BÁO CÁO ĐỒ ÁN

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: 20520349 Họ và tên: Lê Hữu Vinh Mã số sinh viên: 20521595 Họ và tên: Phạm Văn Mạnh Mã số sinh viên: 20521594 Họ và tên: Võ Minh Mẫn
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. Sinh viên xuất ra file .pdf và đặt tên theo cấu trúc: MSSV_HoTen_Labx_Report.pdf (Trong đó: MSSV là mã số sinh viên, HoTen là họ và tên, x trong Labx là chỉ số của bài thực hành tương ứng]

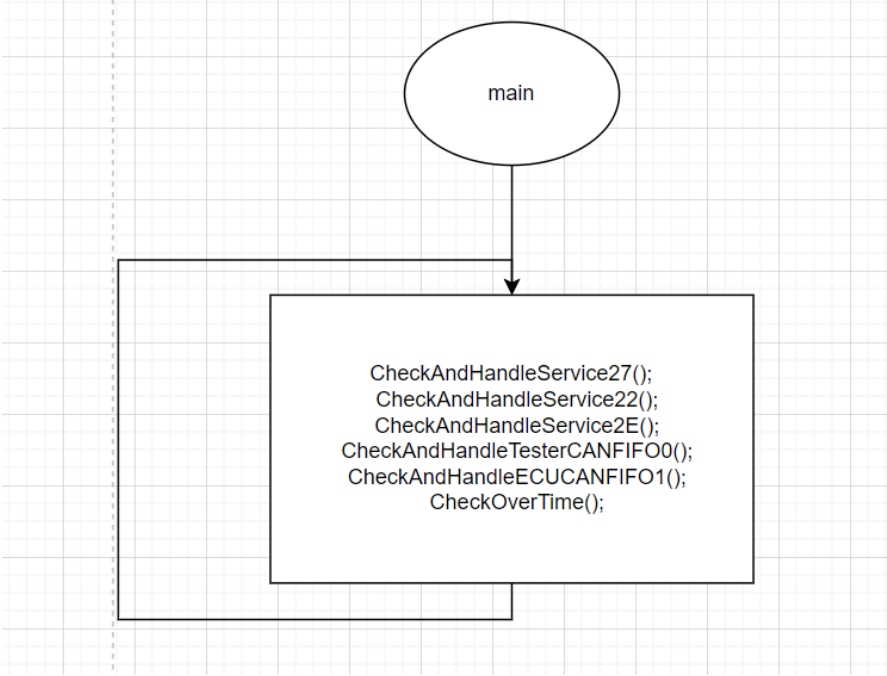
Mục lục

Đề bài

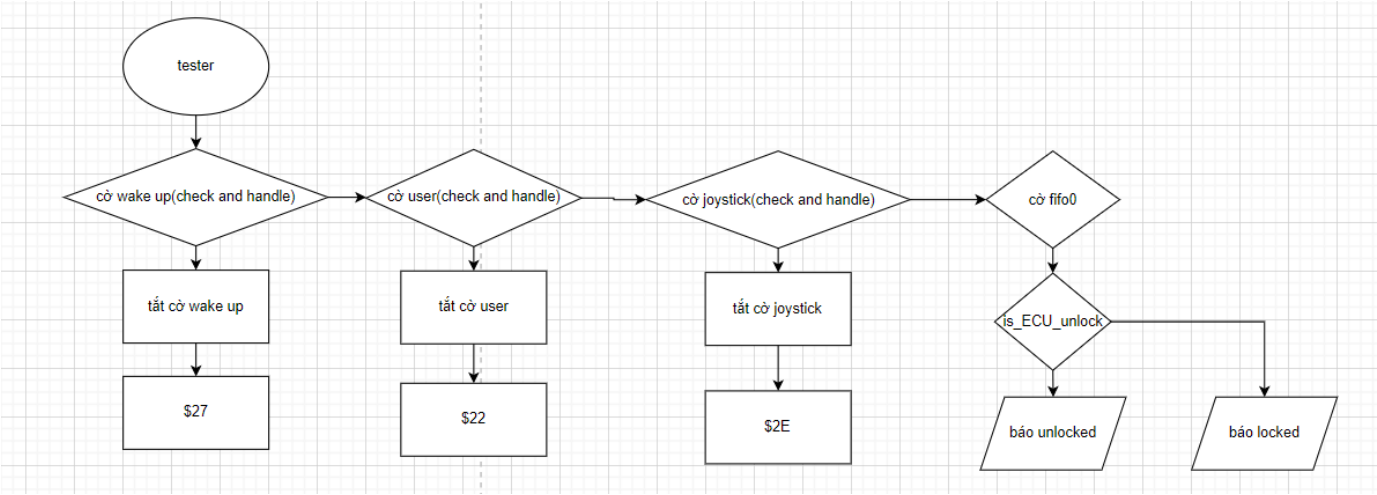
1. **Thực hiện Service $27H (Security Access) của chuẩn giao tiếp CAN**

KẾT QUẢ

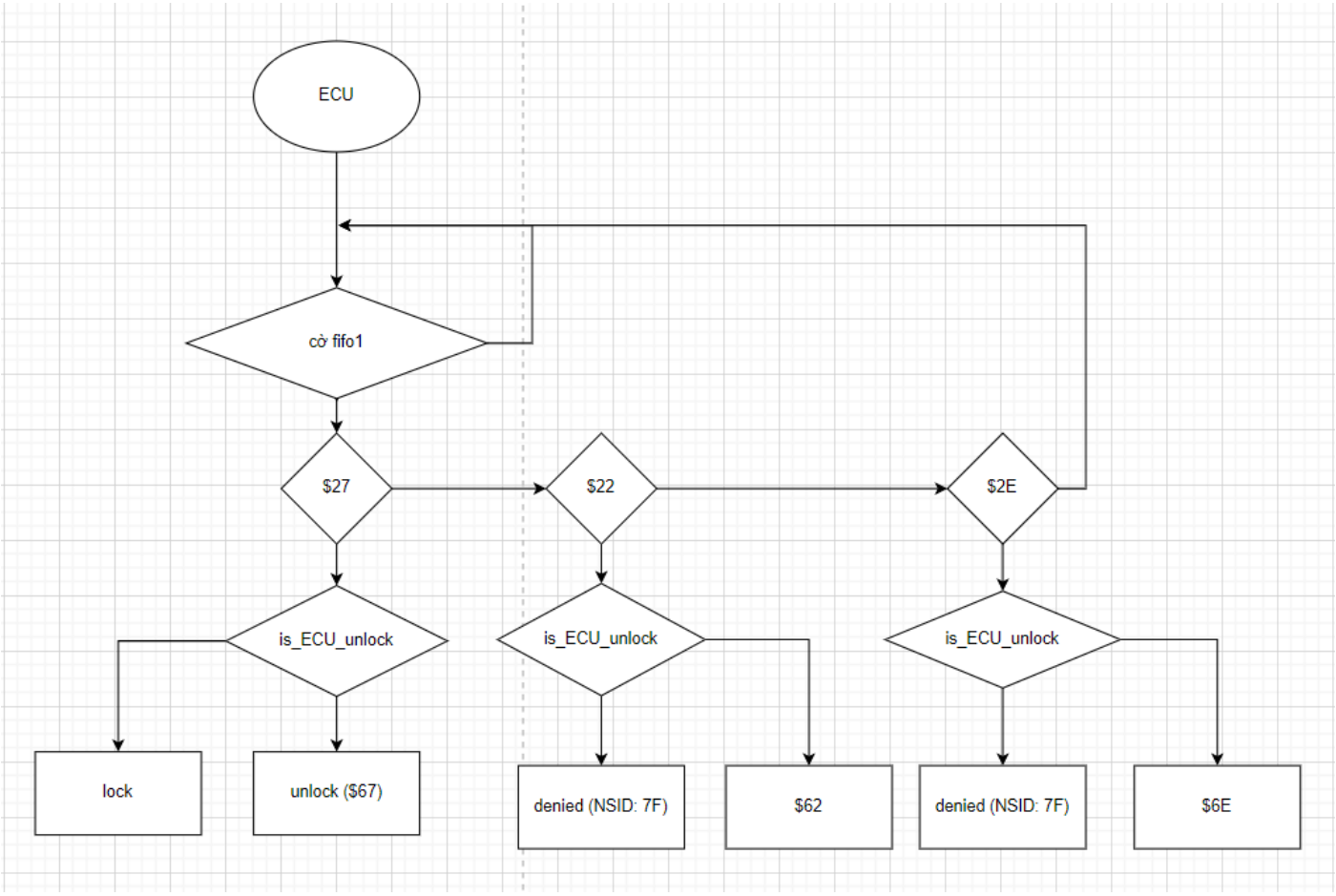
1. Lưu đồ thuật toán:



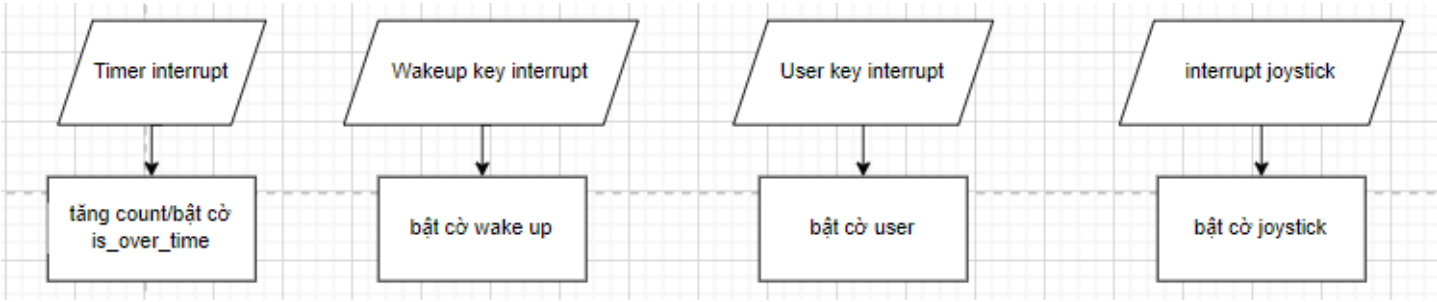
Hình 1: Lưu đồ hàm Main



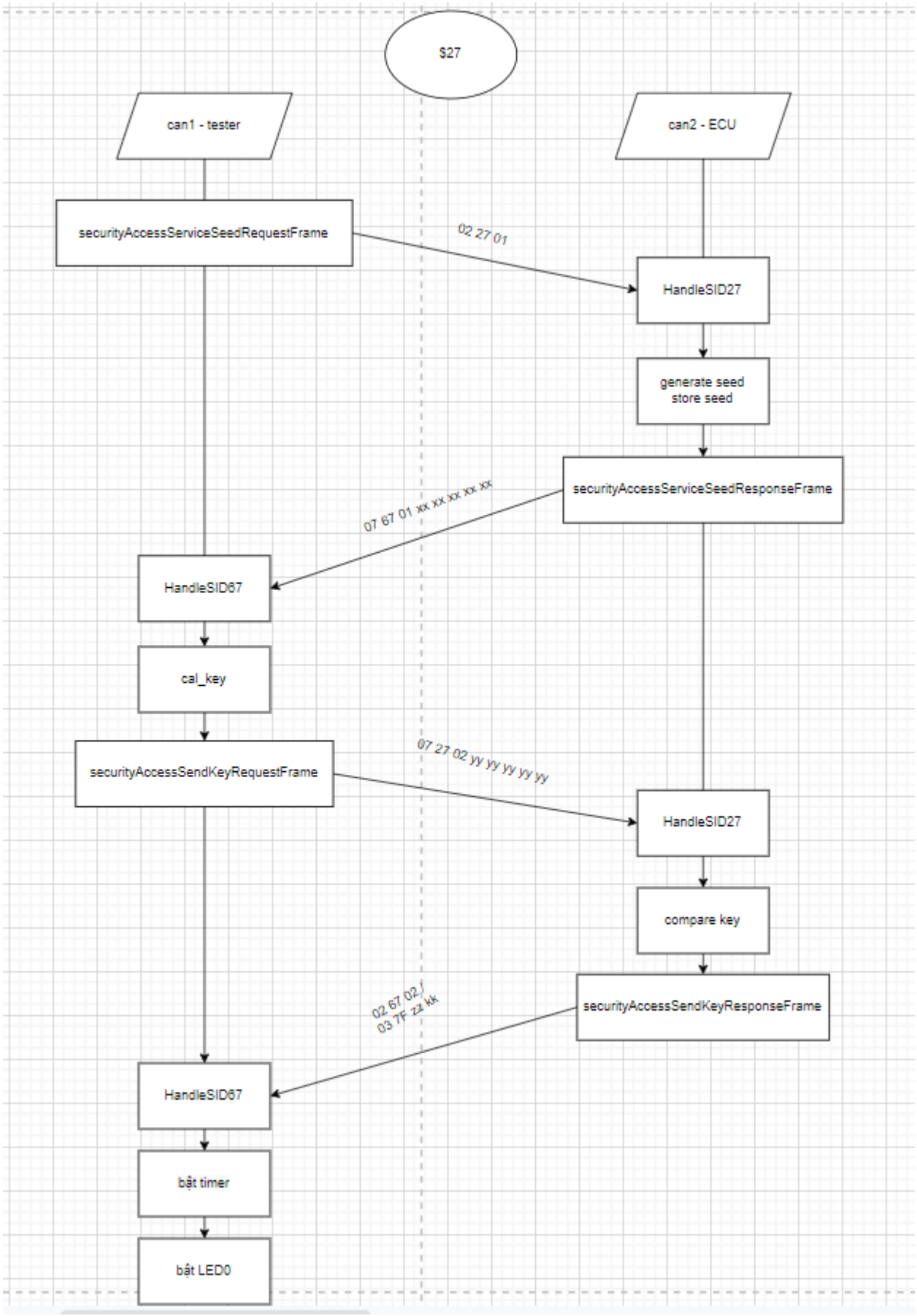
Hình 2: Lưu đồ của tester xử lí các cờ ngắt



