**Báo cáo thực hành Lab 2 (19/04)**

Họ và tên: Phạm Đức Long

Mã số sinh viên: 20225737

**MỤC LỤC**

[**I, Lưu ý:**](#_8e5h7aeyha3x) **1**

[1. Cam kết](#_ocbu2prrpkmw) 1

[2. Hình ảnh, video](#_xs4m6fyvweo0) 1

[**II, Nội dung**](#_h7itvresfa2n) **2**

[3.1. Tìm hiểu module DS1307 + AT24C32 Tiny RTC](#_swp56jkxgm68) 2

[- Hai IC này có thể đồng thời hoạt động được không? Tại sao?](#_gpekstny10l2) 2

[- Các chân tín hiệu cần để ghép nối với module Tiny RTC là gì?](#_4h7drbhdzgwm) 2

[- Địa chỉ của DS1307 và AT24C32 tương ứng là bao nhiêu?](#_jy5bce2u3437) 2

[3.6. Lập trình ghép nối DS1307](#_1vkz284gmys5) 4

[3.7. Lập trình ghép nối SH1106](#_u1jsx48l464k) 5

[3.8. Lập trình với RC522](#_c3oqy73m5diq) 6

[3.9. Bài tập tổng hợp](#_j4br4pje8uze) 6

[3.9.1. Kiểu cấu trúc dùng trong bài](#_w5d5xi8qzh1r) 7

[3.9.2. Các biến sử dụng](#_vxxyvu1xhytd) 8

[3.9.3. Các hàm](#_na7tewiw6y5z) 8

[3.7.\*. Bài bonus: BCD Timer (Source trong file zip bai3.6\_3.7\_3.8.zip)](#_ylcblfh9ircn) 14

# **I, Lưu ý:**

## **1. Cam kết**

- Nội dung và mã nguồn trong báo cáo thực hành này là do em và bạn Nguyễn Bá Hoàng (20225844) cùng tự làm. Bất cứ nội dung nào tham khảo từ bên ngoài thì sẽ được nêu rõ nguồn gốc và tác giả.

## **2. Hình ảnh, video**

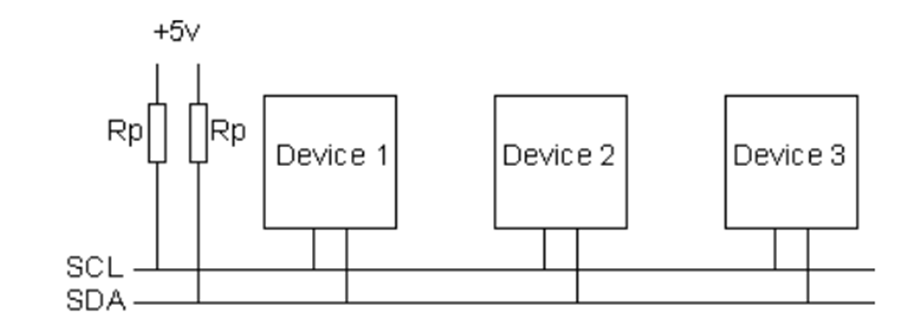
- Tất cả các hình ảnh, video về các bài tập trong buổi thực hành được nhóm em chụp và quay sẽ có ở [link](https://drive.google.com/drive/folders/1GEKJKogC9gpHEZ627_BRftsk73HRnEK7?usp=sharing) để đảm bảo chất lượng hình ảnh, video và dung lượng file báo cáo ạ.

# **II, Nội dung**

## **3.1. Tìm hiểu module DS1307 + AT24C32 Tiny RTC**

### - *Hai IC này có thể đồng thời hoạt động được không? Tại sao?*

-> Hai IC DS1307 và AT24C32 có thể hoạt động cùng lúc trên cùng một bus vì cả hai đều sử dụng giao tiếp I2C và có địa chỉ riêng biệt. DS1307 có địa chỉ mặc định là 0x68 (1101000), trong khi AT24C32 sử dụng địa chỉ cơ bản là 0x57 (1010111), có thể điều chỉnh được thông qua các chân A0 đến A2. Giao thức I2C còn hỗ trợ nhiều thiết bị hoạt động trên cùng đường truyền SDA và SCL, miễn là mỗi thiết bị có địa chỉ khác nhau.



