THIẾT KẾ MÔ HÌNH ĐIỀU KHIỂN XE TỪ XA

THÔNG QUA GIAO THỨC RF VÀ BLUETOOTH

Danh sách sinh viên thực hiện

Họ và tên: **Mã số sinh viên**

▪

1. Giới thiệu (2đ)

1.1 Mục đích đề tài:

- Nhằm ôn lại kiến thức đã học trên lớp để thiết kế một mô hình xe có thể điều khiển thông qua hai giao thức là RF và Bluetooth.
- Nhằm tìm hiểu các ngoại vi thông dụng khác của vi điều khiển STM32F103C8T6 như UART, SPI, ADC…. Đồng thời, tìm hiểu thêm cách thức truyền thông không dây cơ bản cho một hệ thống nhúng.
- Nhằm thực hiện mô hình hóa ý tưởng điều khiển xe ô tô thông qua các giao thức chuẩn cơ bản. Cũng như sau này, là nền tảng để bắt đầu trong lĩnh vực Automotive.

1.2 Ý nghĩa đề tài:

- Giúp cho việc hiểu cơ bản về vi điều khiển của hãng ST bằng cách điều khiển các thành phần đơn như điều khiển động cơ, lập trình nhiều trạng thái, thiết lập truyền thông không giây.
- Tạo mô hình khái quát chung để chúng ta có thể hình dung về việc thiết kế cho một hệ thống nhúng, bản chất chỉ là những phần đơn giản lắp ghép lại sao cho phù hợp.
- Giúp các thành viên trong nhóm có cơ hội tìm hiểu các chuẩn giao thức không dây cơ bản và lập trình phần cứng cho hệ thống.

2. Cơ sở lý thuyết (2đ)

2.1 Hệ thống xung clock của vi điều khiển

- Cách cấu hình xung Clock trên STMCubeMX.
- Cách chia tần số xung Clock khi cấu hình xuất xung PWM (tần số được sử dụng trong đề tài là 1 kHz).
- Cách thức hoạt động của ngắt System timer 24 bit đếm xuống.

2.2 Hệ thống ngoại vi của vi điều khiển

- Cách cấu hình Port/Pin vào ra trên STMCubeMX
- Cách thức hoạt động của các thanh ghi cấu hình GPIO như thanh ghi CRL, CRH, BSRR…
- Cách đọc các trạng thái phím nhấn cũng như mức logic ở các ngõ vào/ra sao cho phù hợp với mức logic của vi điều khiển.

2.3 Hệ thống truyền thông

- Hiểu cơ bản cách hoạt động của các bộ UART, SPI để cấu hình và sử dụng các hàm từ thư viện HAL

- Cách thức truyền nhận một chuỗi và xử lí chuỗi để thực hiện các công việc điều khiển.

2.4 Hệ thống ngoại vi Analog/Digital Convert

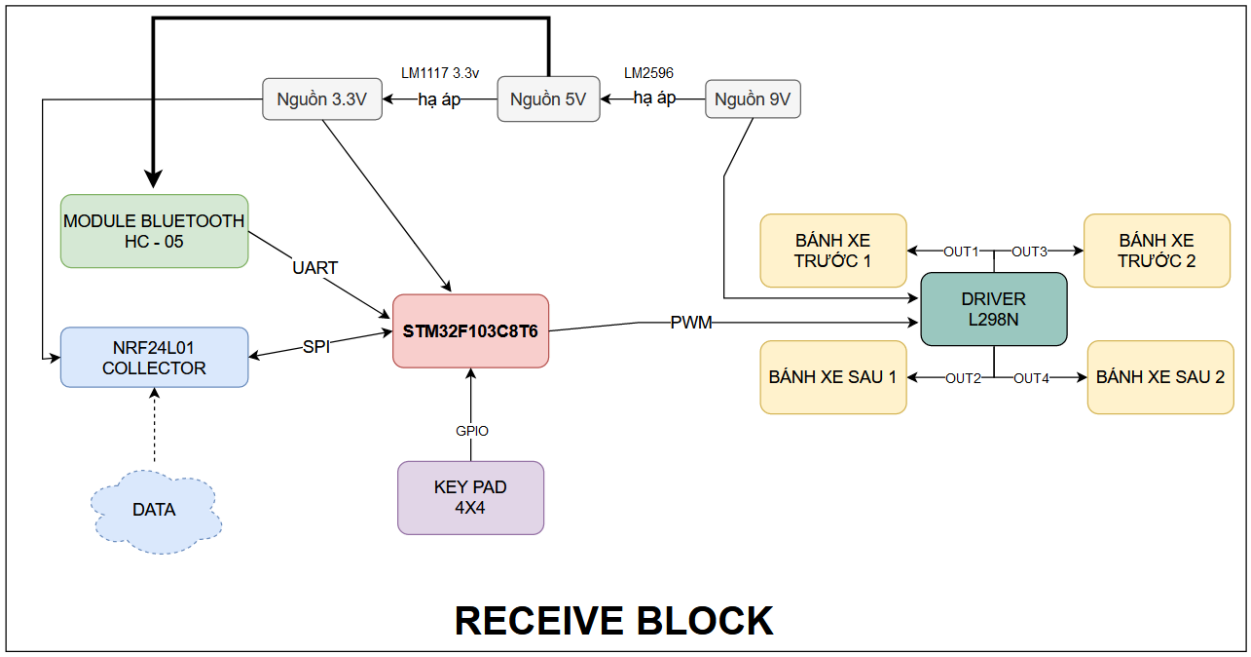
- Hiểu được cách hoạt động của bộ ADC và các cách đọc từng kênh.
- Thực hiện mapping giá trị vào các ngưỡng.

2.5 Hệ điều hành nhúng Embedded Operating System

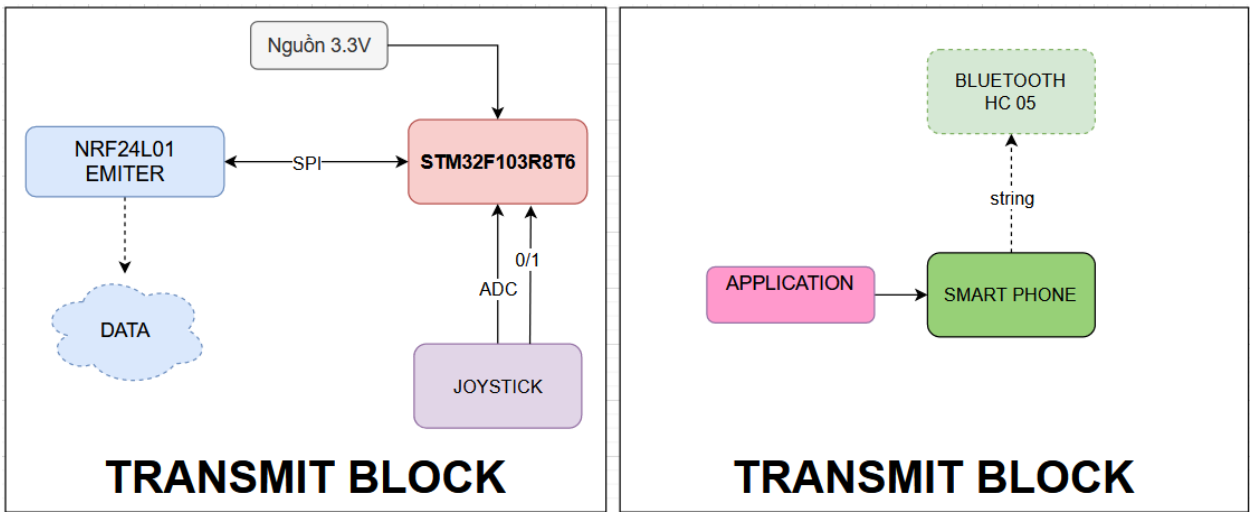
- Hiểu cách cấu hình và chọn giá trị khoảng thời gian cho các tiến trình(TASK)