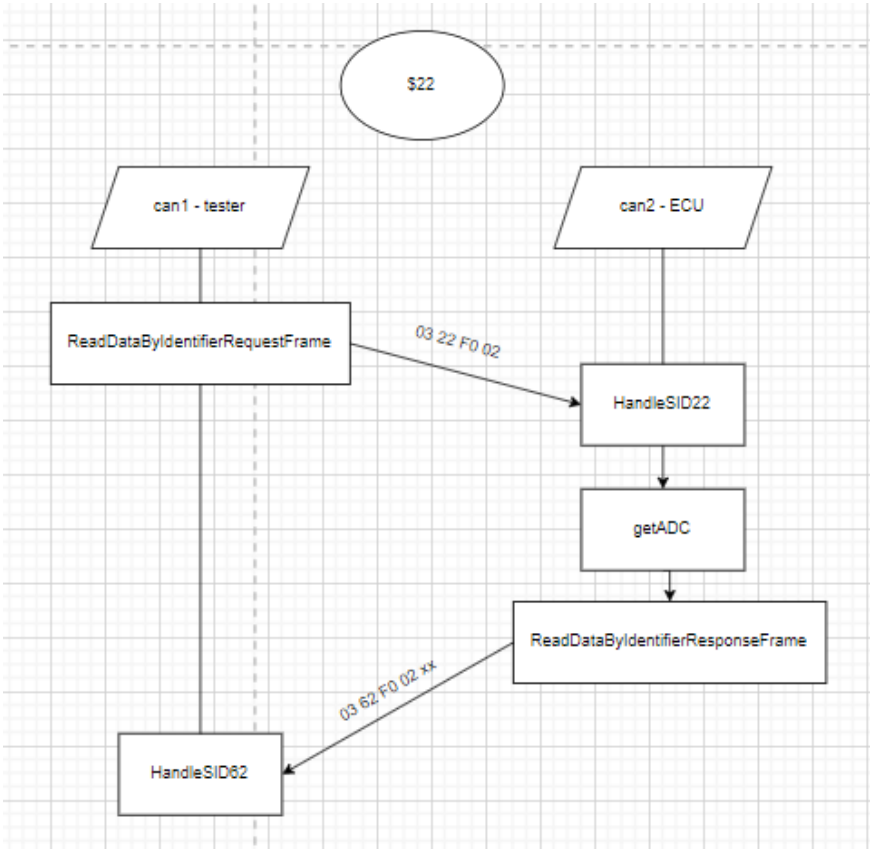
Hình 3: Lưu đồ của ECU xử lí các cờ ngắt



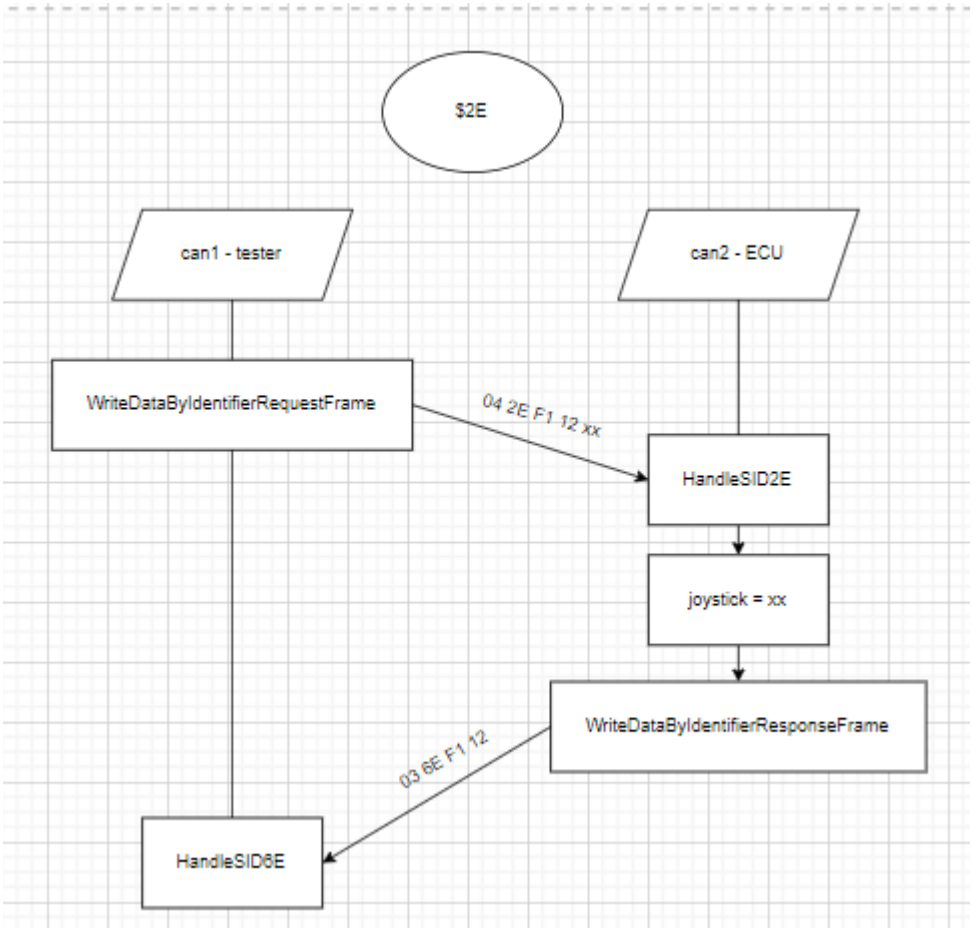
Hình 4: Các cờ ngắt



Hình 5: Service 27 flow



Hình 6: Service 22 flow



Hình 7: Service 2E flow

2. Demo: Security Access:

Link GoogleDrive: https://drive.google.com/file/d/1LHxhgssol0-vaMhTelrsCl7beVDFm_wP/view

Demo này sẽ có 5 nút nhấn có thể thực hiện:

+ joystick trái (Left): khi nhấn vào nút này chương trình sẽ thực hiện $2EH (Write By Identifier) như ở Lab4, gán giá trị 0xAA cho biến joystick của ECU. Nhưng khi ECU chưa được unlock bởi $27H (Security Access) thì sẽ không gán được.

+ joystick phải (Right): tương tự joystick trái nhưng giá trị được gán sẽ là 0x00 thay vì 0xAA.

+ joystick giữa (Mid): tương tự joystick trái nhưng giá trị được gán sẽ là 0xFF thay vì 0xAA.

+ Wake up: khi nhấn vào nút này chương trình sẽ thực hiện $27H (Security Access) nếu ECU chưa được unlock hoặc lock lại khi ECU đang unlock.

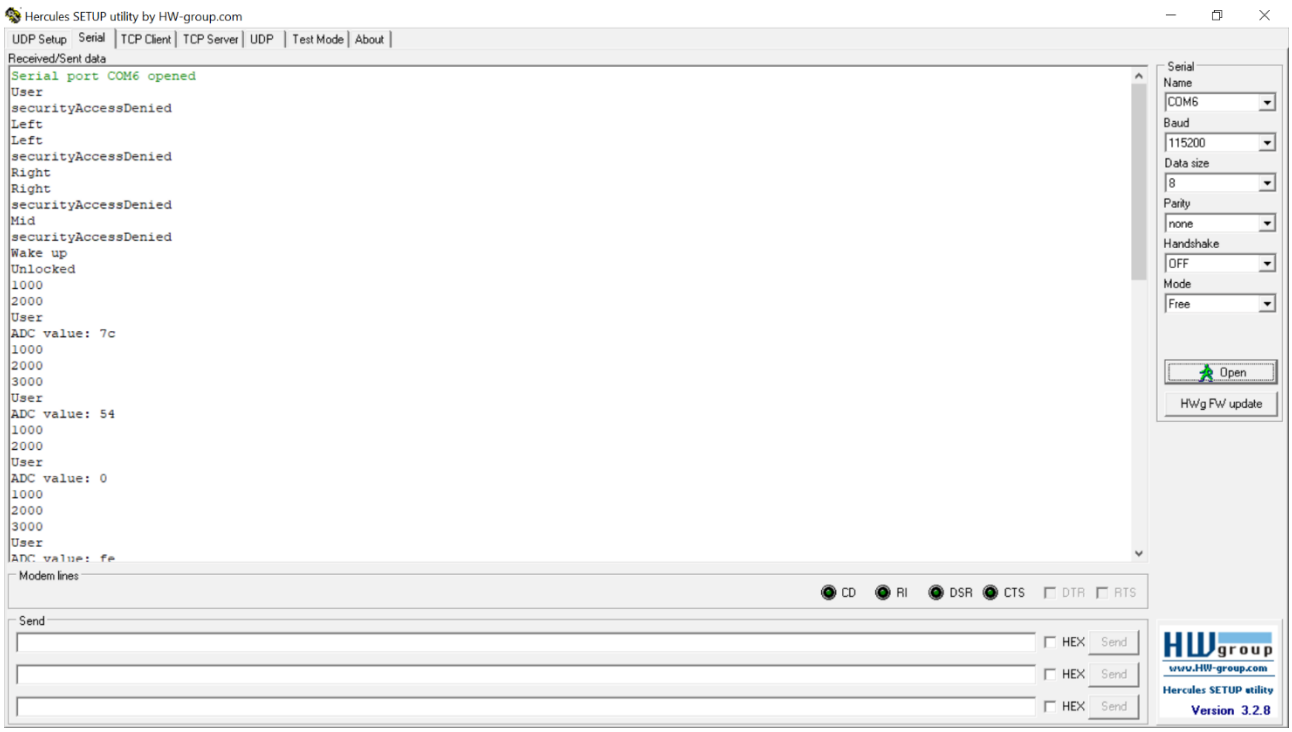
+ User: khi nhấn vào nút này chương trình sẽ thực hiện $22H(Read By Identifier) như ở Lab4, đọc giá trị từ ADC và hiển thị lên màn hình của tester.