### - *Các chân tín hiệu cần để ghép nối với module Tiny RTC là gì?*

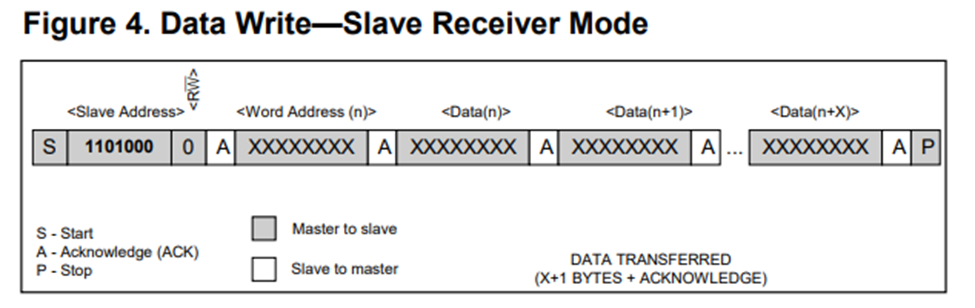
-> Các chân nguồn: VCC -> 5V, GND -> Nối đất, BAT: sử dụng khi không có PIN

-> Các chân tín hiệu: SCL -> chân xung clock (chân PA8), SDA -> chân dữ liệu (chân PC9)

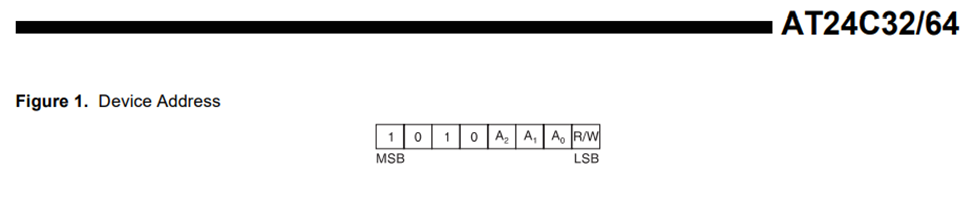
### *- Địa chỉ của DS1307 và AT24C32 tương ứng là bao nhiêu?*

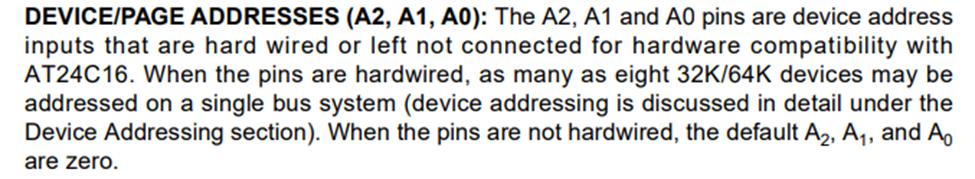
-> DS1307: 0x68 (1101000)

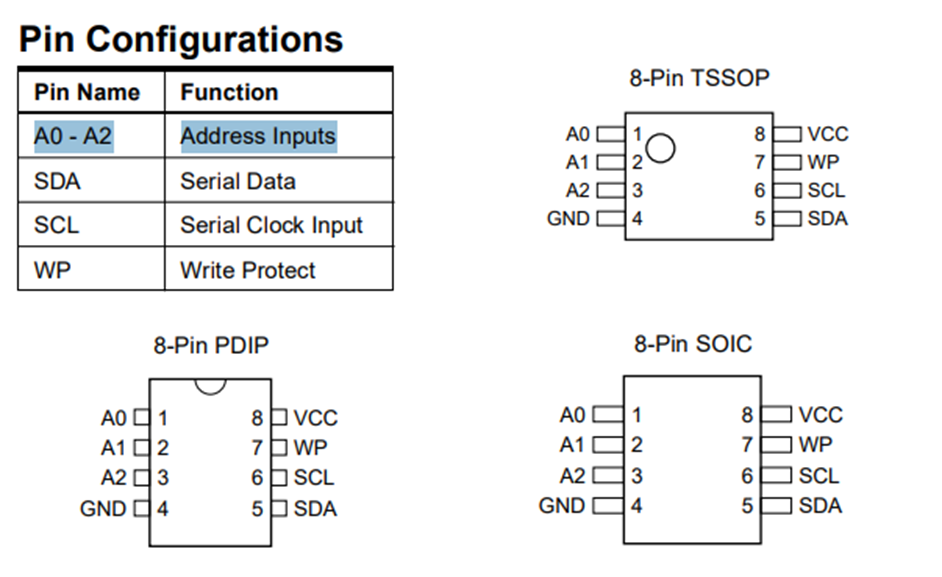




-> AT24C32: 0x57 (1010111) (Mặc định nhưng có thể bị thay đổi theo A0-A2)



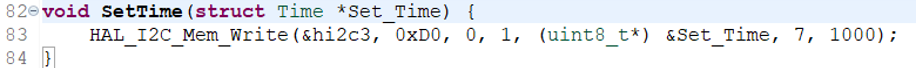




## **3.6. Lập trình ghép nối DS1307**

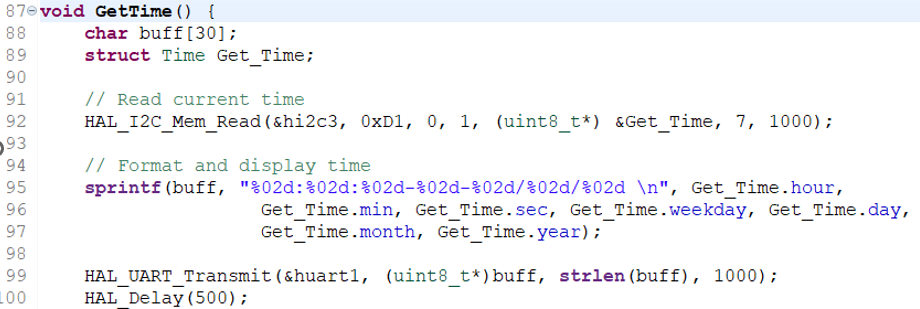
- [Video bài 3.6](https://drive.google.com/file/d/17yfrALwtPW38bUgHcFOPO8QUQHTPqdFF/view?usp=sharing)

