BÁO CÁO THỰC HÀNH BÀI 4

Môn học: **CHUYÊN ĐỀ THIẾT KẾ HỆ THỐNG NHÚNG 1**- Mã lớp: **CE437.N11**
Giảng viên hướng dẫn thực hành: Phạm Minh Quân

Thông tin sinh viên	MSSV — Họ và tên 20520211 — Trương Hữu Khang 20520219 — Nguyễn Linh Anh Khoa 20520597 — Phan Duy Thông
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

Câu 1. **Viết chương trình thực hiện dịch vụ $22H – Read Data by Identifier để đọc và hiện giá trị ADC qua UART**

Phần chương trình của Tester để gửi yêu cầu $22H đến ECU

```
uint8_t uds_request[2] = {0x22, 0xF1}; // uds request package
// send request to ECU
HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, uds_request, &Node1TxMsgMailBox);
```

Phần chương trình xử lí của ECU khi nhận được yêu cầu $22H

```
uint8_t message[8]; // store the message
HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &Node2_RxHeader, message); // get
message from FIFO
if (message[0] == 0x22) { // check UDS request service ID
   switch (message[1]) {
      case 0xF1:
           HAL_ADC_Start(&hadc1);
        uds_response[2] = HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);
        break;
     default:
        break;
      }
uds_response[0]   = message[0] + 0x40; // positive reponse
uds_response[1]   = message[1];
Node2_TxHeader.DLC = sizeof(uds_response);
HAL_CAN_AddTxMessage(hcan, &Node2_TxHeader, uds_response, &Node2TxMsgMailBox);
// send response
```

Giải thích:

uds_request bao gồm 2 byte:

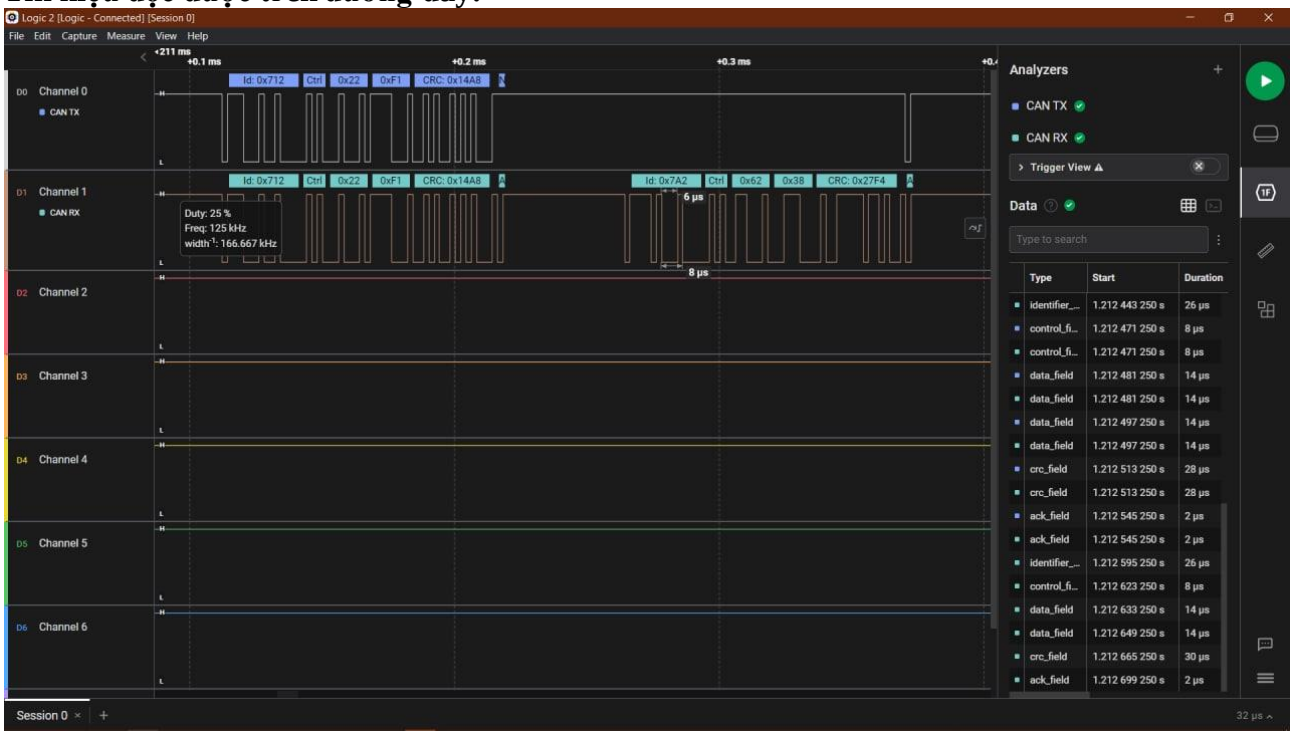
- Byte 1 là Service ID Read Data by Identifier
- Byte 2 là Data Identifier