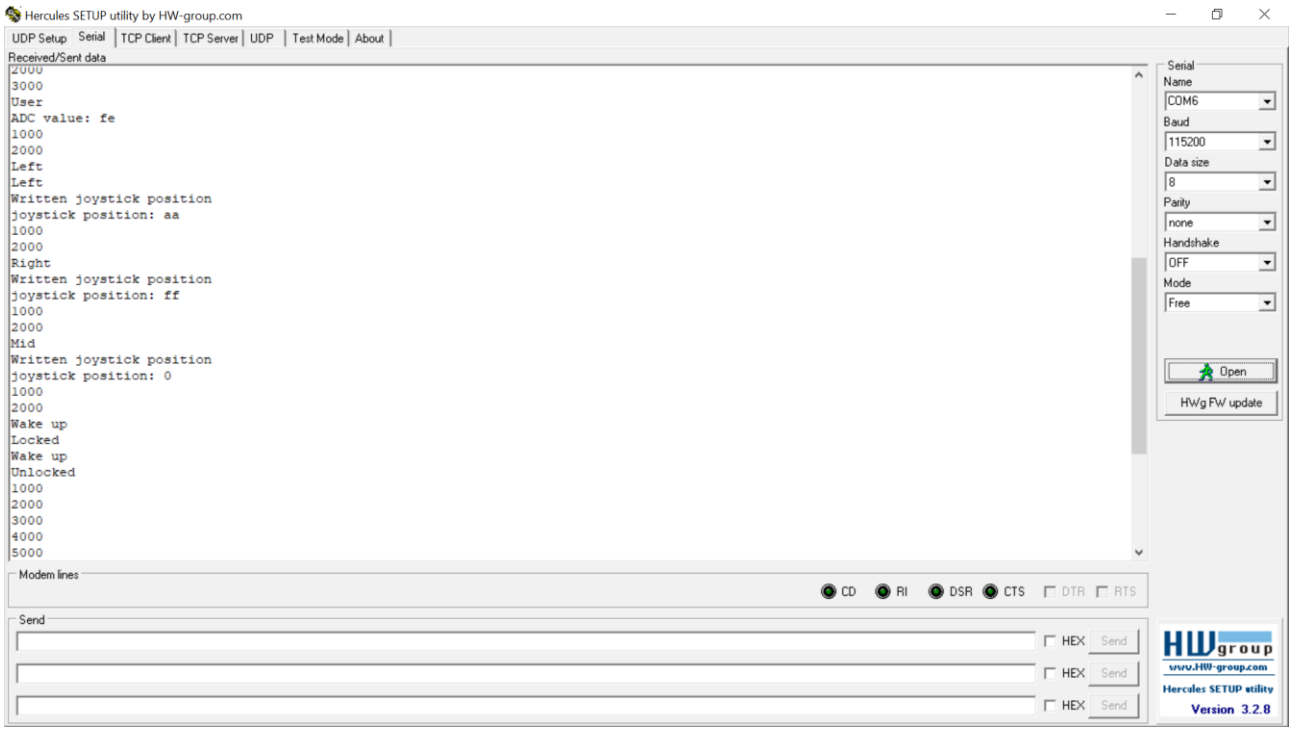
Màn hình Hercules sẽ đóng vai trò là màn hình mà tester dung để đọc.

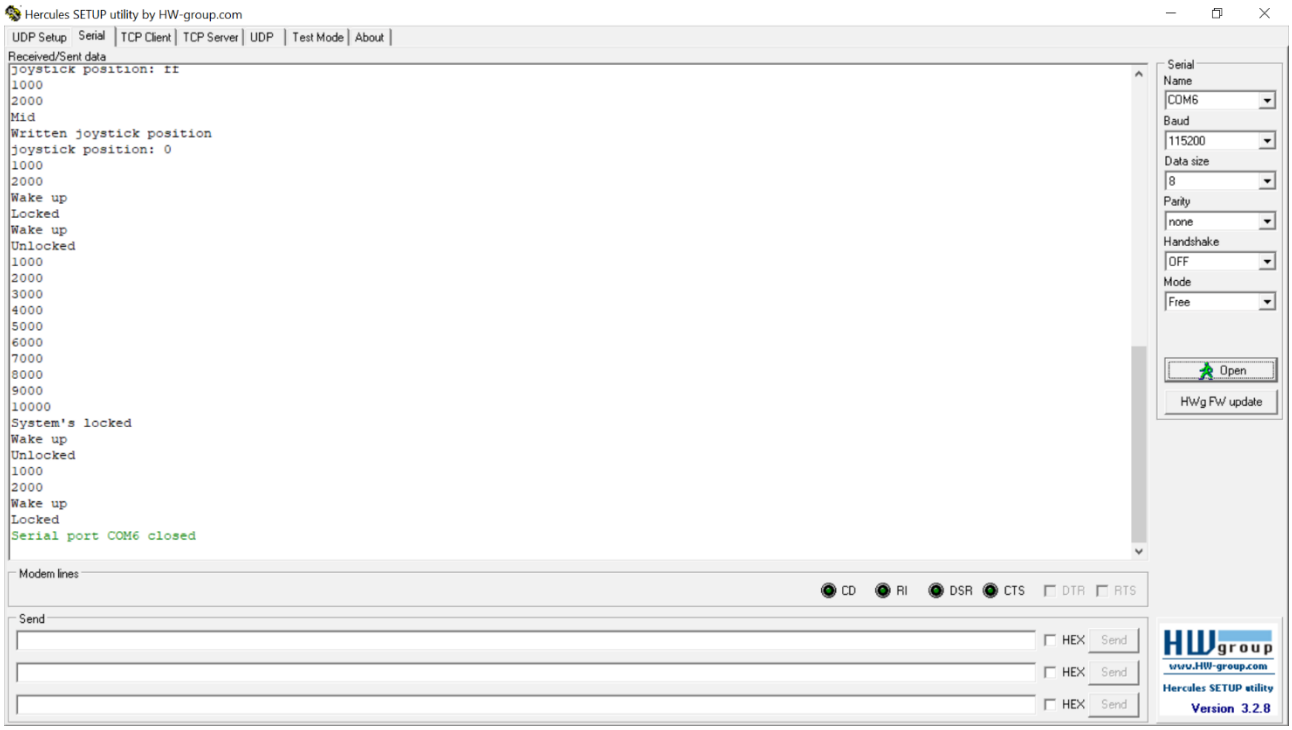
Khi nhấn 1 nút bất kì, màn hình sẽ hiện ra vị trí mà mình đã nhất ví dụ như User, Left, Right, Mid, Wake up.

Khi ECU được unlock sẽ có thời gian hiển thị để quan sát móc thời gian để khi tăng đến 10s thì ECU sẽ được lock trở lại(10s kể từ khi không có lệnh nào từ tester được gửi đến ECU). Trên thực tế thời gian này sẽ không được hiển thị lên cửa sổ Hercules vì thời gian này được điều khiển bởi ECU mà màn hình chỉ hiển thị những gì mà tester điều khiển nên móc thời gian chỉ để quan sát.

Cửa sổ Hercules của hiển thị giá trị của biến joystick (biến này dung để lưu vị trí của joystick khi nhấn vào joystick của tester để thực hiện $2EH) để tiện quan sát, trên thực tế giá trị của biến này sẽ không được hiển thị với lí do tương tự giá trị móc thời gian vì được điều khiển bởi ECU mà màn hình Hercules có mục đích hiển thị những gì mà tester điều khiển.

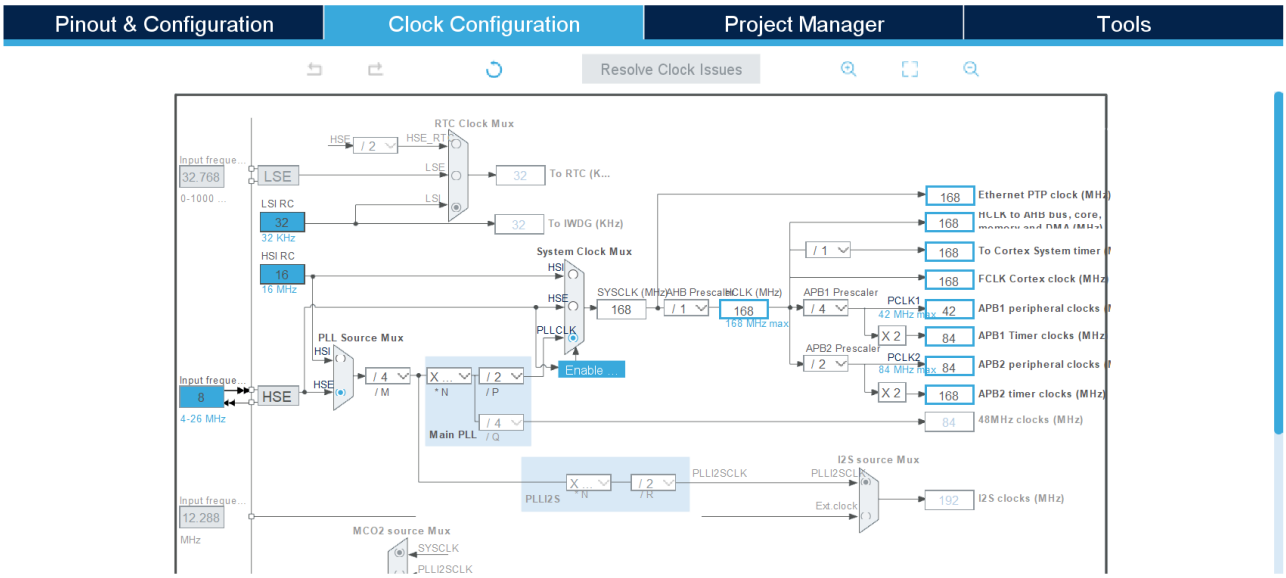






Khi chương trình chưa được unlock thì dù nhấn vào nút nào cũng sẽ hiển mã lỗi là "securityAccessDinied", trừ nút Wake up vì khi nhấn nút này chương trình sẽ thực hiện $27H để unlock ECU. Khi ECU được unlock biến timer của ECU sẽ bắt đầu đếm lên 10s, nếu trong vòng 10s rãnh không có lệnh từ tester ECU sẽ tự động lock trở lại (lúc đó màn hình tester sẽ hiện dòng chữ "System's locked"), hoặc nếu muốn lock ECU lại mà không cần đợi 10s thì chỉ cần nhấn vào nút Wake up của tester 1 lần nữa (lúc đó màn hình tester sẽ hiện dòng chữ "Locked"). Nếu trong 10s ECU được unlock, tester gửi các yêu cầu như $22H hoặc $27H thì timer sẽ chạy lại từ đầu và bắt đầu đợi 10s.

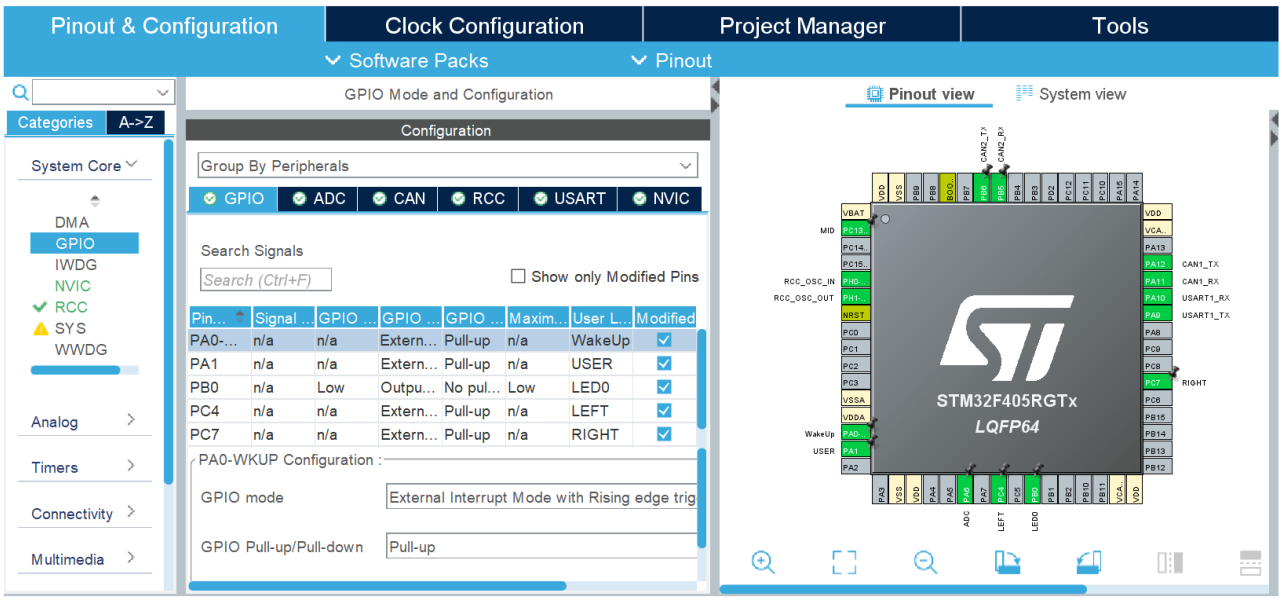
CLOCK CONFIGURATION



Tần số vào là 8MHz vì trên Kit có thạch anh ngoài 8MHz và chọn HSE (High Speed External). System Clock MUX chọn bộ PLLCLCK và tần số chọn max có thể là 168MHz.