- Hàm SetTIme:



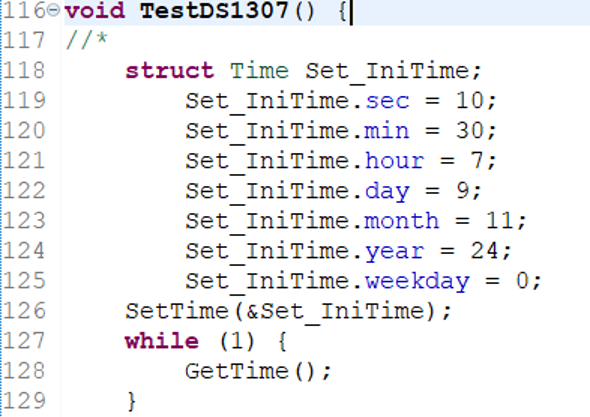
-> Trong hàm SetTime, HAL\_I2C\_Mem\_Write sẽ thực hiện ghi tuần tự 7 byte dữ liệu, tương ứng với các thông số: giây, phút, giờ, thứ, ngày, tháng và năm vào bộ nhớ của module Tiny RTC.

- Hàm GetTime:



-> Trong hàm GetTime, HAL\_I2C\_Mem\_Read sẽ đọc dữ liệu thời gian từ RTC và lưu vào biến Get\_time có kiểu dữ liệu Time được khai báo sẵn. Các thông số này sau đó được sao chép vào biến buff để truyền tới màn hình Hercules thông qua hàm HAL\_UART\_Transmit.

- Hàm Test:



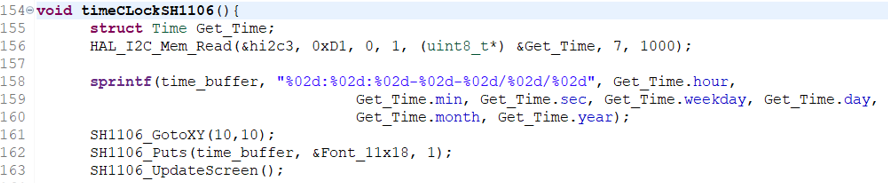
-> Hàm Test khởi tạo cứng thời gian cho RTC, liên tục hiển thị dữ liệu trên Hercules qua vòng lặp while.

- Cuối cùng gọi hàm Test trong hàm main() để chạy chương trình.

## **3.7. Lập trình ghép nối SH1106**

- [Video bài 3.7](https://drive.google.com/file/d/1nv294inMgoxiGAyHgxIVWHIv9PqfMKRP/view?usp=sharing)

- Hàm hiển thị thời gian trên SH1106:



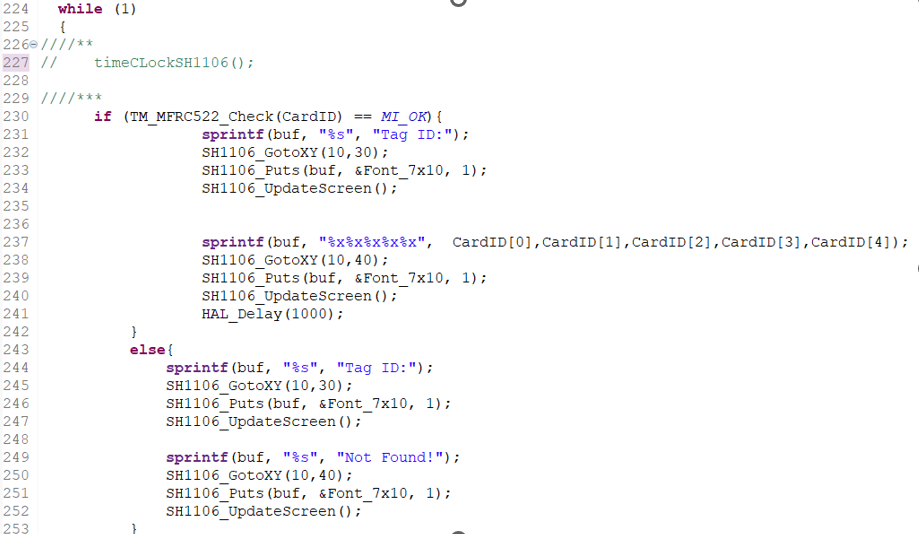
-> Hàm timeClockSH1106 dùng các hàm trong thư viện SH1106 để hiển thị thời gian lên màn hình OLED SH1106, trong đó, hàm SH1106\_GotoXY được sử dụng để đặt vị trí con trỏ vẽ tới tọa độ chỉ định (X, Y) = (10, 10). Sau khi xác định vị trí, SH1106\_Puts sẽ ghi nội dung từ biến vào bộ nhớ đệm của màn hình, nhưng dữ liệu vẫn chưa được hiển thị ngay lập tức. Cuối cùng, hàm SH1106\_UpdateScreen sẽ được gọi để cập nhật và đưa nội dung bộ đệm lên màn hình.