Gói tin của UDS Protocol sẽ nằm trong phần Payload của một CAN Frame

Ở phía ECU sau khi nhận được gói tin sẽ kiểm tra Service ID của request và kiểm tra Data Identifier để tiến hành đọc dữ liệu

Sau khi đọc dữ liệu hoàn tất tiến hành reponse về Tester với positive reponse

Tín hiệu đọc được trên đường dây:



Câu 2. **Viết chương trình để thực hiện dịch vụ $2EH – Write Data by Identifier để ghi vị trí của Joy Stick xuống ECU và xuất dữ liệu ra UART:**

Phần chương trình của Tester

```
// khai báo các mảng dữ liệu
uint8_t const middle[10] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
uint8_t const left[10] = {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,
0xAA};
uint8_t const right[10] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF};
uint8_t const DataID[2] = {0xF0, 0x02};
GPIO_PinState a;
uint8_t message1[8] = {};
for (;;) {
     HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
     a = HAL_GPIO_ReadPin(Button_GPIO_Port, Button_Pin);
     if (a == GPIO_PIN_SET) {
        // First Frame
        message1[0] = 0x10;
        message1[1] = 0x0A;
        message1[2] = 0x2E;
        message1[3] = DataID[0];
        message1[4] = DataID[1];
        for (int i = 5; i < 8; i++) {
           message1[i] = middle[i - 5];
        }
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, message1,
&Node1TxMsgMailBox);
        message1[0] = 0x21;
        for (int i = 1; i < 8; i++) {
           message1[i] = middle[i + 2];
```

```
        }
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, message1,
&Node1TxMsgMailBox);
     } else {
        // First Frame
        message1[0] = 0x10;
        message1[1] = 0x0A;
        message1[2] = 0x2E;
        message1[3] = DataID[0];
        message1[4] = DataID[1];
        for (int i = 5; i < 8; i++) {
           message1[i] = right[i - 5];
        }
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, message1,
&Node1TxMsgMailBox);
        message1[0] = 0x21;
        for (int i = 1; i < 8; i++) {
           message1[i] = right[i + 2];
        }
        HAL_CAN_AddTxMessage(&hcan1, &Node1_TxHeader, message1,
&Node1TxMsgMailBox);
     }
     osEventFlagsSet(DoneTransferHandle, ACTIVATE_FLAG);
     osDelay(1000);
     HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &Node1_RxHeader, message1);
     if (message1[0] == 0x2E + 0x40) {
        printf("%s\n", "Success");
     }
  }
```

Phần chương trình của ECU:

```
uint32_t fifoLevel;
uint8_t raw_message[8] = {};
uint8_t message2[30]   = {};
uint32_t datalength;
uint8_t UDS_service_ID;
uint8_t DataID[2];
uint8_t reponse[3];
int i;
/* Infinite loop */
for (;;) {
    osEventFlagsWait(DoneTransferHandle, ACTIVATE_FLAG, osFlagsWaitAny,
osWaitForever);
    fifoLevel = HAL_CAN_GetRxFifoFillLevel(&hcan2, CAN_RX_FIFO1);
    if (fifoLevel != 0) {
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
        HAL_CAN_GetRxMessage(&hcan2, CAN_RX_FIFO1, &Node2_RxHeader,
raw_message);
        if (raw_message[0] & 0b11110000) {
            datalength     = raw_message[0] & 0b00001111;
            datalength     = (datalength << 4) + raw_message[1];
            UDS_service_ID = raw_message[2];
            DataID[0]    = raw_message[3];
```

```
            DataID[1]    = raw_message[4];
            for (i = 0; i + 5 < 8; i++) {
                message2[i] = raw_message[i + 5];
            }
        for (i = i; i < datalength; i++) {
            HAL_CAN_GetRxMessage(&hcan2, CAN_RX_FIFO1, &Node2_RxHeader,
raw_message);
            for (int x = 1; x < 8; x++) {
                message2[i] = raw_message[x];
                i++;
                if (i == datalength) break;
          }
       }
    } else {
        datalength = raw_message[0] & 0b00001111;
        for (i = 0; i < datalength; i++) {
        message2[i] = raw_message[i + 1];
      }
   }
    fifoLevel = 0;
    for (int x = 0; x < datalength; x++) {
        printf("%02X ", message2[x]);
    }
    printf("\n");
    Node2_TxHeader.DLC = 3;
    reponse[0]      = UDS_service_ID + 0x40;
    reponse[1]      = DataID[0];
    reponse[2]      = DataID[1];
    HAL_CAN_AddTxMessage(&hcan2, &Node2_TxHeader, reponse, &Node2TxMsgMailBox);
    }
}
```