PINOUT & CONFIGURATION

1. GPIO Mode and Configuration:

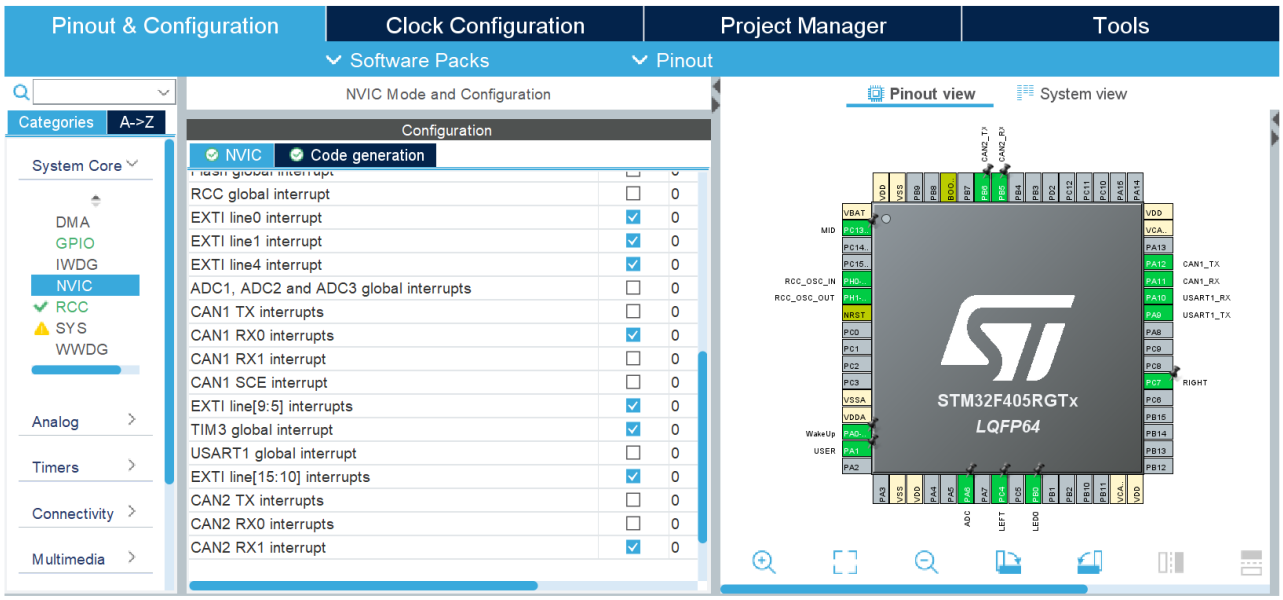


Chọn chế độ Output mặc định No pull-up and no pull-down.

Chọn xuất LED0 để báo hiệu ECU đang unlock(sáng) hay lock(tắt).

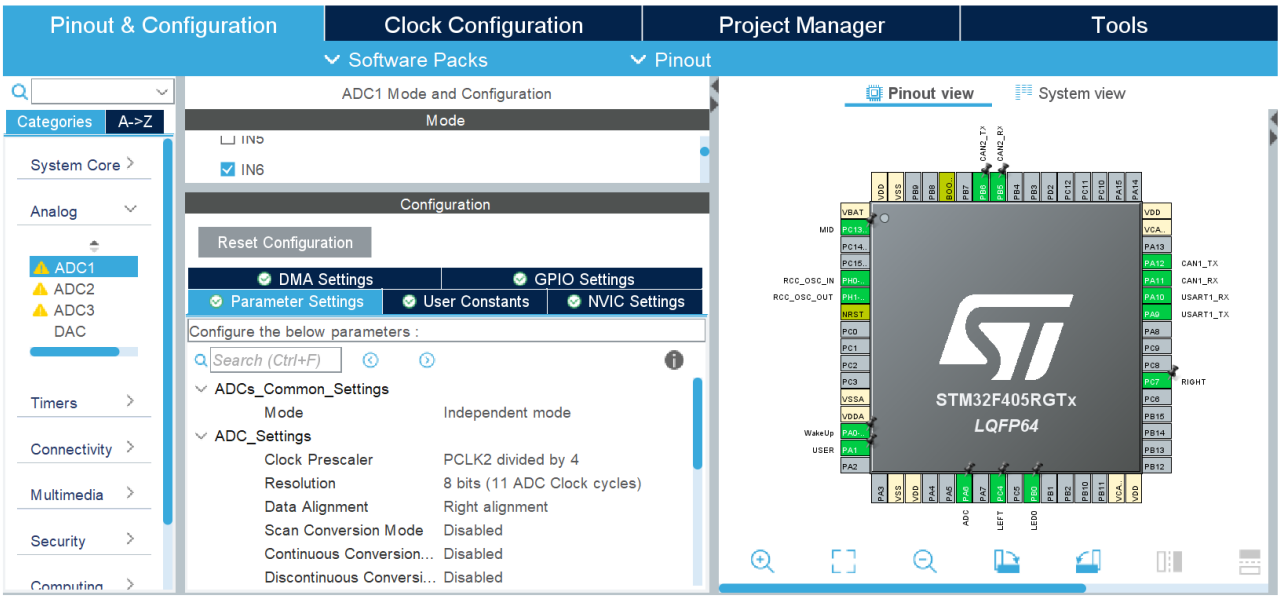
Chọn chế độ EXTI pull-up cho 5 chân ngắt gồm 3 chân của: LEFT, MID, RIGHT; và chân Wake Up và chân User để thực hiện ngắt mỗi khi nhận phím.

2. NVIC Mode and Configuration:



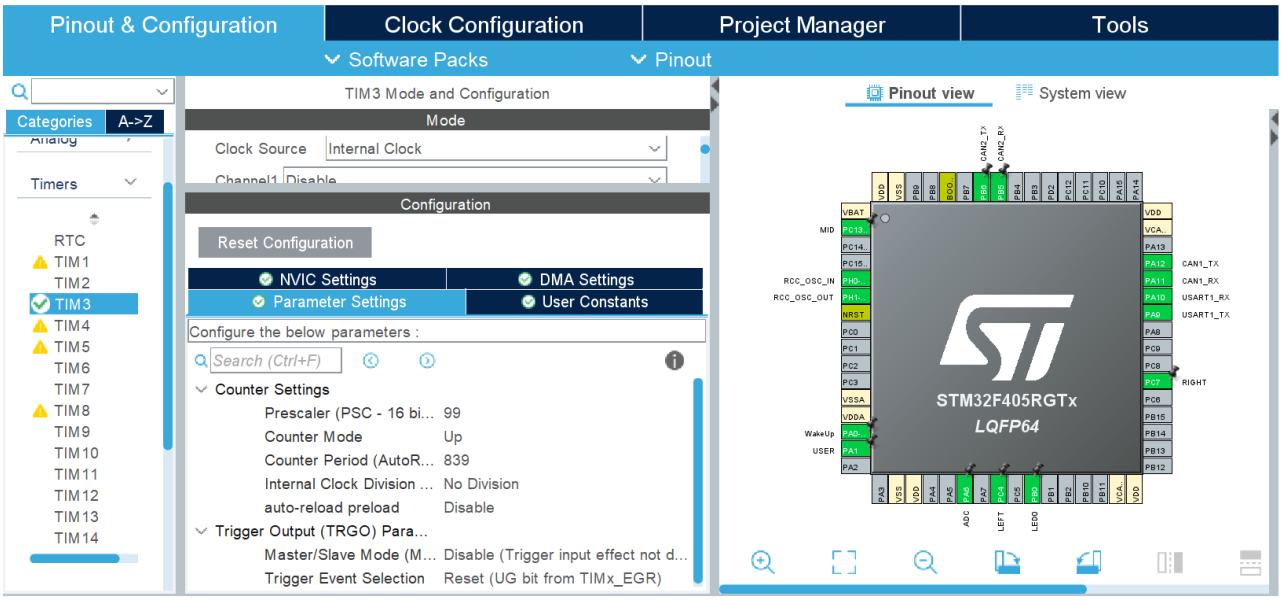
Cho phép các ngắt liên quan đến EXTI(tester control) và cho phép các hàm ngắt liên quan đến CAN1 (tester control), CAN2 (ECU control) và Timer(ECU control).

3. ADC CONFIGURATION:



Dùng ADC có độ phân giải 8bits vừa đủ cho 1byte dữ liệu truyền đi được điều khiển bởi ECU.

4. TIMER CONFIGURATION:



Timer sẽ được điều khiển bởi ECU để đếm đủ 10s nhằm lock ECU lại nếu không có lệnh trong vòng 10s.

Sử dụng Internal Clock làm clock source và áp dụng công thức:

$$F_{timer} = \frac{F_{hệ\ thống}}{(Prescaler+1)}$$

$$T_{event} = \frac{1}{F_{timer}} * (Counter\ Period + 1)$$

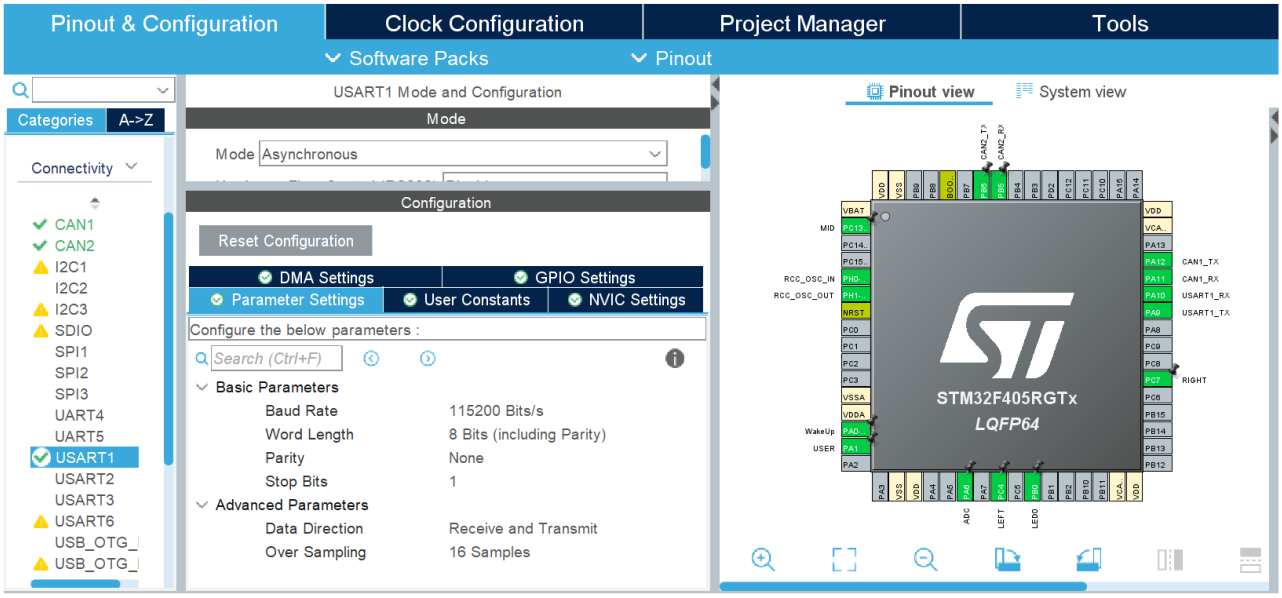
$$= \frac{(Prescaler + 1) * (Counter\ Period + 1)}{F_{hệ\ thống}}$$

TIM3 được kết nối với APB (theo datasheet), mà APB1 có clock là 84MHz (CLOCK CONFIGURATION) nên $F_{hệthống}$ sẽ là 84MHz.

Để có T_{event} là 0.001s thì sẽ chỉnh Prescaler là 99; Counter Period là 839 và Counter Mode là Up.

Dùng TIMER để tạo delay như yêu cầu của đề bài.