- Sau đó, gọi timeClockSH1106 trong vòng lặp while của hàm main() để chạy chương trình.

## **3.8. Lập trình với RC522**

- [Video bài 3.8](https://drive.google.com/file/d/1Y2JZimKPiWwDzjMbBxjGOT38LwhO8shV/view?usp=sharing)

- Khai báo các biến time\_buffer[], cardID[], buf[]. ta có hàm đọc mã thẻ RFID và hiện lên SH1106:



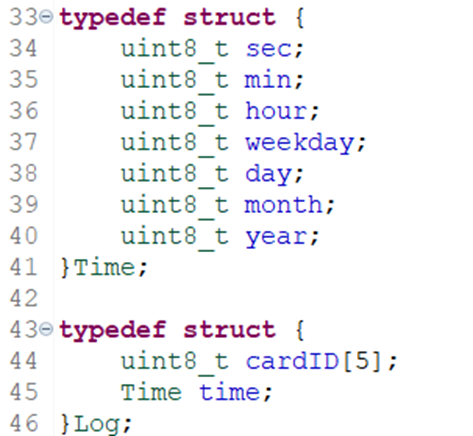
-> Trong vòng lặp while của hàm main, nếu thẻ RFID được nhận dạng thành công (khi hàm TM\_MFRC522\_Check return MI\_OK), màn hình SH1106 sẽ hiển thị dòng chữ "Tag ID:" tại tọa độ (10, 30) và tiếp theo là mã thẻ dưới dạng hexadecimal tại vị trí (10, 40). Ngược lại, nếu không quét được thẻ, màn hình sẽ hiển thị thông báo "Not Found!" tại cùng vị trí.

Việc hiển thị diễn ra qua ba bước: đầu tiên, SH1106\_GotoXY được sử dụng để xác định vị trí con trỏ trên màn hình; tiếp theo, SH1106\_Puts ghi dữ liệu cần hiển thị vào bộ nhớ đệm; và cuối cùng, SH1106\_UpdateScreen cập nhật nội dung từ bộ đệm lên màn hình OLED.

## **3.9. Bài tập tổng hợp**

- [Video bài 3.9](https://drive.google.com/file/d/1-Y3iblUBZmd_EJQgCfkP3OD5_4eqLj8i/view?usp=sharing)

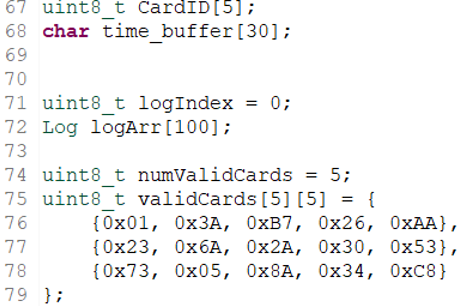
#### **3.9.1. Kiểu cấu trúc dùng trong bài**



- Time: Lưu trữ các thông tin liên quan đến thời gian do RTC cung cấp, bao gồm giây, phút, giờ, thứ, ngày, tháng và năm.

- Log: Dùng để ghi nhận thông tin của thẻ RFID cùng với dữ liệu thời gian, tạo thành một bản ghi đầy đủ cho mỗi lần quét thẻ. Trong đó, cardID lưu mã thẻ dưới dạng mảng gồm 5 byte, còn time lưu trữ thời điểm quét, lấy từ cấu trúc Time.

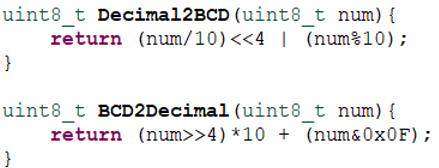
#### **3.9.2. Các biến sử dụng**



* CardID: dùng để tạm thời lưu mã thẻ RFID vừa quét, gồm 5 byte dữ liệu.
* time\_buffer: biến kiểu chuỗi ký tự, lưu thông tin thời gian cần hiển thị.
* logArr: mảng gồm 100 phần tử, mỗi phần tử chứa một bản ghi gồm mã thẻ và thời gian quét tương ứng, phục vụ việc lưu lịch sử quét thẻ.
* logIndex: biến chỉ số đóng vai trò như con trỏ, theo dõi vị trí hiện tại trong mảng logArr để ghi bản ghi mới.
* numValidCards: lưu tổng số thẻ hợp lệ hiện có, giúp việc kiểm tra và mở rộng danh sách validCards được thực hiện dễ dàng hơn.
* validCards: mảng chứa danh sách các thẻ hợp lệ đã được hệ thống định nghĩa trước, hỗ trợ tối đa 5 thẻ mẫu.