Giải thích:

Do chương trình sử dụng RTOS để giả lập hoạt động của hai thiết bị nên có sử dụng thêm biến cờ để đồng bộ việc gửi và nhận ở cả hai Thread.

Ở Task của Tester, task sẽ tiến hành đọc giá trị của Joy Stick, sau đó tiến hành quá trình tạo ra các gói tin (do dữ liệu vượt quá 8 byte nên không thể gửi được trên một CAN Frame). Các gói tin này sẽ tuân theo format:

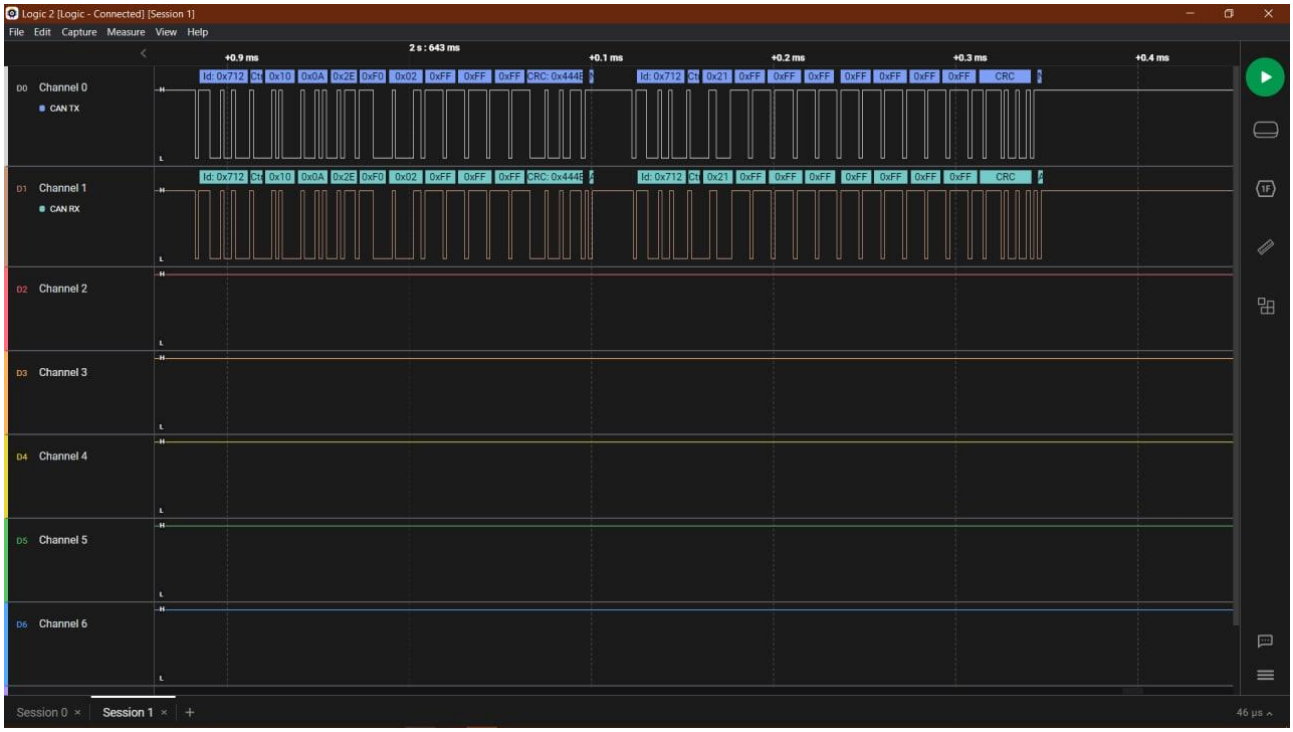
Frame Type	D0		D1		D2	
	Bits 7-4	Bits 3-0	Bits 7-4	Bits 3-0	Bits 7-4	Bits 3-0
Single Frame	0	Single frame Data Length	Not Applicable			
First Frame	1	Data Length of Multi-Frame Data			Not Applicable	
Consecutive Frame	2	Sequence Number	Not Applicable			
Flow Control Frame	3	Flow Status	Block Size		Separation Time Minimum	