5. USART1 Mode and Configuration:

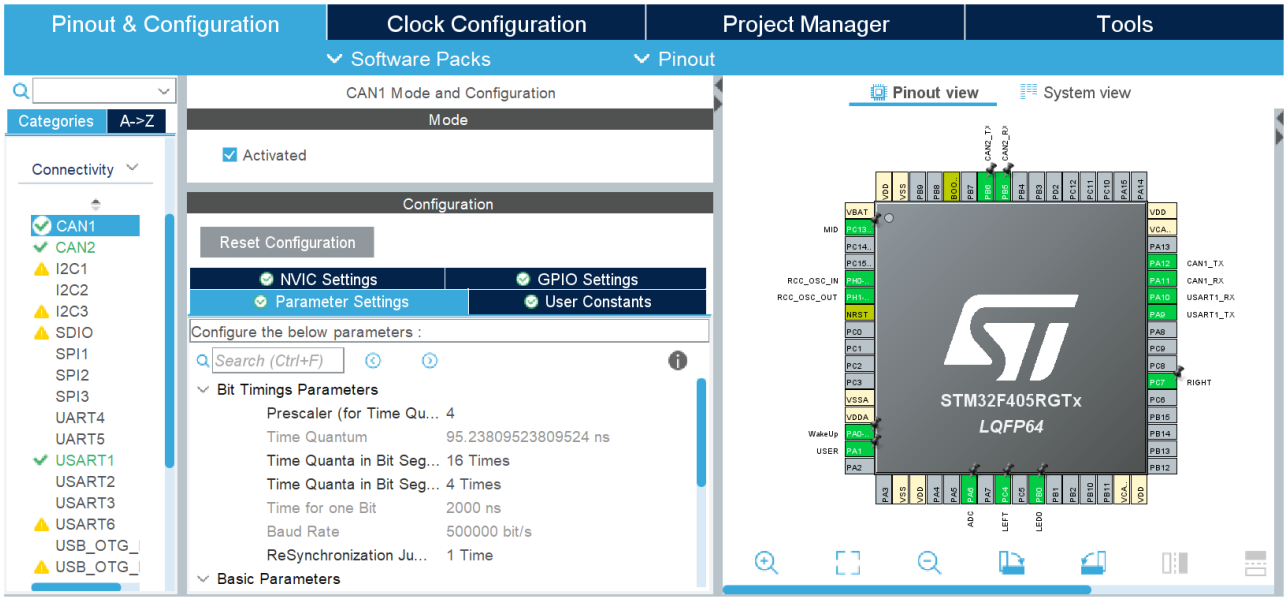


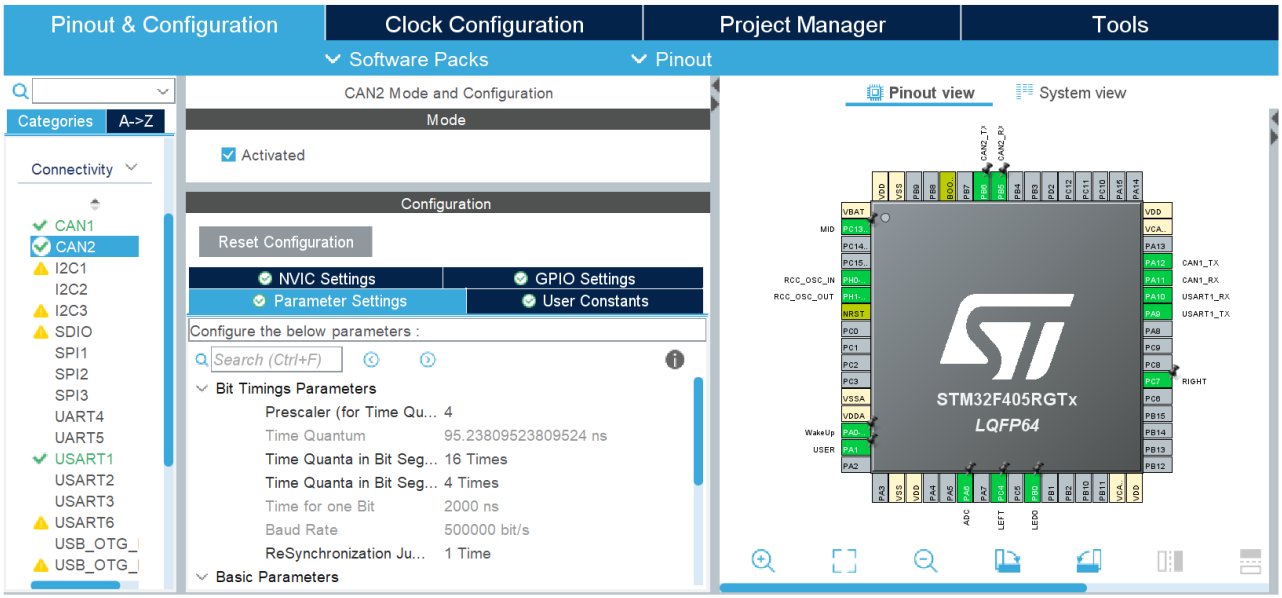
USART1 dùng để hiển thị những thông tin mà tester cần lên Hercules và được điều khiển bởi tester (bên cạnh đó sẽ hiển thị 1 số thông tin của ECU nhằm mục đích quan sát dù trên thực tế thì điều này là không đúng vì Màn hình của tester không xuất được thông tin của ECU một cách trực tiếp như vậy).

USART1 chọn chế độ Asynchronous và Baud Rate là 115200 Bits/s.

Dùng USART để theo dõi các mốc thời gian và dữ liệu nhận được từ CAN1 và CAN2 và các tín hiệu khác.

6. CAN Mode Configuration:





Trong bài này CAN1 sẽ đóng vai trò là tester và CAN2 sẽ đóng vai trò là ECU.

Điều chỉnh các thông số Prescaler, Time Quanta in Bit Segment 1 và Time Quanta in Bit Segment 2 sao cho phù hợp với yêu cầu của đề bài được trình bày ở bảng phía dưới.

Parameter	Symbol	Minimum value	Nominal value	Maximum value	Unit	Remarks
Bit timing	tBit	1992	2000	2008	ns	fHSCAN = 500kbps (± 0.4 %)
Tq quantity	NBT	10	16	20	Tq	
Sampling position	tSP	75	-	82	%	
Synchronisation jump width	SJW	2	2	3	Tq	
Sampling amount	NSP	-	1	-		

CODE

1. Include:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>

/* USER CODE END Includes */
```

Sử dụng thêm thư viện <*stdio.h*> để sử dụng hàm printf.

2. Define and Macro:

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define data_length 8
#define tester_ID   0x712
#define ECU_ID      0x7A2
#define true        1
#define false       0
#define zero        0
#define over_time   10000
#define LEFT 0xAA
#define MID 0x00
#define RIGHT 0xFF

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#ifdef __GNUC__
 /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
 set to 'Yes') calls __io_putchar() */
 #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
 #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/* USER CODE END PM */
```

data_length là 8 theo yêu cầu của đề bài.

tester_ID là 0x712 theo yêu cầu của đề bài đóng vai trò là ID của tester.

ECU_ID là 0x7A2 theo yêu cầu của đề bài đóng vai trò là ID của ECU.

true và *false* dung để tiện sử dụng các phép toán logic.

zero là giá trị 0.

over_time là thời gian để sau bao lâu mili giây thì hệ thống sẽ được lock trở lại khi không có lệnh nào cần thực hiện.

LEFT, MID, RIGHT là cờ chỉ vị trí của joystick với giá trị đưa cho bởi đề bài. Tương ứng của LEFT là 0xAA, MID là 0x00 và RIGHT là 0xFF.

3. Private variable:

```
/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

CAN_HandleTypeDef hcan1;
CAN_HandleTypeDef hcan2;

TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
CAN_TxHeaderTypeDef tester_TxHeader;
CAN_RxHeaderTypeDef tester_RxHeader;
uint8_t tester_TxData[data_length];
uint8_t tester_RxData[data_length];
uint32_t tester_TxMailbox;
CAN_TxHeaderTypeDef ECU_TxHeader;
CAN_RxHeaderTypeDef ECU_RxHeader;
uint8_t ECU_TxData[data_length];
uint8_t ECU_RxData[data_length];
uint32_t ECU_TxMailbox;

uint8_t ECU_store_seed[data_length];    // ECU control
uint8_t joystick;                       // ECU control
uint16_t count = zero;                  // ECU control
uint8_t is_ECU_unlock = false;          // ECU control
uint8_t is_run_timer = false;           // ECU control
uint8_t is_get_fifo1 = false;           // ECU control
uint8_t is_over_time = false;           // ECU control

uint8_t is_get_fifo0 = false;           // tester control
uint8_t flag_wake_up_key = false;       // tester control
uint8_t flag_user_key = false;          // tester control
uint8_t flag_left_key = false;          // tester control
uint8_t flag_right_key = false;         // tester control
uint8_t flag_mid_key = false;           // tester control

/* USER CODE END PV */
```

Các biến được comment là ECU control là được điều khiển bởi ECU và tester control sẽ được điều khiển bởi tester.

Biến *tester_TxHeader*, *ECU_TxHeader* chứa thông tin header của mỗi Node truyền.

Biến *tester_RxHeader*, *ECU_RxHeader* chứa thông tin header của mỗi Node nhận.

Mảng *tester_TxData []*, *ECU_TxData []* chứa dữ liệu cần gửi của mỗi Node.

Mảng *tester_RxData []*, *ECU_RxData []* chứa dữ liệu nhận được của mỗi Node.

Biến *tester_TxMailbox*, *ECU_TxMailbox* lưu header và data vào đây để gửi đi.

Mảng *ECU_store_seed []* lưu các seed được gửi đi để khi nhận về key sẽ tính ra key tương ứng với những seed đó để so sánh với key nhận được.

Biến *joystick* dung để lưu vị trị của joystick, sẽ lưu giữ giá trị LEFT, MID, RIGHT.

Biến *count* sẽ tăng lên 1 đơn vị mỗi khi timer tràn, đến khi tăng đến giá trị over_time thì sẽ lock hệ thống lại, bình thường hệ thống mặc định là lock nên count không cần chạy và được gán mặc định bằng 0.

Biến *is_ECU_unlock* cờ trạng thái hệ thống đang lock hay không lock, mặc định sẽ là lock nên được gắn false.

Biến *is_run_timer* cờ được bật khi hệ thống bắt đầu rãnh để khởi động biến count.

Biến *is_get_fifo0* cờ báo hiệu có nhận dữ liệu từ CAN1(FIFO0).

Biến *is_get_fifo1* cờ báo hiệu có nhận dữ liệu từ CAN2(FIFO1).

Biến *flag_wake_up_key* cờ báo hiệu có ngắt từ chân WakeUp.

Biến *flag_user_key* cờ báo hiệu có ngắt từ chân USER.

Biến *flag_left_key* cờ báo hiệu có ngắt từ chân LEFT.

Biến *flag_right_key* cờ báo hiệu có ngắt từ chân RIGHT.

Biến *flag_mid_key* cờ báo hiệu có ngắt từ chân MID.

Biến *is_over_time* cờ báo hiệu hết over time và hệ thống sẽ tiến hành lock.

4. Private function prototypes:

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_CAN1_Init(void);
static void MX_CAN2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void HAL_GPIO_EXTI_Callback(uint16_t);
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan);
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan);    // ECU control
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);        // ECU control
void securityAccessServiceSeedRequestFrame();                       // tester control
void securityAccessServiceSeedResponseFrame();                      // ECU control
void securityAccessSendKeyRequestFrame(uint8_t Tx_Data[], uint8_t Data[]);  // tester control
void securityAccessSendKeyResponseFrame();                          // ECU control
void SecurityAccessNegativeResponseMessage(uint8_t Data[], uint8_t flag_NRC);
void ReadDataByIdentifierRequestFrame(uint8_t Data[]);                      // tester control
void ReadDataByIdentifierResponseFrame(uint8_t Data[], uint8_t adc_value);  // ECU control
void WriteDataByIdentifierRequestFrame(uint8_t Data[], uint8_t joystick_position);
void WriteDataByIdentifierResponseFrame(uint8_t Data[]);            // ECU control
void Service22();                                   // tester control
void Service27();                                   // tester control
void Service2E(uint8_t joystick_position);          // tester control
void CheckAndHandleService22();                     // tester control
void CheckAndHandleService27();                     // tester control
void CheckAndHandleService2E();                     // tester control
void CheckAndHandleTesterCANFIFO0();                // tester control
void CheckAndHandleECUCANFIFO1();                   // ECU control
void CheckOverTime();                               // ECU control
void HandleSID22();                                 // ECU control
void HandleSID27();                                 // ECU control
void HandleSID2E();                                 // ECU control
void HandleSID62();                                 // tester control
void HandleSID67();                                 // tester control
void HandleSID6E();                                 // tester control
void HandleSID7F();                                 // tester control
void store_joystick_value(uint8_t Data);            // ECU control
uint8_t is_accept_key(uint8_t Data[],uint8_t  store_seed[]);    // ECU control
uint8_t get_PCI(uint8_t);
uint8_t get_size(uint8_t);
uint8_t get_SID(uint8_t Data[]);
uint16_t get_DID(uint8_t Data[]);
uint8_t get_sub_function(uint8_t Data[]);
uint8_t get_NRC(uint8_t Data[]);
uint8_t generate_seed();
uint8_t cal_key(uint8_t);