#### **3.9.3. Các hàm**

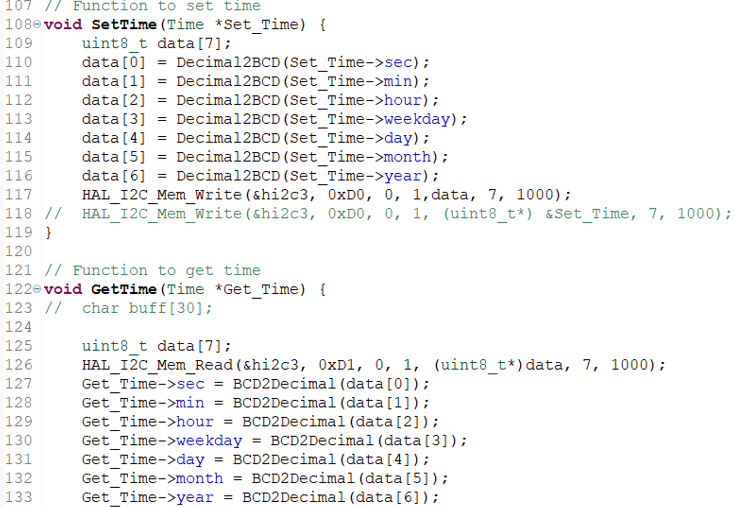
- Hàm chuyển thời gian từ BCD sang thập phân và ngược lại:



Hai hàm trên dùng để thực hiện việc chuyển đổi qua lại giữa số thập phân và định dạng BCD:

* Hàm Decimal2BCD: nhận một số thập phân đầu vào, tách thành chữ số hàng chục và hàng đơn vị. Hàng chục sẽ được dịch trái 4 bit để nằm ở nửa cao của byte, sau đó kết hợp với hàng đơn vị ở nửa thấp thông qua toán tử OR, tạo thành một giá trị BCD.
* Hàm BCD2Decimal: thực hiện việc chuyển BCD về số thập phân. Hàm lấy 4 bit cao (hàng chục) bằng phép dịch phải, nhân với 10 rồi cộng thêm 4 bit thấp (hàng đơn vị) được tách ra bằng phép AND với 0x0F.

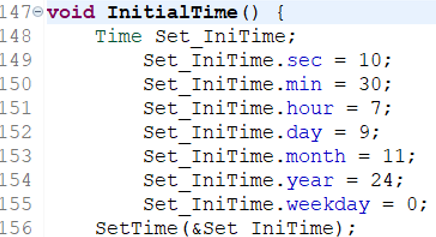
- Hàm thiết lập thời gian và lấy giá trị thời gian:



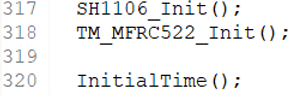
Đây là hai hàm thực hiện các thao tác cơ bản với module Tiny RTC:

* Hàm SetTime: nhận tham số là con trỏ tới một biến kiểu Time, sau đó chuyển từng thành phần thời gian sang định dạng BCD. Các giá trị này được ghi vào RTC bằng hàm HAL\_I2C\_Mem\_Write với địa chỉ 0xD0 (chế độ ghi dữ liệu).
* Hàm GetTime: thực hiện thao tác ngược lại, tức là đọc 7 byte dữ liệu từ RTC tại địa chỉ 0xD1 (chế độ đọc), sau đó chuyển đổi các giá trị từ BCD về dạng thập phân và lưu vào cấu trúc Time.

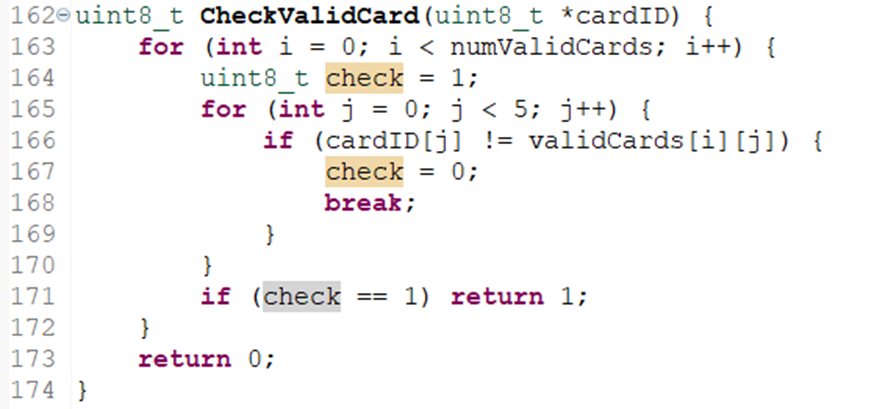
- Hàm khởi tạo giá trị thời gian cứng cho RTC:



Gọi trong main để khởi tạo:



- Hàm kiểm tra thẻ:



* Hàm CheckValidCard() có nhiệm vụ kiểm tra xem mã thẻ RFID đầu vào (cardID) có thuộc danh sách các thẻ hợp lệ (validCards) hay không. Hàm nhận vào một mảng gồm 5 byte.
* Cách hoạt động: sử dụng hai vòng lặp lồng nhau — vòng ngoài quét từng thẻ trong validCards, vòng trong so sánh từng byte giữa thẻ đang xét và cardID. Nếu phát hiện byte không trùng khớp, vòng lặp lập tức dừng và chuyển sang thẻ kế tiếp. Nếu tìm được một thẻ hoàn toàn trùng khớp, hàm lập tức return 1 (hợp lệ). Nếu không tìm thấy sau khi duyệt hết, hàm sẽ return 0 (không hợp lệ).

| void CheckCard()  {  char buf[100];  uint8\_t CardID[5];  if (TM\_MFRC522\_Check(CardID) == MI\_OK) {  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, GPIO\_PIN\_SET);  if (CheckValidCard(CardID)) {  //WELCOME  sprintf(buf, "%s", "WELCOME");  SH1106\_GotoXY(12, 10);  SH1106\_Puts(buf, &Font\_11x18, 1);  SH1106\_UpdateScreen();  //Log time  Time currentTime;  GetTime(&currentTime);  if (logIndex < 100) {  memcpy(logArr[logIndex].cardID, CardID, 5);  logArr[logIndex].time = currentTime;  logIndex++;  }  //Set led 14  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_14, GPIO\_PIN\_SET);  } else {  sprintf(buf, "%s", "REJECTED");  SH1106\_GotoXY(12, 10);  SH1106\_Puts(buf, &Font\_11x18, 1);  SH1106\_UpdateScreen();  }  //Delay until card is out  while (TM\_MFRC522\_Check(CardID) == MI\_OK) {  HAL\_Delay(100);  }  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_14, GPIO\_PIN\_RESET);  } else {  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, GPIO\_PIN\_RESET);  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_14, GPIO\_PIN\_RESET);  sprintf(buf, "%s", "-------------");  SH1106\_GotoXY(12, 10);  SH1106\_Puts(buf, &Font\_11x18, 1);  SH1106\_UpdateScreen();  }  } |
| --- |

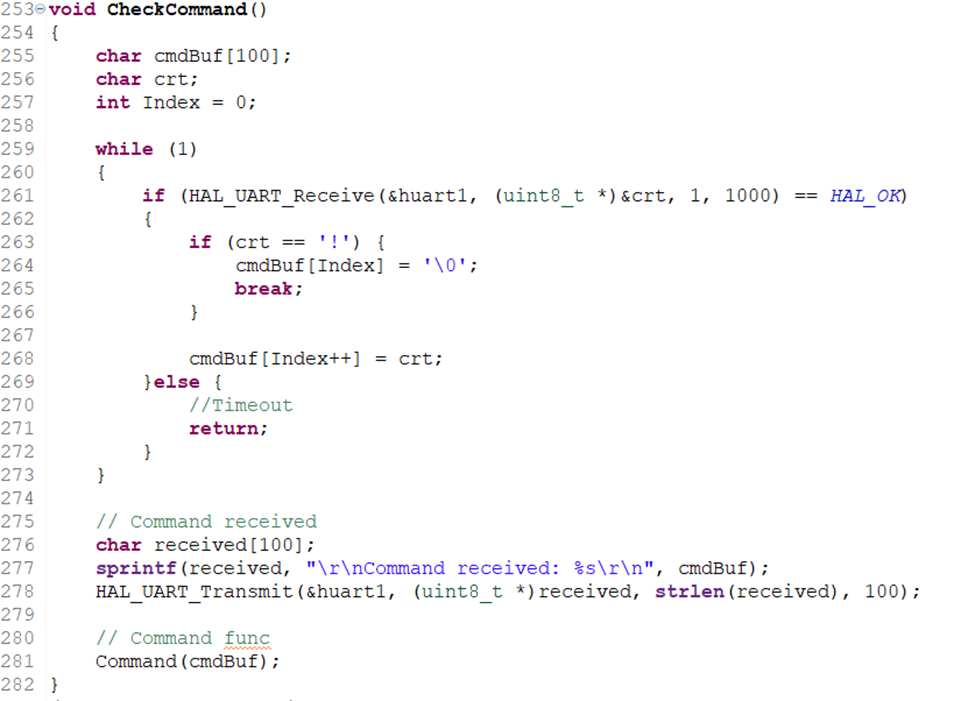
* Hàm CheckCard() có vai trò nhận diện và xử lý thẻ RFID mỗi khi thẻ được đưa vào vùng quét. Khi phát hiện thẻ, chương trình sẽ kiểm tra tính hợp lệ của thẻ, phản hồi kết quả thông qua đèn LED và màn hình hiển thị, đồng thời lưu lại thông tin nếu thẻ được chấp nhận.
* Cụ thể, hàm bắt đầu bằng việc kiểm tra sự xuất hiện của thẻ bằng TM\_MFRC522\_Check(). Nếu có thẻ, hệ thống sẽ bật đèn LED G13 và thực hiện xác thực với CheckValidCard().
* Nếu thẻ hợp lệ, thông báo "WELCOME" sẽ được hiển thị trên màn hình OLED, đèn LED G14 sẽ sáng lên và dữ liệu thẻ cùng thời gian quét sẽ được ghi vào logArr.
* Ngược lại màn hình sẽ hiển thị thông báo "REJECTED".