3. Thiết kế phần cứng (2đ)

3.1 Sơ đồ khối:



Hình 1: Sơ đồ khối của khối nhận sử dụng STM32F103C8T6



Hình 2: Sơ đồ khối của hai khối truyền dữ liệu sử dụng STM32F103R8T6

Giải thích sơ đồ khối: gồm có 2 khối chính là khối nhận và khối truyền dữ liệu

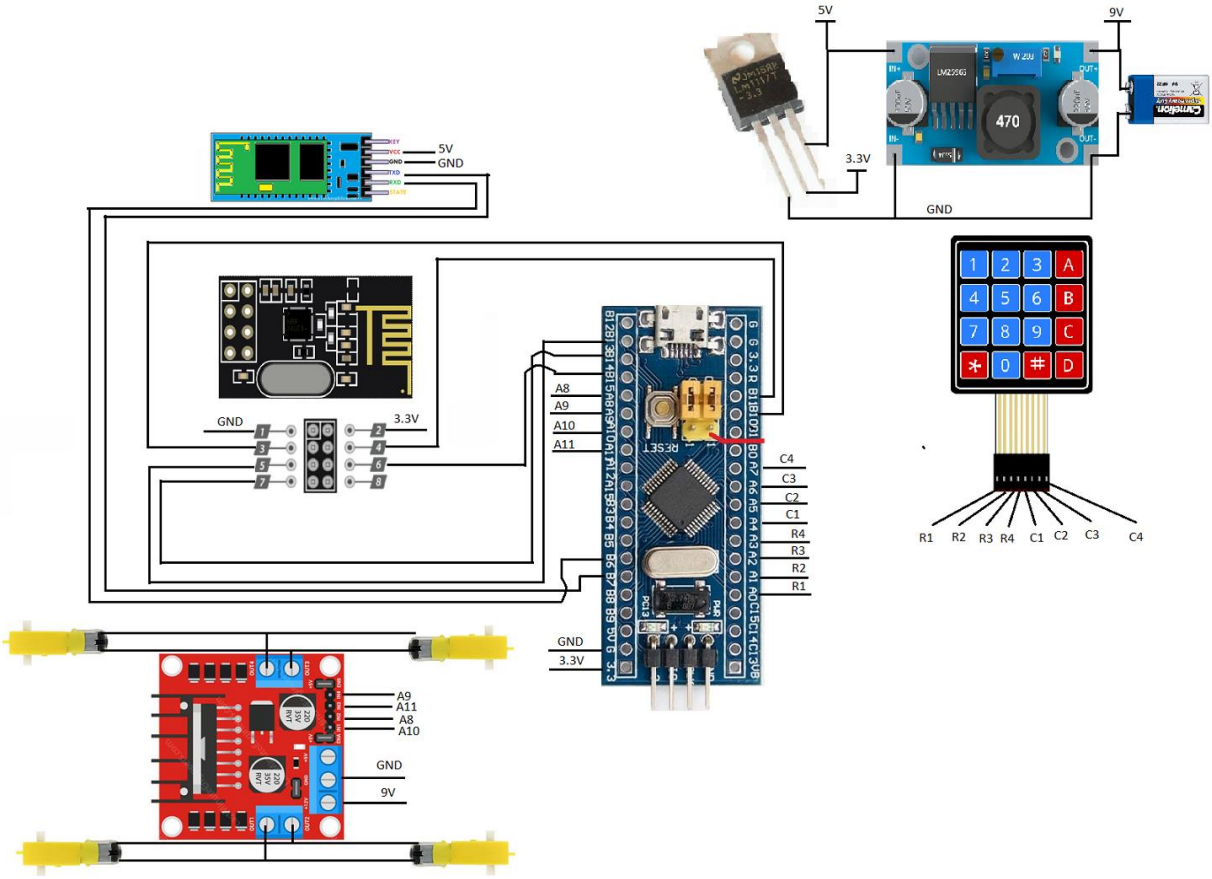
3.1.1 Khối nhận và xử lí tín hiệu:

- Khối nguồn: Sử dụng nguồn pin 9V để cấp cho 4 động cơ giảm tốc, tiếp tục hạ áp từ 9V xuống 5V để cấp cho module Bluetooth HC 05, tiếp tục thực hiện hạ áp từ 5V xuống 3.3V đê cấp cho vi điều khiển STM32F103C8T6 và module NRF24L01.
- Khối Vi điều khiển STM32F103C8T6: Vi điều khiển thực hiện nhận tín hiệu không dây và xử lí các tiến trình như kiểm tra KEYPAD, phân tích chuỗi kí tự, định thời chương trình, băm xung PWM…
- Khối Blutooth HC05: nhận chuỗi tín hiệu từ ứng dụng điều khiển trên điện thoại và thực hiện gửi vào vi điều khiển thông qua giao thức UART.
- Khối NRF24L01: nhận tín hiệu từ khối truyền và thực hiện gửi chuỗi kí tự thông qua giao thức SPI.
- Khối Driver L298N: nhận tín hiệu xung PWM từ vi điều khiển để cho ra dải điện áp khác nhau làm thay đổi tốc độ động cơ.

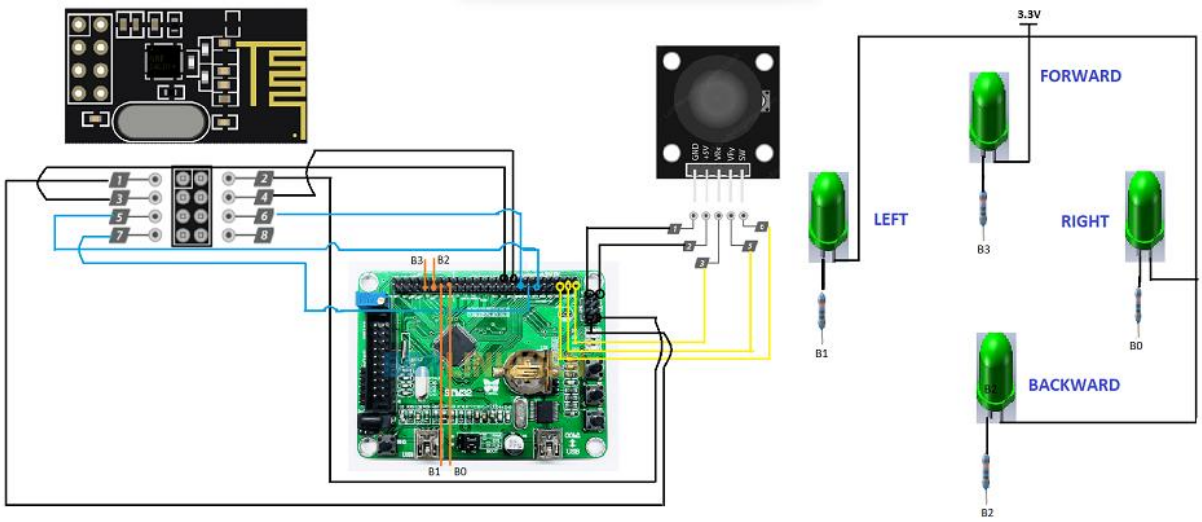
3.1.2 Khối truyền tín hiệu:

- Khối nguồn: nguồn cấp 3.3V lấy từ USB mạch nạp ST-Link V2.
- Khối vi điều khiển STM32F103R8T6: Vi điều khiển thực hiện gửi chuỗi kí tự điều hướng dựa vào tiến trình xử lí Joystick. Joystick sẽ được đọc tín hiệu vào MCU thông qua kênh ADC 12 bit và một nút nhấn tích hợp sẵn trên Joystick để thực hiện chuyển đổi qua lại giữa 2 sóng Blutooth và RF.
- Khối Smartphone: được cài sẵn ứng dụng Bluetooth RC Controller. Khi kết nối với thành công với điện thoại sẽ bấm các nút điều hướng trên giao diện để gửi chuỗi kí tự xuống HC 05.

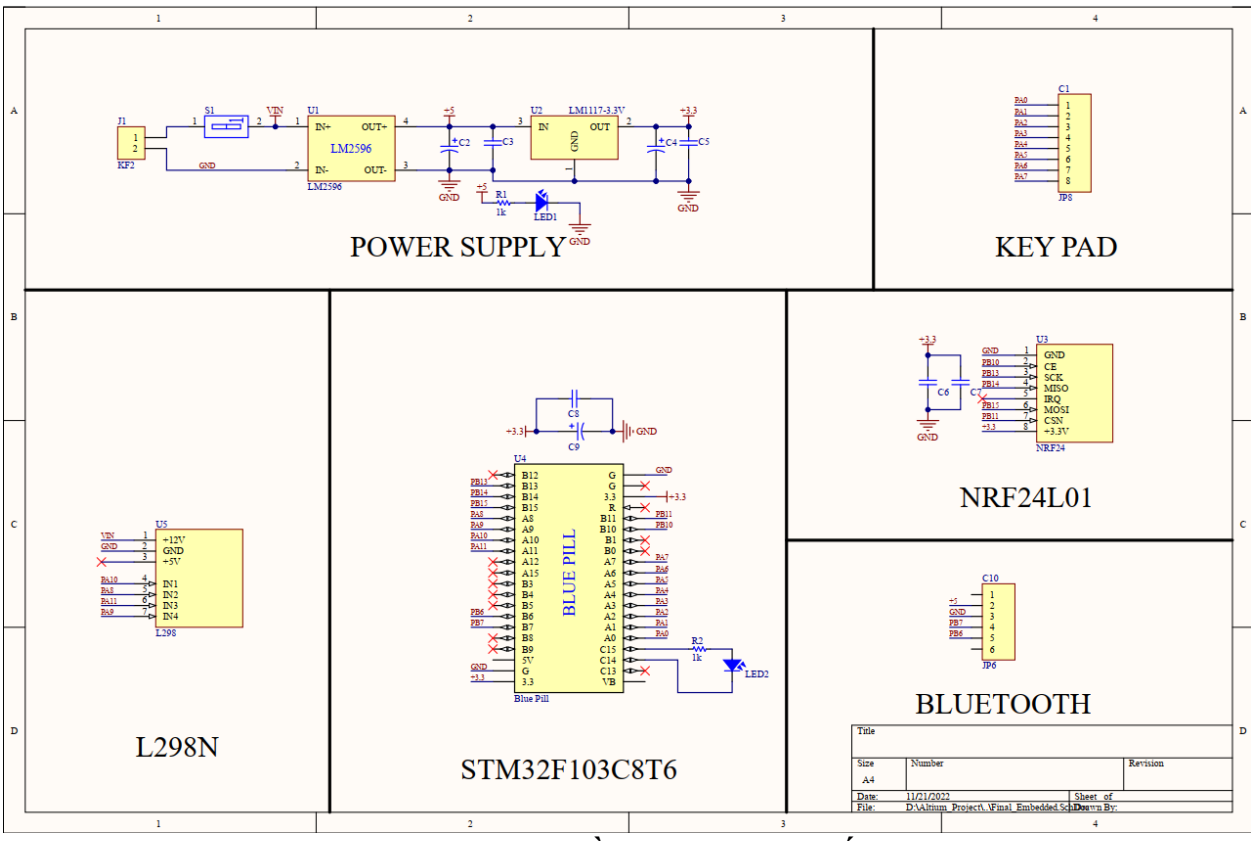
3.2 Sơ đồ chi tiết phần cứng:



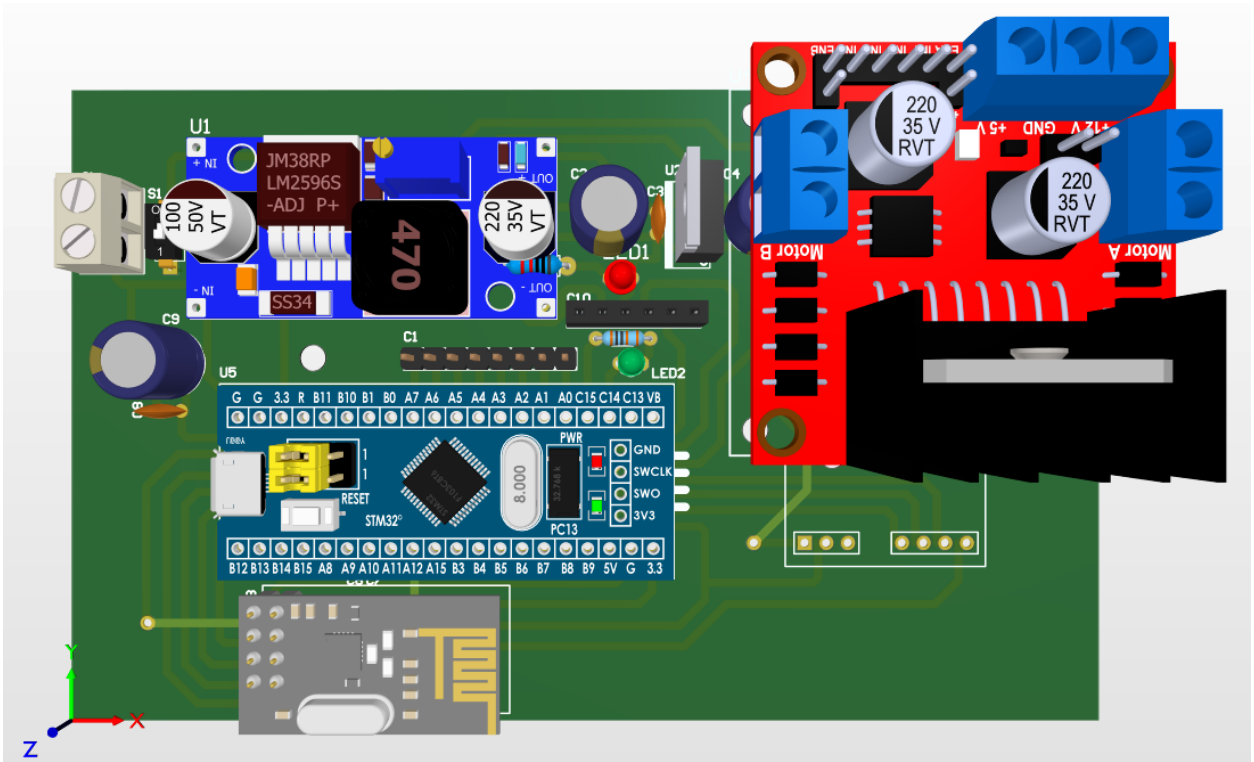
Hình 3: Sơ đồ phần cứng của khối nhận



Hình 4: Sơ đồ phần cứng của khối truyền



Hình 5: Sơ đồ nguyên lí của khối nhận



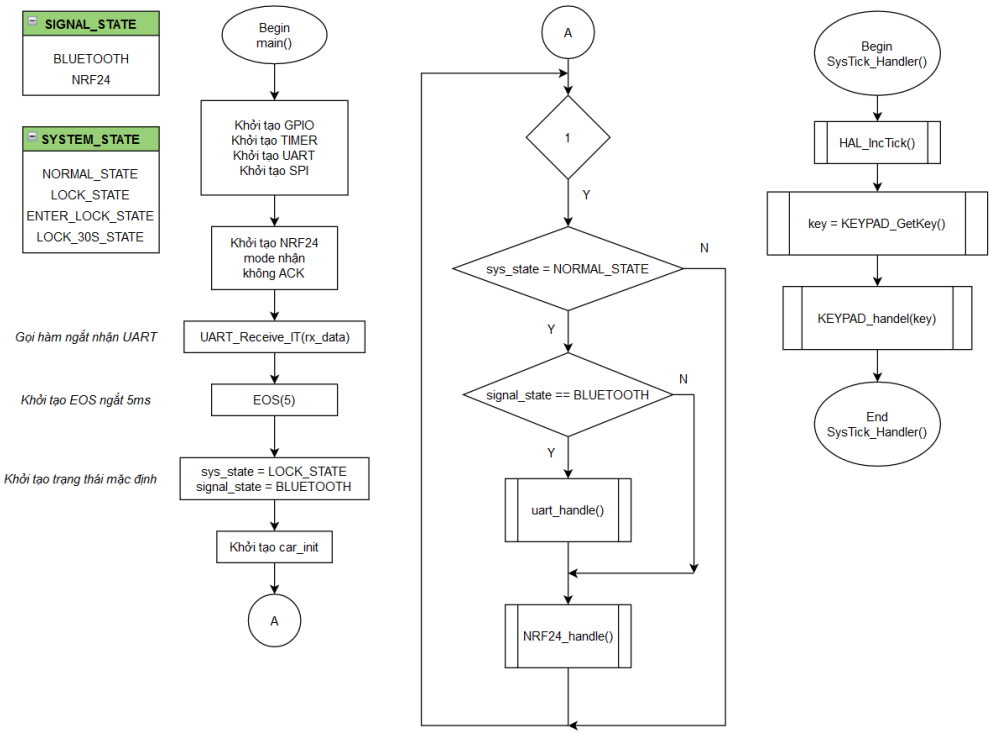
Hình 6: Mạch phay PCB của khối nhận

Giải thích sơ đồ nguyên lí:

- Khối nguồn cấp vào: sử dụng 2 pin Li-on 4.2V cấp nguồn cho driver L298. Qua mạch hạ áp LM2596 còn 5V để cấp cho Bluetooth HC 05. Tiếp tục hạ áp qua LM1117-3.3V để cấp cho vi điều khiển và NRF24L01.
- Khối vi điều khiển STM32F103C8T6:
  + Các chân A0 – A3(Output), A4 - A7(Input) dùng cho chức năng quét KEYPAD 4x4
  + Các chân A8 nối với IN2, A9 nối với IN4, A10 nối với IN1, A11 nối với IN3 trên driver L298.
  + Chân A7 nối với Rx, chân A6 nối với Tx của moudule Bluetooth HC 05.
  + Chân B10 nối với chân CE, B11 nối với chân CSN, B13 nối với chân SCK, B14 nối với chân MISO, chân B15 nối với chân MOSI của module NRF24L01.
  + Chân C14 và C15 mắc nối tiếp với 1 LED và 1 điện trở để tạo tín hiệu khi cần.

4. Thiết kế phần mềm (2đ)

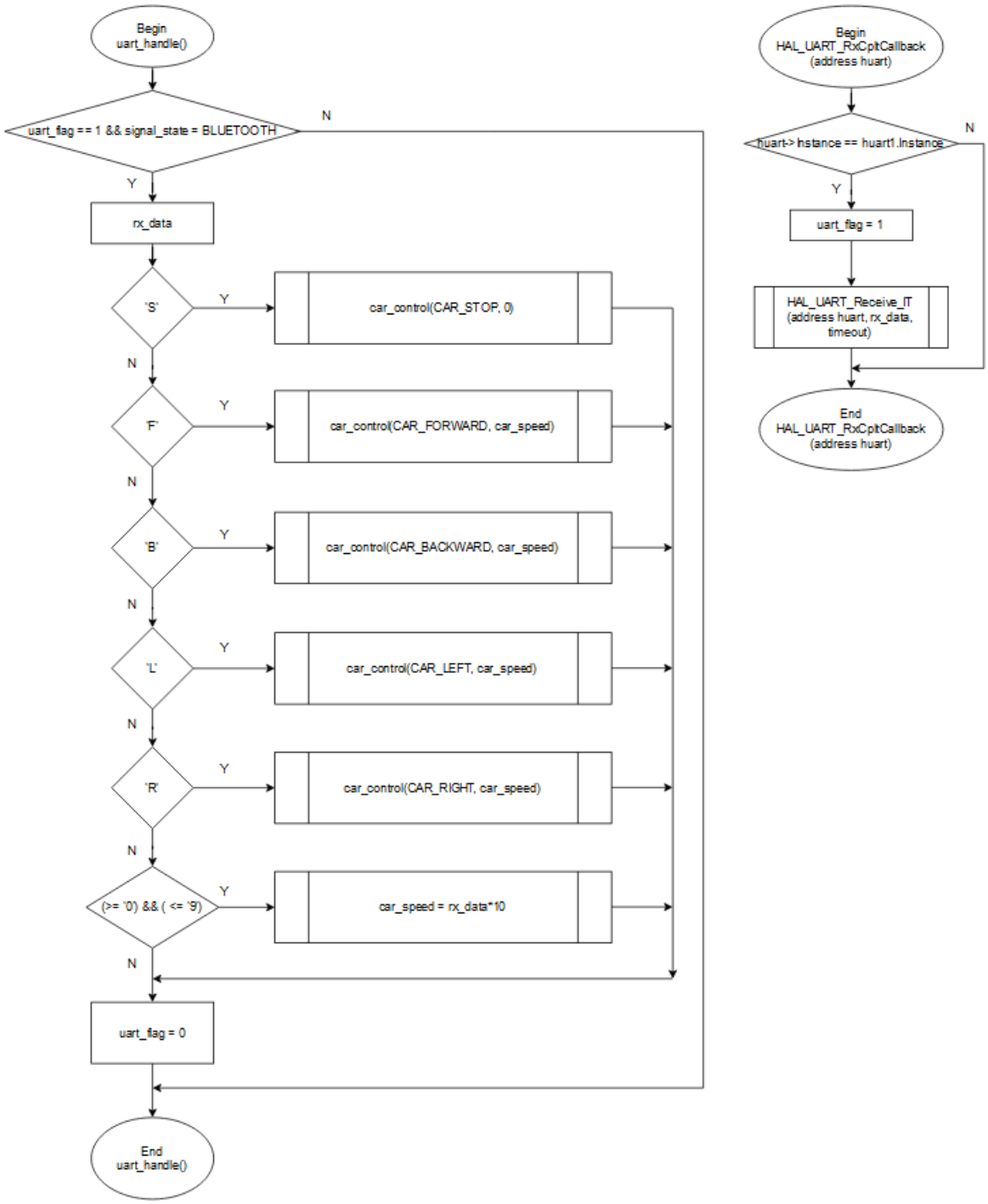
4.1 Lưu đồ chương trình trạm thu – Receive Blcok:



Hình 7: Lưu đồ chương trình chính trong file main.c (1)

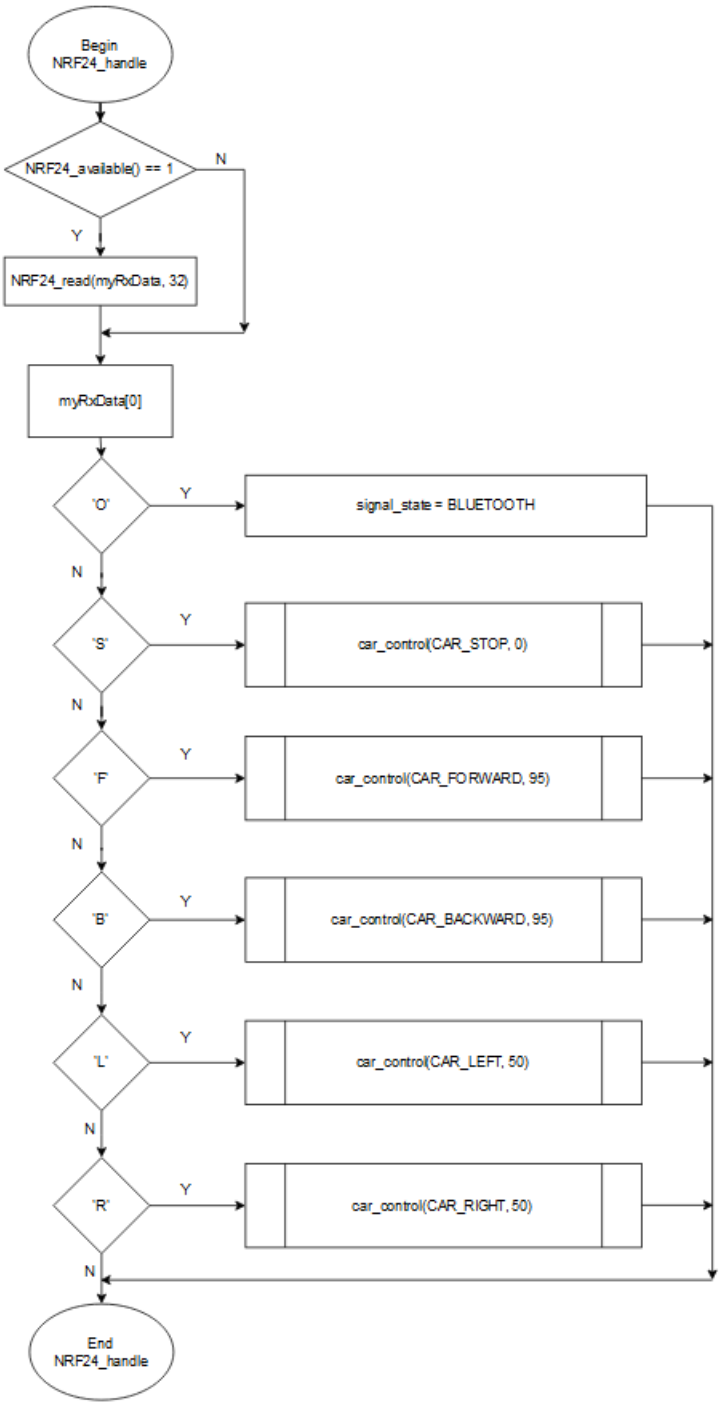
Giải thích: Hệ thống sẽ chờ được mở khóa(NORMAL_STATE) để tiếp nhận các tín hiệu điều khiển. Ngoài ra hệ thống còn khởi tạo hệ điều hành nhúng EOS với chu kỳ ngắt 5ms để thực hiện quét KEYPAD và các hàm thực thi KEYPAD.

Mặc định hệ thống sẽ được điều khiển thông qua tín hiệu Bluetooth. Nếu nút nhấn trên Transmit Block được nhấn thì sẽ chuyển sang điều khiển tín hiệu NRF24 và ngược lại.



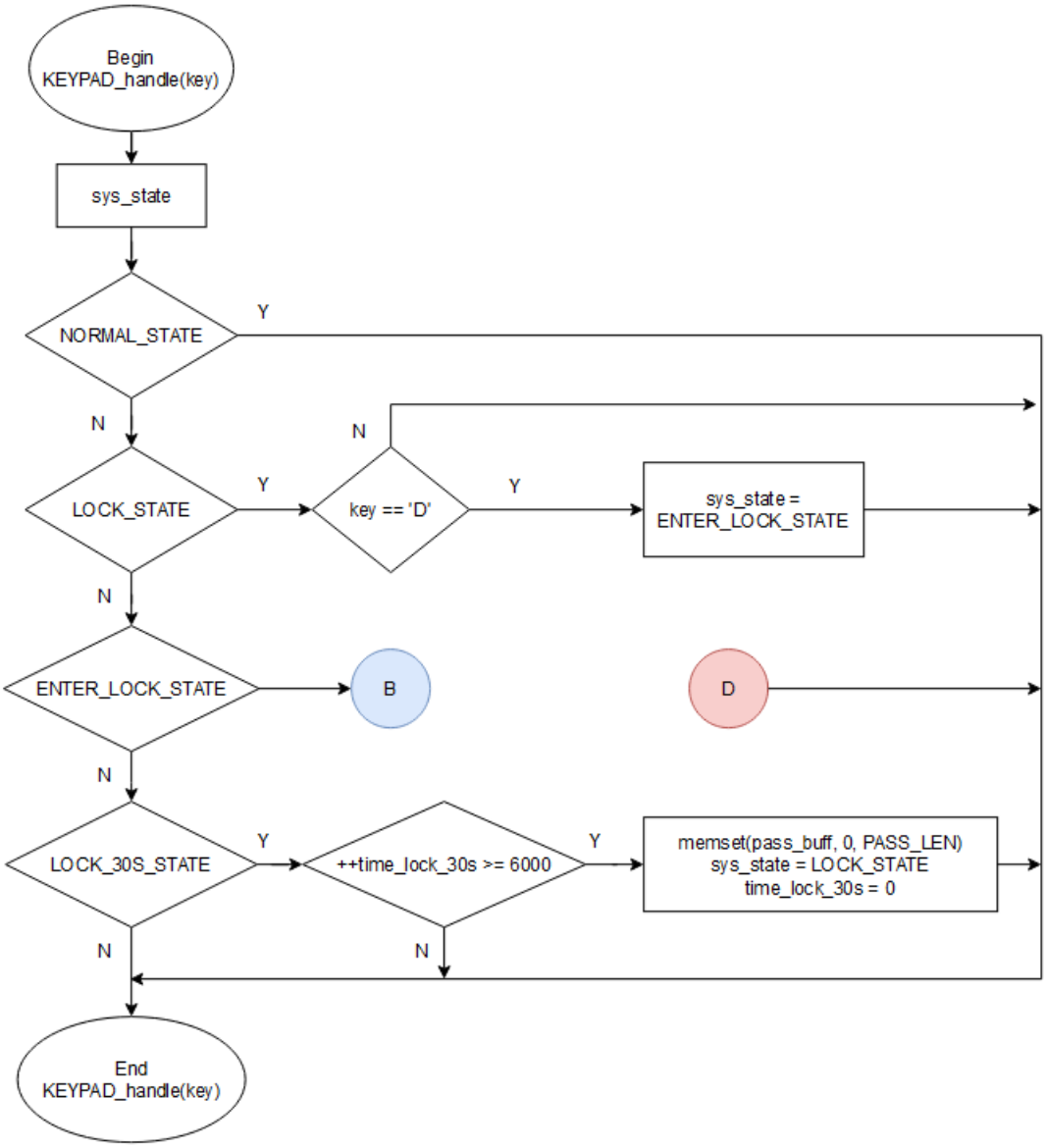
Hình 8: Lưu đồ chương trình con trong file main.c (2)