Hình 1. Format của các gói tin CAN

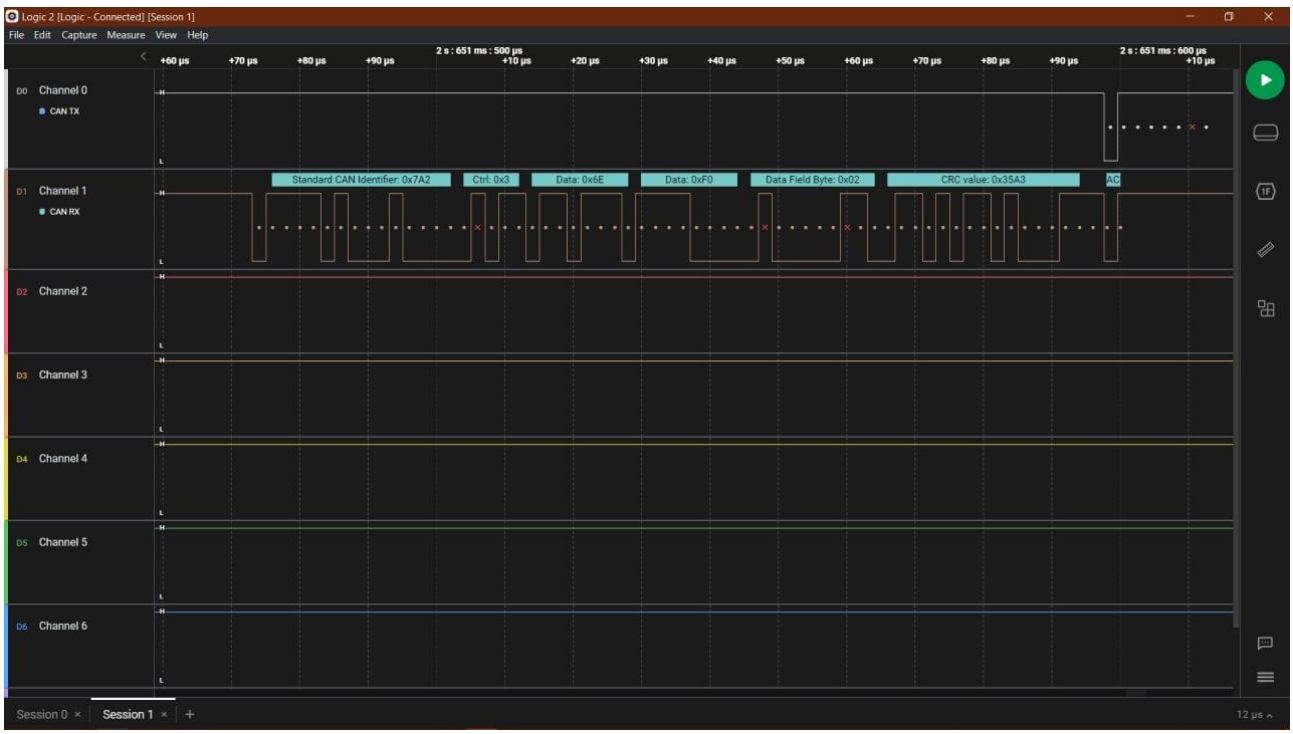
Do không yêu cầu việc phải cấu hình các giá trị cho lớp mạng nên gói tin Flow Control sẽ không được thực hiện.

Ở Task của ECU, sau khi nhận được cờ báo hiệu đã hoàn thành việc chuyển dữ liệu sẽ tiến hành ghép các gói tin lại với nhau. Các giá trị như DataLength, SequenceNumber sẽ được lưu lại để sử dụng cho quá trình ghép gói tin (việc lấy dữ liệu sẽ dựa vào format của gói tin có sẵn). Sau khi việc ghép gói tin hoàn tất ECU sẽ phản hồi về Tester với gói tin Positive Reponse. Tester sau khi nhận được gói tin Positive Reponse sẽ in "Success" ra UART

Các tín hiệu đọc được trên đường dây:



Hình 2. Các gói tin trong nhiều khung truyền CAN



Hình 3. Gói tin Positive Reponse do ECU gửi về cho Tester