Sau khi xử lý, hệ thống sẽ đợi đến khi thẻ được đưa ra khỏi vùng quét rồi mới tắt các đèn LED. Nếu không có thẻ quét, màn hình sẽ hiển thị chuỗi mặc định "------------".

- Hàm kiểm tra command:

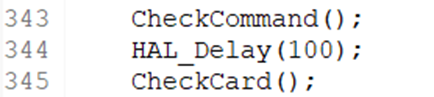
| void Command(char \*cmd)  {  if (strncmp(cmd, "VIEW", 4) == 0)  {  char buf[100];  for (int i = 0; i < logIndex; i++) {  sprintf(buf,"UID: %02X%02X%02X%02X%02X - %02d:%02d:%02d %02d/%02d/20%02d\r\n",  logArr[i].cardID[0], logArr[i].cardID[1], logArr[i].cardID[2],logArr[i].cardID[3], logArr[i].cardID[4],  logArr[i].time.hour, logArr[i].time.min,logArr[i].time.sec, logArr[i].time.day,logArr[i].time.month, logArr[i].time.year);  HAL\_UART\_Transmit(&huart1, (uint8\_t \*)buf, strlen(buf), 100);  }  } |
| --- |

* Hàm Command() hoạt động như một cổng giao tiếp lệnh giữa hệ thống và người dùng thông qua UART. Trong đoạn code này, hàm nhận lệnh "VIEW" để yêu cầu hiển thị toàn bộ lịch sử quét thẻ RFID đã ghi nhận.
* Cụ thể, Command() sử dụng strncmp() để kiểm tra nếu lệnh người dùng nhập vào là "VIEW". Nếu đúng, chương trình sẽ duyệt qua từng bản ghi trong mảng logArr, định dạng thông tin mã thẻ (dạng hexa) và thời gian quét thành chuỗi, rồi truyền lần lượt từng chuỗi này qua UART bằng HAL\_UART\_Transmit(). Biến buf đảm nhận vai trò vùng đệm tạm để chứa nội dung trước khi gửi đi.



* Hàm CheckCommand() đóng vai trò là bộ thu lệnh từ UART, giúp hệ thống giao tiếp trực tiếp với người dùng. Hàm liên tục đọc từng ký tự được gửi đến bằng HAL\_UART\_Receive() và lưu vào bộ đệm cmdBuf. Khi phát hiện ký tự kết thúc '!', hàm đóng chuỗi, xác nhận đã nhận lệnh bằng cách gửi lại qua UART, sau đó gọi Command(cmdBuf) để xử lý lệnh vừa nhận.
* Nếu quá 1000ms không nhận được ký tự nào (timeout), hàm sẽ tự động dừng để tránh treo hệ thống.

- Cuối cùng gọi các hàm trong main:

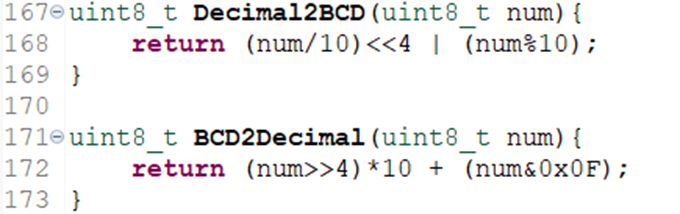


-> Chương trình vận hành bằng cách thay phiên thực hiện hai nhiệm vụ là tiếp nhận lệnh từ người dùng thông qua UART và kiểm tra, xác thực thẻ RFID bằng hàm CheckCard(). Sau mỗi lần thực hiện một tác vụ, hệ thống sẽ tạm dừng 100ms để nghỉ nhằm duy trì hoạt động ổn định giữa các vòng lặp.