Giải thích: Chương trình này sẽ kiểm tra biến cờ(uart_flag) nhận dữ liệu và có đang ở trong trạng thái tín hiệu điều khiển bằng Bluetooth không. Nếu cả 2 đúng thì thực hiện giải mã dữ liệu từ UART gửi đến. Hình bên trái là hàm ngắt nhận UART khi có kí tự gửi đến sẽ xảy ra ngắt, gán vào biến rx_data và đặt cờ uart_flag = 1.



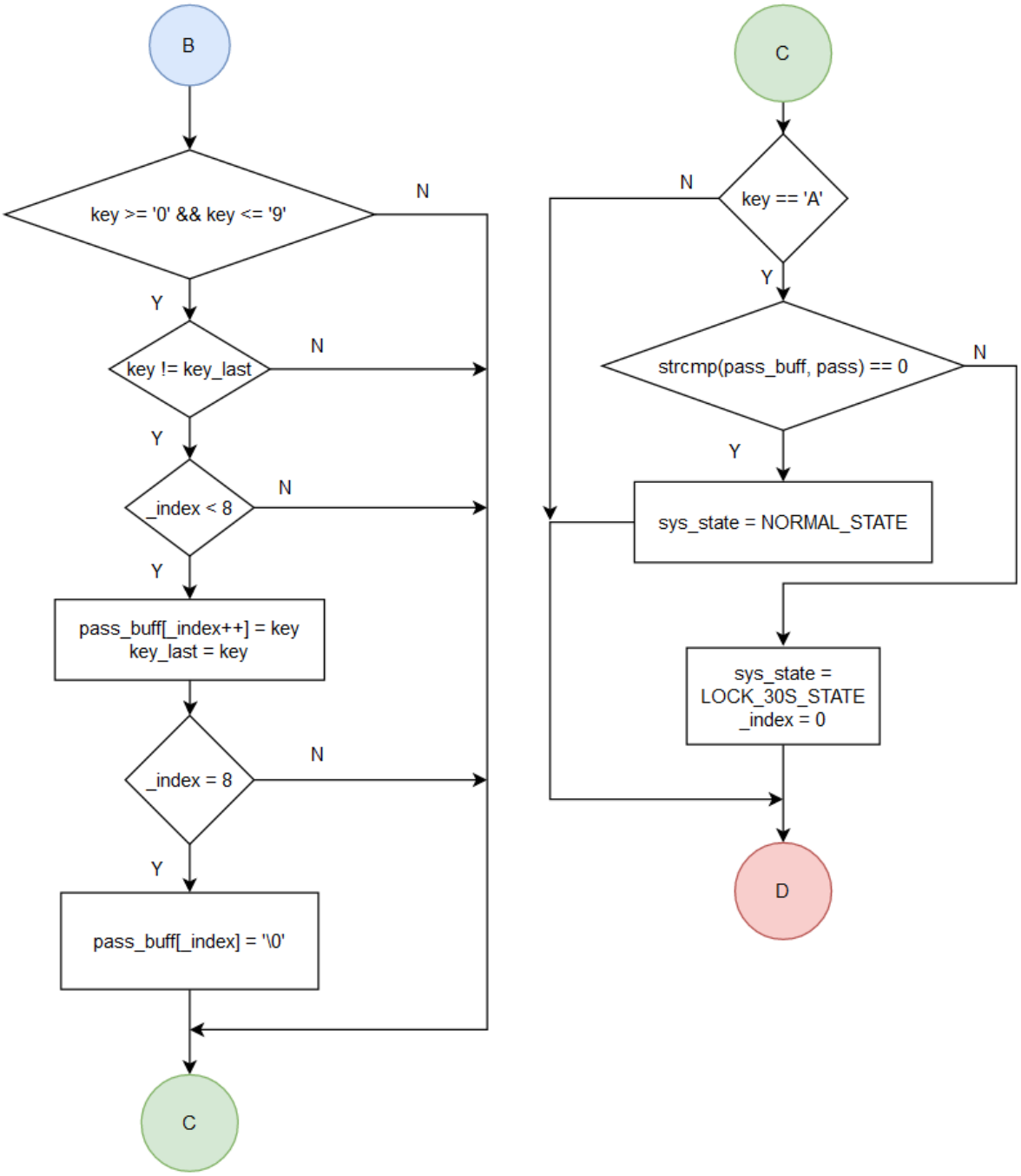
Hình 9: Lưu đồ chương trình con trong file main.c (3)

Giải thích: Chương trình này sẽ lắng nghe có tín hiệu NRF24 gửi đến hay không (hàm NRF24_available()), nếu có tín hiệu thì sẽ thực hiện giải mã kí tự đã nhận để điều khiển xe. Do khối Transmit Block đã cấu hình gửi 1 kí tự nên chỉ cần sử dụng dữ liệu mảng đầu tiên để thực hiện điều khiển. Kí tự 'O' để hệ thống bật chế độ điều khiển Bluetooth.