/* USER CODE END PFP */
```

Hàm HAL_GPIO_EXTI_Callback(uint16_t) được gọi khi có ngắt nút nhấn.

Hàm HAL_CAN_RxFifo0MsgPendingCallback (CAN_HandleTypeDef *hcan) được gọi khi ngắt CAN_IT_RX_FIFO0_MSG_PENDING được bật.

Hàm HAL_CAN_RxFifo1MsgPendingCallback (CAN_HandleTypeDef *hcan) được gọi khi ngắt CAN_IT_RX_FIFO1_MSG_PENDING được bật.

Hàm HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim): được gọi khi timer tràn.

Hàm securityAccessServiceSeedRequestFrame () được gọi để cài đặt gói tin request seed cho $27.

Hàm securityAccessServiceSeedResponseFrame () được gọi để cài đặt gói tin response seed cho $27.

Hàm securityAccessSendKeyRequestFrame(uint8_t Tx_Data[], uint8_t Data[]) được gọi để cài đặt gói tin request key cho $27.

Hàm securityAccessSendKeyResponseFrame() được gọi để cài đặt gói tin response key cho $27.

Hàm SecurityAccessNegativeResponseMessage(uint8_t Data[], uint8_t flag_NRC) được gọi để cài đặt gói tin Negative response khi truyền nhận không thành công.

Hàm ReadDataByIdentifierRequestFrame(uint8_t Data[]) được gọi để cài gói tin cho request $22.

Hàm ReadDataByIdentifierResponseFrame(uint8_t Data[], uint8_t adc_value) được gọi để cài gói tin cho response $22.

Hàm WriteDataByIdentifierRequestFrame(uint8_t Data[], uint8_t joystick_position) được gọi để cài gói tin cho request $2E.

Hàm WriteDataByIdentifierResponseFrame(uint8_t Data[]) được gọi để cài gói tin cho response $2E.

Hàm Service22() để tester bắt đầu $22.

Hàm Service27() để tester bắt đầu $27.

Hàm Service2E (uint8_t joystick_position) để tester bắt đầu $2E.

Hàm CheckAndHandleService22() để ECU kiểm tra xem có nhận được $22 thì sẽ xử lí.

Hàm CheckAndHandleService27() để ECU kiểm tra xem có nhận được $27 thì sẽ xử lí.

Hàm CheckAndHandleService2E() để ECU kiểm tra xem có nhận được $2E thì sẽ xử lí.

Hàm CheckAndHandleTesterCANFIFO0() để tester kiểm tra xem có nhận được tín hiệu từ FIFO0 thì sẽ xử lí.

Hàm CheckAndHandleECUCANFIFO1() để ECU kiểm tra xem có nhận được tín hiệu từ FIFO1 thì sẽ xử lí.

Hàm CheckOverTime() kiểm tra cờ is_over_time và xử lí.

Hàm HandleSID22() để ECU xử lí SID22.

Hàm HandleSID27() để ECU xử lí SID27.

Hàm HandleSID2E() để ECU xử lí SID2E.

Hàm HandleSID62() để tester xử lí SID62.

Hàm HandleSID67() để tester xử lí SID67.

Hàm HandleSID6E() để tester xử lí SID6E.

Hàm HandleSID7F() để tester xử lí SID7F.

Hàm store_joystick_value(uint8_t Data) để lưu giá trị joystick nhận được từ tester vào biến joystick của ECU.

Hàm is_accept_key(uint8_t Data[],uint8_t store_seed[]) để so sánh key tính được từ tester và ECU.

Hàm get_PCI(uint8_t) để lấy PCI.

Hàm get_size(uint8_t) để lấy size của gói tin.

Hàm get_SID(uint8_t Data[]) để lấy SID từ các gói tin.

Hàm get_DID(uint8_t Data[]) để lấy DID từ $22 và $2E.

Hàm get_sub_function(uint8_t Data[]) để lấy sub-function từ $27.

Hàm get_NRC(uint8_t Data[]) để lấy NRC từ gói tin SID7F.

Hàm generate_seed() để tạo seed cho $27.

Hàm cal_key(uint8_t) để tạo key cho $27.

5. CODE BEGIN 2:

```
/* USER CODE BEGIN 2 */
HAL_ADC_Start(&hadc1);            // khoi dong module ADC
HAL_TIM_Base_Start_IT(&htim3);    // khoi dong module timer
HAL_CAN_Start(&hcan1);            // khởi động module can1
HAL_CAN_Start(&hcan2);            // khởi động module can2
HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);  // Activate the notification
HAL_CAN_ActivateNotification(&hcan2, CAN_IT_RX_FIFO1_MSG_PENDING);  // Activate the notification
/* USER CODE END 2 */
```

Hàm HAL_ADC_Start(&hadc1): khởi động ADC

Hàm HAL_TIM_Base_Start_IT(&htim3): khởi động timer và ngắt khi tràn.

Hàm HAL_CAN_Start(&hcan1), HAL_CAN_Start(&hcan2): khởi động CAN.

Hàm HAL_CAN_ActivateNotification (&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING), HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO1_MSG_PENDING): kích hoạt ngắt, bất cứ khi nào có một số thông báo đang chờ xử lý trong *CAN_IT_RX_FIFO0_MSG_PENDING* hay *CAN_IT_RX_FIFO1_MSG_PENDING*. Khi ngắt được kích hoạt, chức năng gọi lại sẽ làm chạy hàm HAL_CAN_RxFifo1MsgPendingCallback nếu là FIFO 1 và HAL_CAN_RxFifo0MsgPendingCallback nếu là FIFO 0.

6. CODE BEGIN WHILE:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    CheckAndHandleService27();
    CheckAndHandleService22();
    CheckAndHandleService2E();
    CheckAndHandleTesterCANFIFO0();
    CheckAndHandleECUCANFIFO1();
    CheckOverTime();
  /* USER CODE END WHILE */

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

Khi vào vòng lặp while sẽ lần lượt gọi các hàm CheckAndHandleService27, CheckAndHandleService22, CheckAndHandleService2E để khi cờ ngắt bật thì sẽ thực hiện các service này. Khi có tín hiệu từ FIFO0 hoặc FIFO1 sẽ được ngắt và bật cờ để xử lí các tín hiệu này. Gọi hàm CheckOverTime để lock lại ECU khi hết thời gian.

7. CODE BEGIN CAN INIT

```
/* USER CODE BEGIN CAN1_Init 0 */
  tester_TxHeader.DLC = data_length;  // data length
  tester_TxHeader.IDE = CAN_ID_STD;
  tester_TxHeader.RTR = CAN_RTR_DATA;
  tester_TxHeader.StdId = tester_ID;  // ID
/* USER CODE END CAN1_Init 0 */
```

```
/* USER CODE BEGIN CAN2_Init 0 */
  ECU_TxHeader.DLC = data_length;  // data length
  ECU_TxHeader.IDE = CAN_ID_STD;
  ECU_TxHeader.RTR = CAN_RTR_DATA;
  ECU_TxHeader.StdId = ECU_ID;  // ID
/* USER CODE END CAN2_Init 0 */
```

Khởi tạo thông tin cho Header với:

DLC là kích thước dữ liệu.

IDE là kích thước ID: 29 với mở rộng (CAN) và 11 với tiêu chuẩn (*CAN_ID_STD* là tiêu chuẩn). Ở đây chọn ID tiêu chuẩn.

RTR là chọn frame truyền: Data frame (truyền dữ liệu), Remote frame (yêu cầu Node khác truyền khung dữ liệu có ID (IDENTIFIER) trùng với khung yêu cầu), Error frame (truyền bởi bất kỳ Node nào khi Node đó phát hiện lỗi bus), Overload frame (tạo thêm độ trễ giữa giữa các khung dữ liệu hoặc khung yêu cầu). Ở đây chọn truyền dữ liệu (*CAN_RTR_DATA*).

StdId là thông tin ID của Node.

8. CODE BEGIN 2

```
347  /* USER CODE BEGIN CAN1_Init 2 */
348  CAN_FilterTypeDef canfilterconfig;
349  canfilterconfig.FilterActivation = CAN_FILTER_ENABLE;
350  canfilterconfig.FilterBank = 18;  // which filter bank to use from the assigned ones
351  canfilterconfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
352  canfilterconfig.FilterIdHigh = ECU_ID<<5;
353  canfilterconfig.FilterIdLow = 0;
354  canfilterconfig.FilterMaskIdHigh = ECU_ID<<5;
355  canfilterconfig.FilterMaskIdLow = 0x0000;
356  canfilterconfig.FilterMode = CAN_FILTERMODE_IDMASK;
357  canfilterconfig.FilterScale = CAN_FILTERSCALE_32BIT;
358  canfilterconfig.SlaveStartFilterBank = 20;  // how many filters to assign to the CAN1 (master can)
359  HAL_CAN_ConfigFilter(&hcan1, &canfilterconfig);
360
361  /* USER CODE END CAN1_Init 2 */
```

```
401   /* USER CODE BEGIN CAN2_Init 2 */
402   CAN_FilterTypeDef canfilterconfig;
403   canfilterconfig.FilterActivation = CAN_FILTER_ENABLE;
404   canfilterconfig.FilterBank = 10;  // which filter bank to use from the assigned ones
405   canfilterconfig.FilterFIFOAssignment = CAN_FILTER_FIFO1;
406   canfilterconfig.FilterIdHigh = tester_ID<<5;
407   canfilterconfig.FilterIdLow = 0;
408   canfilterconfig.FilterMaskIdHigh = tester_ID<<5;
409   canfilterconfig.FilterMaskIdLow = 0x0000;
410   canfilterconfig.FilterMode = CAN_FILTERMODE_IDMASK;
411   canfilterconfig.FilterScale = CAN_FILTERSCALE_32BIT;
412   canfilterconfig.SlaveStartFilterBank = 0;  // doesn't matter in single can controllers
413   HAL_CAN_ConfigFilter(&hcan2, &canfilterconfig);
414
415   /* USER CODE END CAN2_Init 2 */
```

Để giảm tải CPU Load, chỉ những gói tin đáp ứng được yêu cầu của Filter mới cho phép vượt qua với các thông tin của Filter như sao:

FilterActivation: công tắt bật Filter hay tắt Filter.

FilterBank: filter sẽ được sử dụng trong tiến trình

FilterFIFOAssignment: FIFO nào sử dụng để nhận message

FilterIdHigh: 16 bits cao của thanh ghi ID. Ta chỉ so sánh STD ID (bắt đầu từ bit thứ 5) vì thế ta dịch phải 5 bit

FilterIdLow: dùng cho CAN LOW

FilterMaskIdHigh: 16 bits cao của Mask register

FilterMaskIdLow: dùng cho CAN LOW

FilterMode: với MaskMode, Mask register sẽ so sánh 1 số bit cụ thể của thanh ghi ID và ID truyền đến

*FilterScale:*chọn bộ lọc 32bit hoặc 16bit

SlaveStartFilterBank: số lượng Filter Bank muốn gán cho CAN

Dùng hàm HAL_CAN_ConfigFilter(&hcan, &canfilterconfig) để ghép filter vào CAN.

9. CODE BEGIN 4

1. Các hàm được xử lí bởi ECU:

a. HAL_TIM_PeriodElapsedCallback ():

```
915 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
916     if (htim->Instance == TIM3){
917         if (is_run_timer == true){
918             count++;
919             if (count % 1000 == 0) printf("%d\n\r", count);
920             if (count == over_time) {
921                 is_over_time = true;
922                 count %= over_time;
923             }
924         }
925         else count = zero;
926     }
927 }
```

Khi timer tràn, nếu cờ is_run_timer được bật thì sẽ tăng biến count lên 1 đơn vị mỗi lần như vậy sẽ là 0.001s, nếu mỗi lần count chia hết cho 1000 nghĩa là trôi qua 1s sẽ hiển thị mốc thời gian lên màn hình Hercules để tiện theo dõi móc thời gian. Dòng lệnh "*printf("%d\n\r", count)*" là không được gọi vì timer được điều khiển bởi ECU nên tester sẽ không xem được trực tiếp như vậy, việc viết câu lệnh này là để quan sát móc thời gian.

Khi count tiến đến bằng over_time được define sẵn, sẽ bật cờ is_over_time để tiến hành lock ECU lại và reset lại giá trị của count.

b. HAL_CAN_RxFifo1MsgPendingCallback ():

```
907 void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan){
908     HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &ECU_RxHeader, ECU_RxData);
909     if (ECU_RxHeader.DLC == data_length){
910         is_get_fifo1 = true;
911         is_run_timer = false;
912         count = 0;
913     }
914 }
```

Khi nhảy vào hàm HAL_CAN_RxFifo1MsgPendingCallback kiểm tra DLC với data_length nếu khớp nhau sẽ bật cờ is_get_fifo1 để xử lí trong main. Tắt cờ chạy timer vì khi ECU nhận được lệnh nghĩa là ECU không rãnh nên sẽ không phải đếm (chỉ đếm khi ECU bắt đầu rãnh).

c. CheckOverTime ():

