SF_Error_Code(SecurityAccess_Request_SID ,Invalid_Keys);
217         }
218     }
219 }
```

```
193 void Security_Accept(void)
194 {
195     HAL_GPIO_WritePin(GPIOB, LED_G_Pin, 1);
196     HAL_TIM_Base_Start_IT(&htim2);
197     Send_SF_Security_Accept();
198 }
```

```
185 void Send_SF_Security_Accept(void)
186 {
187     TxData[0] = (SF << 4) | 0x2;
188     TxData[1] = SecurityAccess_Response_SID;
189     TxData[2] = Security_KEY_level;
190     HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
191 }
```

```
261 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
262 {
263     HAL_GPIO_WritePin(GPIOB, LED_G_Pin, 0);
264     HAL_TIM_Base_Stop_IT(&htim2);
265     Access_flag = 0;
266 }
```

Bên tester khi nhận được frame accept hoặc frame lỗi thì sẽ tiến hành xuất ra uart

```
213     else if(RxData[2] == Security_KEY_level)
214     {
215         Access_flag = 1;
216         HAL_UART_Transmit(&huart1, (uint8_t*)"Key accept", 11, 100);
217     }
```

Khi đã có quyền access thì nhóm tiến hành gửi SF request write

```
void Send_SF_Write(void)
{
    if(Access_flag)
    {
        TxData[0] = (SF << 4) | 0x3;
        TxData[1] = WriteDataByLocalIdentifier_Request_SID;
        TxData[2] = RecordDataIdentifier_High_Byte;
        TxData[3] = RecordDataIdentifier_Low_Byte;
        HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
    }
}
```

Khi bên ECU nhận được thì sẽ tiến hành write CAN_ID của tester (phần này nhóm chỉ làm tượng trưng) và gửi positive response

```
286                     case WriteDataByLocalIdentifier_Request_SID:
287                         Write_Service_Handler();
288                         break;
```

```
156 void Write_Service_Handler(void)
157 {
158     if(Access_flag)
159     {
160         CANID = RxHeader.StdId;
161         Send_SF_Write();
162     }
163 }
```

```
149 void Send_SF_Write(void)
150 {
151     TxData[0] = (SF << 4) | 0x1;
152     TxData[1] = WriteDataByLocalIdentifier_Response_SID;
153     HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
154 }
```

Khi nhận được positive response thì nhóm sẽ xuất ra uart

```
256                     case WriteDataByLocalIdentifier_Response_SID:
257                         Write_Service_Handler();
258                         break;
```

```
145 void Write_Service_Handler(void)
146 {
147     HAL_UART_Transmit(&huart1, (uint8_t*)"Write ok", 9, 100);
148 }
```