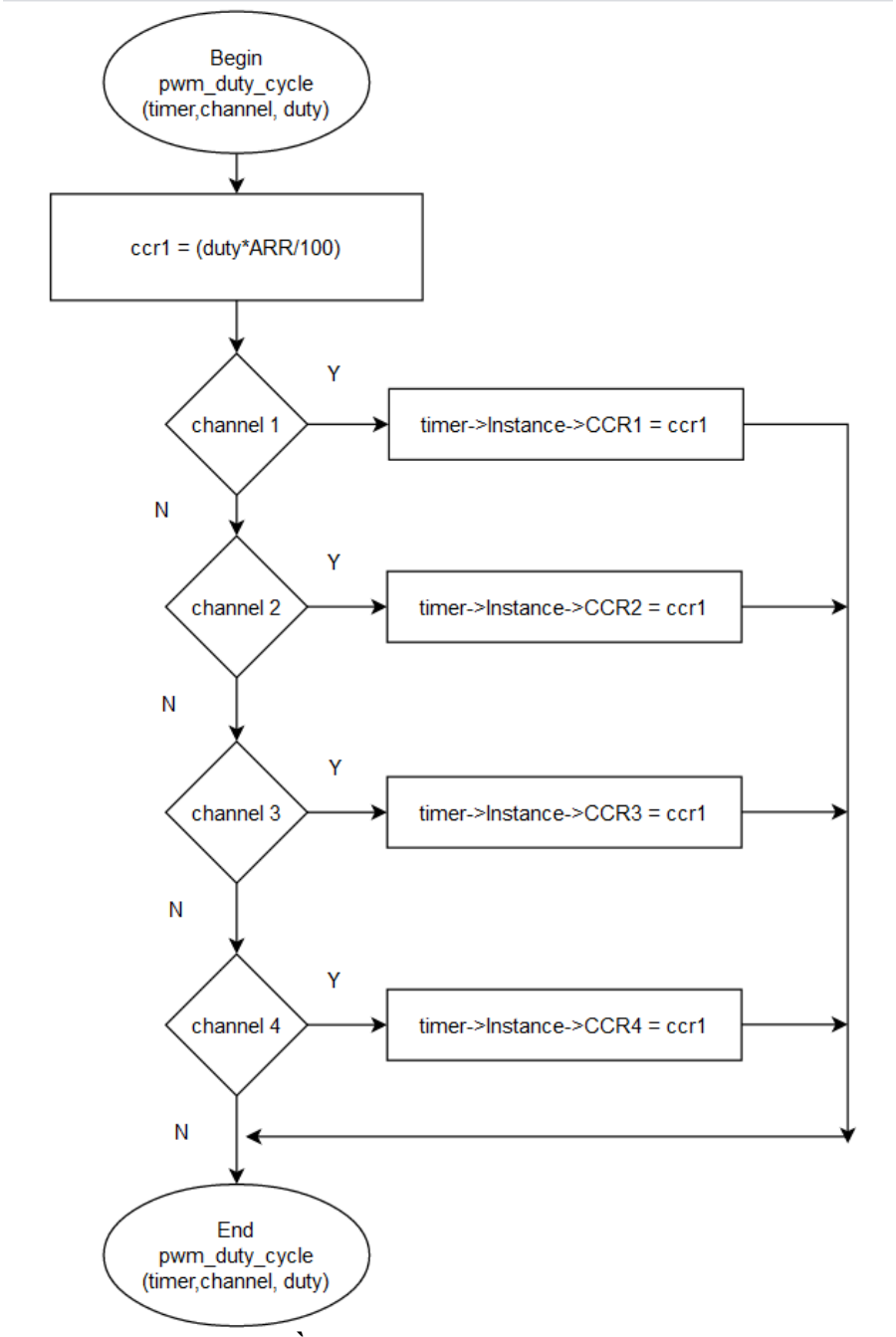
Hình 10: Lưu đồ chương trình con trong file main.c (4)

Giải thích: Chương trình này dùng để nhập mật khẩu khởi động xe. Với mật khẩu được định nghĩa sẵn là "12345678". Khi nhấn phím D thì sẽ vào trạng thái cho phép nhập mật khẩu. Khi đã vào trạng thái ENTER_LOCK_STATE sẽ thực hiện các công việc được trình bày riêng ở phần tiếp theo. Nếu mật khẩu đúng thì sẽ mở khóa xe(NORMAL_STATE), nếu sai thì xe sẽ bị khóa 30 giây. Khi 30 giây kết thúc quay lại trạng thái LOCK_STATE, đồng thời thực hiện reset giá trị các phần tử trong mảng bằng hàm memset để chờ nhập mật khẩu mới.



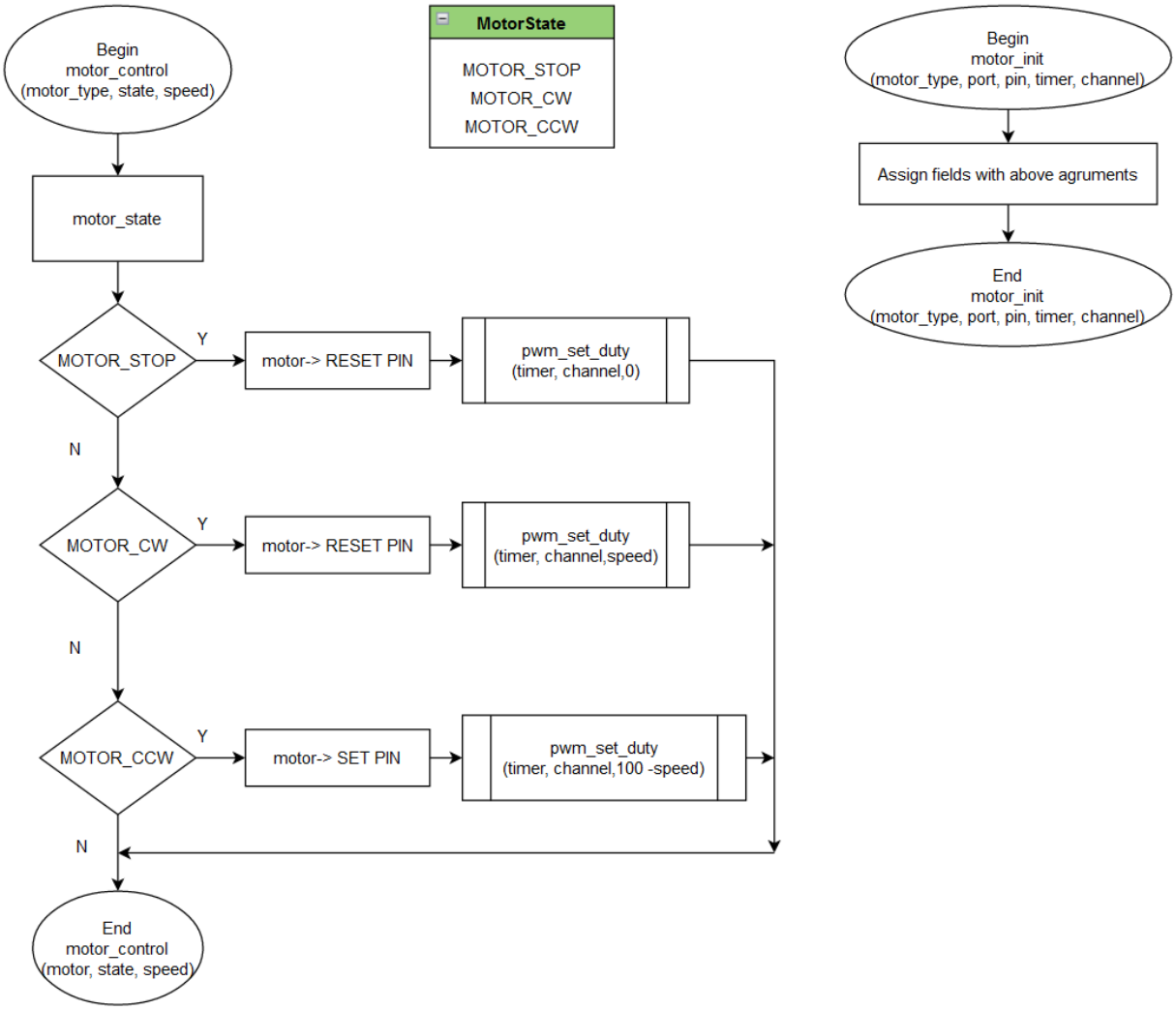
Hình 11: Lưu đồ giải thuật chương trình con trong file main.c (5)

Giải thích: Chương trình này thực hiện đọc giá trị biến key để tiến hành đẩy kí tự vào một mảng đệm (hay mật khẩu của người dùng nhập) gồm 8 phần tử và thêm một phần tử thứ 9 là '\0' để tạo thành một chuỗi. Sau đó nhấn phím A để kiểm tra và sử dụng hàm strcmp trong thư viện string.h để so sánh. Nếu mật khẩu định nghĩa sẵn và mật khẩu đã nhập đúng thì thực hiện mở khóa(NORMAL_STATE), nếu sai thì chuyển sang trạng thái LOCK_30S_STATE.