```
858 void CheckOverTime(){
859     if (is_over_time == true){
860         is_over_time = false;
861         is_run_timer = false;
862         count = 0;
863         is_ECU_unlock = false;
864         HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
865         printf("System's locked\n\r");
866     }
867 }
```

Khi cờ is_over_time được bật, nghĩa là cần phải lock ECU lại vì không còn yêu cầu lệnh tester. Tắt LED0 để báo hiệu hệ thống đã bị lock và hiện dòng chữ "System's locked" lên màn hình Hercules để tiện theo dõi. Dòng lệnh "*printf" (*System's locked"*)*" là không được gọi vì cờ trạng thái lock ECU hay không được điều khiển bởi ECU nên tester sẽ không xem được trực tiếp như vậy, việc viết câu lệnh này là để quan sát ECU đã bị lock.

d. CheckAndHandleECUCANFIFO1 ():

```
816 void CheckAndHandleECUCANFIFO1(){
817     if (is_get_fifo1 == true){
818         is_get_fifo1 = false;
819         uint8_t PCI = get_PCI(ECU_RxData[0]);
820         uint8_t LEN = get_size(ECU_RxData[0]);
821         if (PCI == 0x00){
822             uint8_t SID = get_SID(ECU_RxData);
823             if (is_ECU_unlock == true){
824                 if (SID == 0x22){
825                     if (LEN == 0x03) HandleSID22();
826                     else {
827                         SecurityAccessNegativeResponseMessage(ECU_TxData, 0x13);
828                         HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
829                     }
830                 }
831                 else if (SID == 0x2E) {
832                     if (LEN == 0x04) HandleSID2E();
833                     else {
834                         SecurityAccessNegativeResponseMessage(ECU_TxData, 0x13);
835                         HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
836                     }
837                 }
838                 else if (SID == 0x27){
839                     if (LEN == 0x07 || LEN == 0x02) HandleSID27();
840                 }
841             }
842             else if (is_ECU_unlock == false){
843                 if (SID == 0x27){
844                     if (LEN == 0x07 || LEN == 0x02) HandleSID27();
845                     else {
846                         SecurityAccessNegativeResponseMessage(ECU_TxData, 0x13);
847                         HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
848                     }
849                 }
850                 else {
851                     SecurityAccessNegativeResponseMessage(ECU_TxData, 0x33);
852                     HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
853                 }
854             }
855         }
856     }
857 }
```

Khi cờ is_get_fifo1 được bật nghĩa là ECU vừa nhận dữ liệu từ tester.

Lọc lấy PCI nếu PCI bằng 0x00 nghĩa là cho phép truyền gói tin thì sẽ kiểm tra SID.

Vì khi ECU nhận lệnh từ tester sẽ có 2 hoàn cảnh là ECU đã được unlock và chưa được unlock nên ECU sẽ kiểm tra cờ is_unlock_ECU:

Nếu đã unlock thì tùy vào SID và gọi các hàm Handle tương tứng, ngược lại sẽ gửi NegativeResponse cho tester.

e. HandleSID2E ():

```
707 void HandleSID2E(){
708     uint16_t DID = get_DID(ECU_RxData);
709     if (DID == 0xF112){
710         store_joystick_value(ECU_RxData[4]);
711         WriteDataByIdentifierResponseFrame(ECU_TxData);
712         is_run_timer = true;
713     }
714     else{
715         SecurityAccessNegativeResponseMessage(ECU_TxData, 0x12);
716     }
717     HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
718 }
```

Khi cần xử lí SID2E nếu kiểm tra đúng DID sẽ tiến hành ghi giá trị joystick từ tester vào biến joystick của ECU như yêu cầu của $2E, ngược lại nếu DID không đúng sẽ tra về NegativeResponse cho tester.

f. HandleSID27 ():

```
683 void HandleSID27(){
684     uint8_t SBF = get_sub_function(ECU_RxData);
685     if (SBF == 0x01){
686         securityAccessServiceSeedResponseFrame(ECU_TxData);
687     }
688     else if (SBF == 0x02){
689         if (is_accept_key(ECU_RxData, ECU_store_seed) == true){
690             if (is_ECU_unlock == true){
691                 is_ECU_unlock = false;
692                 securityAccessSendKeyResponseFrame(ECU_TxData);
693             }
694             else{
695                 is_ECU_unlock = true;
696                 is_run_timer = true;
697                 securityAccessSendKeyResponseFrame(ECU_TxData);
698             }
699         }
700         else {
701             SecurityAccessNegativeResponseMessage(ECU_TxData, 0x35);// invalid key
702         }
703     }
704     else SecurityAccessNegativeResponseMessage(ECU_TxData, 0x12);   // invalid sbf
705     HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
706 }
```

Khi cần xử lí SID27, nếu sub-function bằng 0x01 sẽ tiến hành gửi seed cho tester, nếu sub-function bang 0x02 nghĩa là đã nhận được key từ tester thì kiểm tra xem key đó đã đúng chưa, nếu đúng thì sẽ lock hoặc unlock ECU tùy vào trạng thái của ECU hiện tại, ngược lại nếu key nhận được không khớp với key tính được từ ECU sẽ gửi NegativeResponse cho tester, nếu sub-function khác 0x01 và cả 0x02 thì sẽ NegativeResponse cho tester theo mã là 0x12 có nghĩa là “invalid sub-funtion”.

g. HandleSID22 ():

```
669 void HandleSID22(){
670     uint16_t DID = get_DID(ECU_RxData);
671     if (DID == 0xF002){
672         is_run_timer = true;
673         HAL_ADC_Start(&hadc1);
674         HAL_Delay(10);
675         uint8_t adc_value = HAL_ADC_GetValue(&hadc1);
676         ReadDataByIdentifierResponseFrame(ECU_TxData, adc_value);
677     }
678     else{
679         SecurityAccessNegativeResponseMessage(ECU_TxData, 0x12);
680     }
681     HAL_CAN_AddTxMessage(&hcan2, &ECU_TxHeader, ECU_TxData, &ECU_TxMailbox);
682 }
```

Khi cần xử lí SID22 nếu kiểm tra đúng DID, ECU sẽ tiến hành đọc giá trị ADC của ECU như yêu cầu của $22 cho tester, ngược lại nếu DID không đúng sẽ tra về NegativeResponse cho tester.

h. SecurityAccessNegativeResponseMessage ():

```
653 void SecurityAccessNegativeResponseMessage(uint8_t Data[], uint8_t flag_NRC){
654     Data[0] = 0x03; // PCI la 0, size la 3
655     Data[1] = 0x7F; // NRC code
656     switch(flag_NRC) {
657         case 0x12:
658             Data[2] = 0x12; // sub function not supported
659             break;
660         case 0x35:
661             Data[2] = 0x35; // invalid key
662             break;
663         case 0x13:
664             Data[2] = 0x13; // incorrect message length or invalid format
665         default:
666             Data[2] = 0x33; //securityAccessDenied
667     }
668 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), vì là Nevgative Response nên SID sẽ là 7F và byte còn lại là cờ NRC để hiển thị lí do truyền nhận không thành công.

i. securityAccessSendKeyResponseFrame ():

```
646 void securityAccessSendKeyResponseFrame(uint8_t Data[]){
647     Data[0] = 0x02; // PCI la 0, size la 2
648     Data[1] = 0x67; // response $27 nen la $67
649     Data[2] = 0x02; // response seed = request seed + 1
650     if (is_ECU_unlock == true) Data[3] = false; // unlock or lock
651     else Data[3] = true;
652 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), byte tiếp theo sẽ là SBF(Sub-function) theo định dạng của $67, byte tiếp theo sẽ là response key (theo qui ước sẽ là số lẽ các gói tin key của seed đó sẽ là số đó cộng 1 đơn vị nên sẽ là số chẵn), byte còn lại cho biết xem là đã unlock hay lock ECU.

j. securityAccessServiceSeedResponseFrame ():

```
void securityAccessServiceSeedResponseFrame(uint8_t Data[]){
    Data[0] = 0x07; // PCI la 0, size la 4
    Data[1] = 0x67; // response $27 nen la $67
    Data[2] = 0x01; // request seed
    Data[3] = generate_seed();  // Security Seed
    ECU_store_seed[3] = Data[3];
    Data[4] = generate_seed();  // Security Seed
    ECU_store_seed[4] = Data[4];
    Data[5] = generate_seed();  // Security Seed
    ECU_store_seed[5] = Data[5];
    Data[6] = generate_seed();  // Security Seed
    ECU_store_seed[6] = Data[6];
    Data[7] = generate_seed();  // Security Seed
    ECU_store_seed[7] = Data[7];
}
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), byte tiếp theo sẽ là SBF(Sub-function) theo định dạng của $27, các byte tiếp theo sẽ là seed được tạo ra ngẫu nhiên gửi cho tester và lưu các seed đó trong ECU để khi nhận được key từ tester thì tạo ra key từ ECU để kiểm tra 2 key với nhau.

k. WriteDataByIdentifierResponseFrame ():

```
void WriteDataByIdentifierResponseFrame(uint8_t Data[]){
    Data[0] = 0x03; // PCi, size
    Data[1] = 0x6E; // PSID
    Data[2] = 0xF1; // DID
    Data[3] = 0x12; // DID
}
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), 2 bytes tiếp theo là DID (Data Identifier) theo phản hồi của $2E là $6E.

l. ReadDataByIdentifierResponseFrame ():

```
void ReadDataByIdentifierResponseFrame(uint8_t Data[], uint8_t adc_value){
    Data[0] = 0x04; // PCI
    Data[1] = 0x62; // PSID
    Data[2] = 0xF0; // DID
    Data[3] = 0x02; // DID
    Data[4] = adc_value;    // data
}
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), 2 bytes tiếp theo là DID (Data Identifier) và byte cuối cùng sẽ là adc_value đọc được từ ECU theo phản hồi của $22 là $62.

m. is_accept_key ():

```
581 uint8_t is_accept_key(uint8_t Data[],uint8_t  store_seed[]){
582     for (int i = 3; i < 8; i++){
583         if (Data[i] != cal_key(store_seed[i])) return false;
584     }
585     return true;
586 }
```

Khi mọi Data nhận được từ tester đều giống với key được tính từ ECU sẽ giả về true cho biết key đã đúng là sẽ tiến hành unlock ECU theo $27.

n. generate_seed ():

```
575 uint8_t generate_seed(){
576     return (uint8_t)(HAL_GetTick());
577 }
```

Để tạo ra những cái seed random nên dung hàm HAL_GetTick để lấy được 1 giá trị không lường trước, vì HAL_GetTick sẽ trả về giá trị 32bits nên phải ép kiểu về 8bits.

o. store_joystick_value ():

```
554 void store_joystick_value(uint8_t Data){
555     joystick = Data;
556 }
```

Gán giá trị Data nhận được từ tester vào biến joystick của ECU theo $2E.

2. Các hàm được xử lí bởi tester:

a. HAL_CAN_RxFifo0MsgPendingCallback ():

```
901 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){
902     HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &tester_RxHeader, tester_RxData);
903     if (tester_RxHeader.DLC == data_length){
904         is_get_fifo0 = true;
905     }
906 }
```

Khi nhảy vào hàm HAL_CAN_RxFifo0MsgPendingCallback kiểm tra DLC với data_length nếu khợp nhau sẽ bật cờ is_get_fifo0 để xử lí trong main.

b. HAL_GPIO_EXTI_Callback ():

```
868 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
869 {
870   /* Prevent unused argument(s) compilation warning */
871     UNUSED(GPIO_Pin);
872     if(GPIO_Pin == WakeUp_Pin){
873         printf("Wake up\n\r");
874         flag_wake_up_key = true;
875         while(HAL_GPIO_ReadPin(WakeUp_GPIO_Port, WakeUp_Pin) == GPIO_PIN_RESET);
876     }
877     else if (GPIO_Pin == USER_Pin){
878         printf("User\n\r");
879         flag_user_key = true;
880         while(HAL_GPIO_ReadPin(USER_GPIO_Port, USER_Pin) == GPIO_PIN_RESET);
881     }
882     else if (GPIO_Pin == LEFT_Pin){
883         printf("Left\n\r");
884         flag_left_key = true;
885         while(HAL_GPIO_ReadPin(LEFT_GPIO_Port, LEFT_Pin) == GPIO_PIN_RESET);
886     }
887     else if (GPIO_Pin == RIGHT_Pin){
888         printf("Right\n\r");
889         flag_right_key = true;
890         while(HAL_GPIO_ReadPin(RIGHT_GPIO_Port, RIGHT_Pin) == GPIO_PIN_RESET);
891     }
892     else if (GPIO_Pin == MID_Pin){
893         printf("Mid\n\r");
894         flag_mid_key = true;
895         while(HAL_GPIO_ReadPin(MID_GPIO_Port, MID_Pin) == GPIO_PIN_RESET);
896     }
897   /* NOTE: This function Should not be modified, when the callback is needed,
898            the HAL_GPIO_EXTI_Callback could be implemented in the user file
899    */
900 }
```

Khi có tín hiệu từ nút nhấn, hàm ngắt được gọi. Đầu tiên sẽ in ra giá trị phím mà tester vừa được nhấn lên màn hình Hercules để tiện theo dõi, sau đó bật cờ ngắt tại vị trí đó để vào vòng while (1) trong hàm main xử lí khi có cờ được bật. Gọi hàm while HAL_GPIO_ReadPin tại vị trí đó để chống dội phím khi chương trình chạy rất nhanh tay chưa kịp lấy ra đã nhảy vào đây nhiều hơn 1 lần.

c. CheckAndHandleTesterCANFIFO0 ():

```
803 void CheckAndHandleTesterCANFIFO0(){
804     if (is_get_fifo0 == true){
805         is_get_fifo0 = false;
806         uint8_t PCI = get_PCI(tester_RxData[0]);
807         if (PCI == 0x00){
808             uint8_t SID = get_SID(tester_RxData);
809             if (SID == 0x62) HandleSID62();
810             else if (SID == 0x67) HandleSID67();
811             else if (SID == 0x6E) HandleSID6E();
812             else if (SID == 0x7F) HandleSID7F();
813         }
814     }
815 }
```

Khi cờ is_get_fifo0 được bật nghĩa là tester vừa nhận dữ liệu từ ECU.