## **3.7.\*. Bài bonus: BCD Timer (Source trong file zip bai3.6\_3.7\_3.8.zip)**

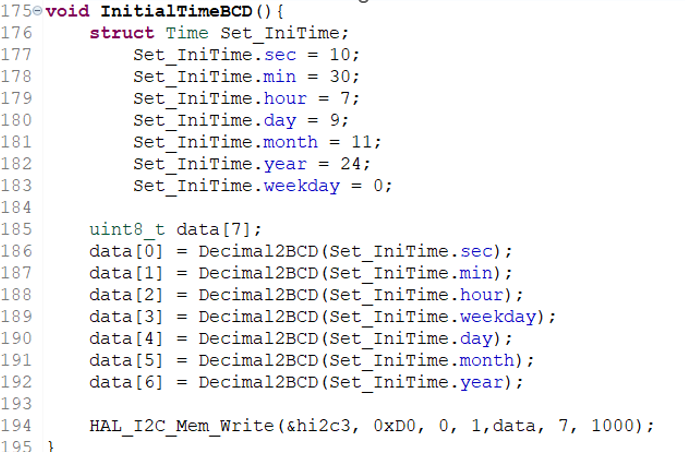
- [Video bài bonus](https://drive.google.com/file/d/18I3mUBMAWPgJ_xlZ-ALFDAd2hAl1WIOl/view?usp=sharing)

- Hàm chuyển đổi thời gian từ dạng BCD sang thập phân và ngược lại:



Hai hàm này dùng để chuyển đổi qua lại giữa số thập phân thông thường và dạng BCD, tương tự như cách đã trình bày trong bài 3.9.

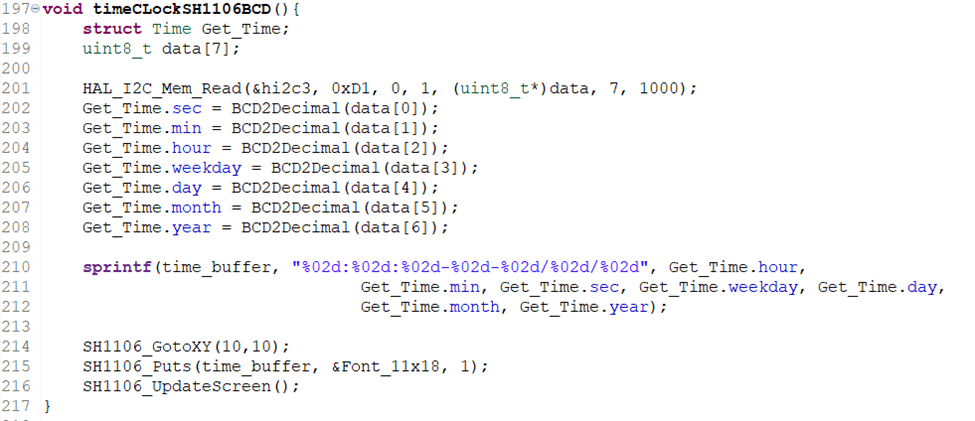
- Hàm thiết lập cứng thời gian:





* Hàm InitialTimeBCD() dùng để thiết lập giá trị thời gian mặc định ban đầu cho hệ thống.
* Đầu tiên, hàm khai báo một biến cấu trúc Set\_IniTime và gán các giá trị thời gian cụ thể dưới dạng thập phân.
* Sau đó, các thành phần thời gian này được chuyển đổi từ định dạng thập phân sang định dạng BCD bằng cách sử dụng hàm Decimal2BCD(), rồi lưu trữ vào mảng data[7]. Mỗi phần tử trong mảng tương ứng với một thành phần thời gian.
* Cuối cùng, toàn bộ mảng dữ liệu này được ghi vào bộ nhớ của module RTC thông qua giao thức I2C, bằng hàm HAL\_I2C\_Mem\_Write().
* Gọi hàm khởi tạo này trong hàm main() trước khi chạy chương trình.

- Hàm hiển thị thời gian đã chuyển đổi BCD lên SH1106:





* Hàm timeClockSH1106BCD() đảm nhận nhiệm vụ đọc thời gian thực từ module RTC và hiển thị nó lên màn hình OLED SH1106.
* Cụ thể, hàm sử dụng giao tiếp I2C để đọc 7 byte dữ liệu thời gian từ RTC (sử dụng địa chỉ 0xD1 ở chế độ đọc). Các giá trị thời gian nhận được ở định dạng BCD sẽ được chuyển đổi sang dạng thập phân thông qua hàm BCD2Decimal(), sau đó lưu vào biến cấu trúc Get\_Time.
* Tiếp theo, hàm định dạng lại các thành phần thời gian thành chuỗi ký tự hoàn chỉnh bằng sprintf(), lưu vào bộ đệm time\_buffer.
* Cuối cùng, nội dung thời gian được hiển thị lên màn hình OLED tại vị trí tọa độ (10,10) bằng cách sử dụng các hàm SH1106\_GotoXY(), SH1106\_Puts(), và SH1106\_UpdateScreen().

**~THE END~**