Lọc lấy PCI nếu PCI bằng 0x00 nghĩa là cho phép truyền gói tin thì sẽ kiểm tra SID. Tùy vào SID và gọi các hàm Handle tương tứng.

a. CheckAndHandleService2E ():

```
789 void CheckAndHandleService2E(){
790     if (flag_left_key == true){
791         flag_left_key = false;
792         Service2E(LEFT);
793     }
794     else if (flag_mid_key == true){
795         flag_mid_key = false;
796         Service2E(MID);
797     }
798     else if (flag_right_key == true){
799         flag_right_key = false;
800         Service2E(RIGHT);
801     }
802 }
```

Hàm này được gọi liên tục trong vòng lặp while (1) của main để luôn kiểm tra xem nếu có cờ từ joystick sẽ gọi các hàm của $2E tương ứng để xử lí.

b. CheckAndHandleService27 ():

```
783 void CheckAndHandleService27(){
784     if (flag_wake_up_key == true){
785         flag_wake_up_key = false;
786         Service27();
787     }
788 }
```

Hàm này được gọi liên tục trong vòng lặp while (1) của main để luôn kiểm tra xem nếu có cờ từ nút wake up không, nếu có sẽ gọi các hàm của $27 tương ứng để xử lí unlock hoặc lock ECU.

c. CheckAndHandleService22 ():

```
777 void CheckAndHandleService22(){
778     if (flag_user_key == true){
779         flag_user_key = false;
780         Service22();
781     }
782 }
```

Hàm này được gọi liên tục trong vòng lặp while (1) của main để luôn kiểm tra xem nếu có cờ từ nút user không, nếu có sẽ gọi các hàm của $22 để yêu cầu ECU đọc giá trị ADC của ECU.

d. Service2E ():

```
773 void Service2E(uint8_t joystick_position){
774     WriteDataByIdentifierRequestFrame(tester_TxData, joystick_position);
775     HAL_CAN_AddTxMessage(&hcan1, &tester_TxHeader, tester_TxData, &tester_TxMailbox);
776 }
```

Gọi hàm WriteDataByIdentifierRequestFrame để cài đặt gói tin cho $2E và gửi đi cho ECU.

e. Service27 ():

```
769 void Service27(){
770     securityAccessServiceSeedRequestFrame(tester_TxData);
771     HAL_CAN_AddTxMessage(&hcan1, &tester_TxHeader, tester_TxData, &tester_TxMailbox);
772 }
```

Gọi hàm securityAccessServiceSeedRequestFrame để cài đặt gói tin cho $27 và gửi đi cho ECU.

f. Service22 ():

```
765 void Service22(){
766     ReadDataByIdentifierRequestFrame(tester_TxData);
767     HAL_CAN_AddTxMessage(&hcan1, &tester_TxHeader, tester_TxData, &tester_TxMailbox);
768 }
```

Gọi hàm ReadDataByIdentifierRequestFrame để cài đặt gói tin cho $22 và gửi đi cho ECU.

g. HandleSID7F ():

```
750 void HandleSID7F(){
751     uint8_t NRC = get_NRC(tester_RxData);
752     if (NRC == 0x12){
753         printf("subFunctionNotSupported\n\r");
754     }
755     else if (NRC == 0x35){
756         printf("invalidKey\n\r");
757     }
758     else if (NRC == 0x13){
759         printf("incorrectMessageLengthOrInvalidFormat\n\r");
760     }
761     else if (NRC == 0x33){
762         printf("securityAccessDenied\n\r");
763     }
764 }
```

Khi cần xử lí SID7F là cần thông báo cho tester biết tại sao lại không thực hiện thành công các yêu cầu từ tester. Đầu tiên là lấy NRC từ gói tin và tùy theo NRC nhận được mà thông báo tình trạng của việc truyền nhận.

h. HandleSID6E ():

```
743 void HandleSID6E(){
744     uint16_t DID = get_DID(tester_RxData);
745     if (DID == 0xF112){
746         printf("Written joystick position\n\r");
747         printf("joystick position: %x\n\r", joystick);
748     }
749 }
```

Khi cần xử lí SID6E kiểm tra đúng DID sẽ tiến hành đọc thông báo đã ghi thành công dữ liệu vào ECU của $2E là $6E. Dòng lệnh "*printf("joystick position: %x\n\r", joystick)*" là không được gọi vì joystick là dữ liệu của ECU nên tester sẽ không xem được trực tiếp như vậy, việc viết câu lệnh này là để quan sát giá trị joystick sau khi thực hiện thành công $2E.

i. HandleSID62 ():

```
736 void HandleSID62(){
737     uint16_t DID = get_DID(tester_RxData);
738     if (DID == 0xF002){
739         uint8_t adc_value = tester_RxData[4];
740         printf("ADC value: %x\n\r", adc_value);
741     }
742 }
```

Khi cần xử lí SID62 kiểm tra đúng DID sẽ tiến hành đọc ADC được gửi từ ECU theo phản hồi của $22 là $62.

j. HandleSID67 ():

```
719 void HandleSID67(){
720     uint8_t SBF = get_sub_function(tester_RxData);
721     if (SBF == 0x01){
722         securityAccessSendKeyRequestFrame(tester_TxData, tester_RxData);
723         HAL_CAN_AddTxMessage(&hcan1, &tester_TxHeader, tester_TxData, &tester_TxMailbox);
724     }
725     else if (SBF == 0x02){
726         if (tester_RxData[3] == false){
727             printf("Unlocked\n\r");
728             HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_SET);
729         }
730         else{
731             printf("Locked\n\r");
732             HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
733         }
734     }
735 }
```

Khi cần xử lí SID 67, tester sẽ gặp 2 trường hợp: 1 là khi SBF bằng 0x01 (tức là request seed thành công và nhận được seed từ ECU), lúc này sẽ gọi hàm securityAccessSendKeyRequestFrame để tính ra key và gửi key đó cho ECU để so sánh. 2 là khi SBF bằng 0x02(tức là việc so sánh key đã thành công và ECU đã được unlock – khi ECU ban đầu là lock hoặc lock – khi ECU ban đầu là unlock), hiển thị lên màn hình Hercules để báo hiệu trạng thái của ECU.

k. securityAccessSendKeyRequestFrame ():

```
636 void securityAccessSendKeyRequestFrame(uint8_t Tx_Data[], uint8_t Data[]){
637     Tx_Data[0] = 0x07;              // PCI la 0, size la 4
638     Tx_Data[1] = 0x27;              // response $27 nen la $67
639     Tx_Data[2] = 0x02;              // response seed = request seed + 1
640     Tx_Data[3] = cal_key(Data[3]);  // Security key
641     Tx_Data[4] = cal_key(Data[4]);  // Security key
642     Tx_Data[5] = cal_key(Data[5]);  // Security key
643     Tx_Data[6] = cal_key(Data[6]);  // Security key
644     Tx_Data[7] = cal_key(Data[7]);  // Security key
645 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), byte tiếp theo sẽ là SBF(Sub-function) theo định dạng của $27, byte tiếp theo sẽ là request key (theo qui ước sẽ là số lẽ các gói tin key của seed đó sẽ là số đó cộng 1 đơn vị nên sẽ là số chẵn), các byte còn lại sẽ là key theo seed nhận được từ ECU thông qua công thức tính key của tester.

l. securityAccessServiceSeedRequestFrame ():

```
616 void securityAccessServiceSeedRequestFrame(uint8_t Data[]){
617     Data[0] = 0x02; // PCI la 0, size la 2
618     Data[1] = 0x27; // service 27
619     Data[2] = 0x01; // request seed
620 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), byte tiếp theo sẽ là SBF(Sub-function) theo định dạng của $27 và byte còn lại sẽ là request seed(theo qui ước sẽ là số lẽ các gói tin key của seed đó sẽ là số đó cộng 1 đơn vị nên sẽ là số chẵn).

m. WriteDataByIdentifierRequestFrame ():

```
603 void WriteDataByIdentifierRequestFrame(uint8_t Data[], uint8_t joystick_position){
604     Data[0] = 0x04; // PCi, size
605     Data[1] = 0x2E; // SID
606     Data[2] = 0xF1; // DID
607     Data[3] = 0x12; // DID
608     Data[4] = joystick_position;    // data
609 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier), 2 bytes tiếp theo là DID(Data Identifier) và byte cuối là giá trị cần ghi cho $2E.

n. ReadDataByIdentifierRequestFrame ():

```
590 void ReadDataByIdentifierRequestFrame(uint8_t Data[]){
591     Data[0] = 0x03; // PCi, size
592     Data[1] = 0x22; // SID
593     Data[2] = 0xF0; // DID
594     Data[3] = 0x02; // DID
595 }
```

Định dạng gói tin theo SingleFrame có byte đầu tiên sẽ là PCI kết hợp với size, byte thứ 2 là SID (Service Identifier) và 2 bytes tiếp theo là DID (Data Identifier) các byte còn lại không sử dụng.

o. get_NRC ():

```
587 uint8_t get_NRC(uint8_t Data[]){
588     return Data[2];
589 }
```

Negative Response Codes (NRC) để xem vì sau gói tin không được chấp nhận. Sẽ là byte thứ 3 của gói tin Negative Response.

3. Các hàm khác:

a. PUTCHAR_PROTOTYPE:

```
928 PUTCHAR_PROTOTYPE
929 {
930   /* Place your implementation of fputc here */
931   /* e.g. write a character to the USART */
932   HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
933   return ch;
934 }
935 /* USER CODE END 4 */
```

Dùng HAL_UART_Transmit để chuyển từng kí tự của phép printf ra UART với time_out bằng 100.

b. cal_key ():

```
578 uint8_t cal_key(uint8_t Data){
579     return Data + 1;
580 }
```

Công thức tính key từ seed để so sánh key của tester và key của ECU, nếu giống nhau sẽ được unlock. Công thức đơn giản sẽ là Data đó cộng thêm 1 đơn vị.

c. get_DID ():

```
566 uint16_t get_DID(uint8_t Data[]){
567     uint16_t temp = Data[2];
568     temp <<= 8;
569     temp += Data[3];
570     return temp;
571 }
```

DID của $22 và $27 sẽ là byte thứ 3 và 4 nên sẽ là 16bits và lấy 8bits của Data[2] dịch phải 8bits cộng với Data[3].

d. get_SID ():

```
563 uint8_t get_SID(uint8_t Data[]){
564     return Data[1];
565 }
```

SID sẽ là byte tiếp theo vì gói tin này là dạng SingleFrame.

e. get_size ():

```
560 uint8_t get_size(uint8_t Data){
561     return Data & 0x0F;
562 }
```

Size gói tin sẽ là 4bits sau của byte data đầu tiên nên sẽ nhân với 0x0F để lấy 4bits sau.

f. get_PCI ():

```
557 uint8_t get_PCI(uint8_t Data){
558     return Data >> 4;
559 }
```

PCI sẽ là 4bits đầu của byte data đầu tiên nên sẽ dịch phải 4bits để lấy được 4bits đầu.