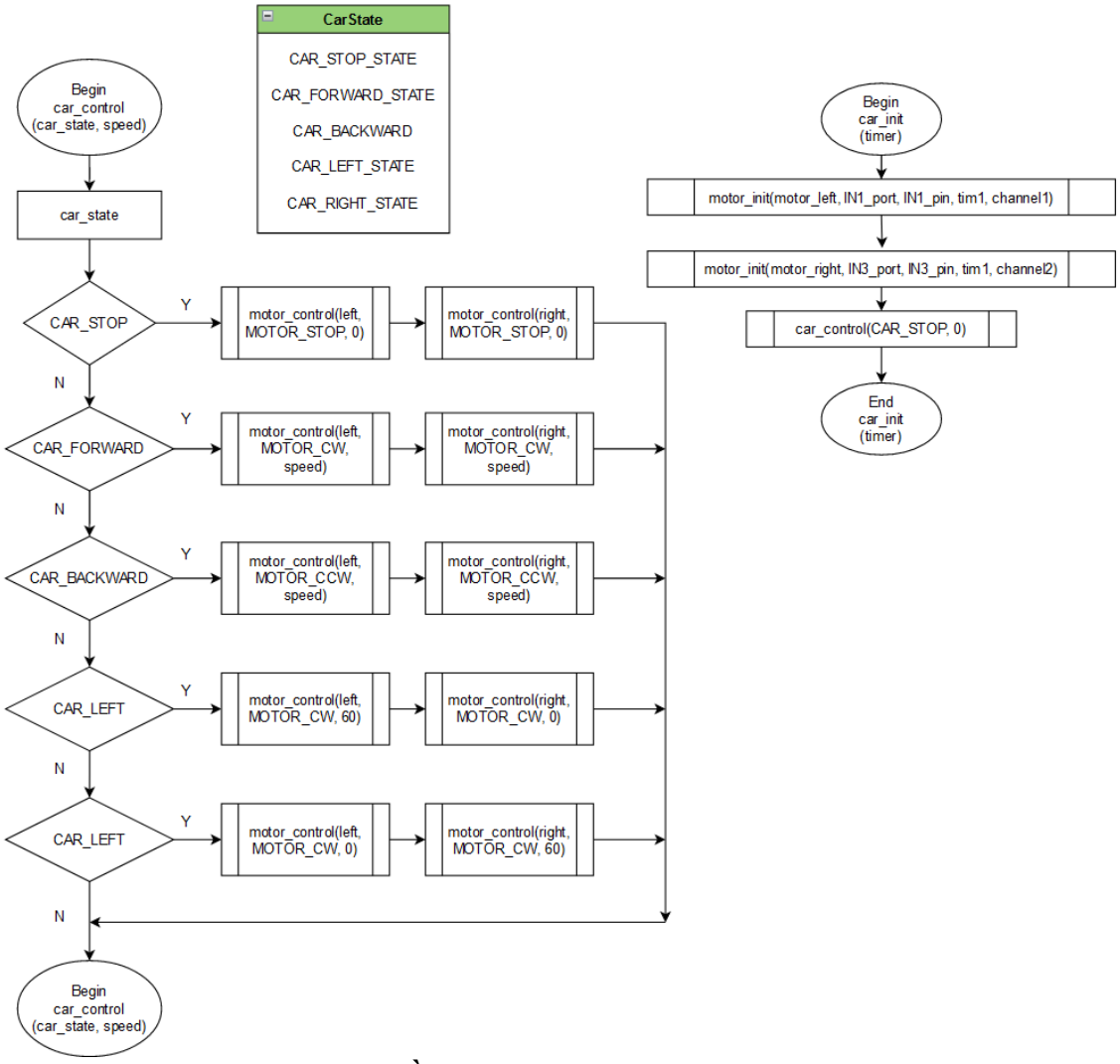
Hình 12: Lưu đồ giải thuật chương trình con trong file pwm.c

Giải thích: Chương trình con này sẽ thực hiện, đọc vào giá trị duty từ 0 đến 100 để đặt giá trị thanh ghi CCR1 (kênh 1). Mục đích là để gán các số từ 0 đến 100 ứng với từ 0 đến 999 (PWM tần số 1 kHz) cho thanh ghi CCR1 để làm thay đổi độ rộng xung.



Hình 13: Lưu đồ giải thuật chương trình con trong file Motor.c

Giải thích: Chương trình con này kiểm tra trạng thái được truyền vào để thực hiện điều khiển động cơ quay theo chiều ở trạng thái đó. Có 3 trạng thái cho một Motor là STOP(Dừng), CW(quay theo chiều kim đồng hồ) và CCW(quay theo ngược chiều kim đồng hồ). Ngoài ra chương trình còn được truyền vào biến tốc độ speed để điều chỉnh tốc độ theo mong muốn. Giải thích thêm với module L298 ta cần 2 chân(IN1 và IN2) để thực hiện đảo chiều quay và điều chỉnh tốc độ cho 1 motor ở ngõ ra OUT1 và OUT2.Do đó cần một chân xuất tín hiệu Output và một chân xuất xung PWM từ vi điều khiển STM32F103C8T6. Tương tự cho với IN3 và IN4.



Hình 14: Lưu đồ giải thuật chương trình con trong file car.c

Giải thích: Chương trình con này kiểm tra trạng thái được truyền vào để điều khiển hướng xe. Với một xe có 5 trạng thái được định nghĩa như sau: STOP(Dừng), FORWARD(chạy tới), BACKWARD(chạy lùi), LEFT(rẽ trái), RIGHT(rẽ phải). Ứng với mỗi trạng thái sẽ gọi hàm điều khiển motor. Với 2 đối tượng motor left và motor right ta sẽ điều hướng được xe theo tín hiệu điều khiển. Bên cạnh đó trước khi điều hướng xe, ta phải khởi tạo trạng thái ban đầu cho xe(hàm car_init) ở trạng thái mặc định là STOP.

4.2 Các đoạn mã lệnh chính trạm thu – Receive Block:

```
333   /* Initialize all configured peripherals */
334   MX_GPIO_Init();
335   MX_TIM1_Init();
336   MX_USART1_UART_Init();
337   MX_SPI2_Init();
338   MX_TIM3_Init();
339   /* USER CODE BEGIN 2 */
340
341   // Initialize NRF24
342   NRF24_begin(CEpin_GPIO_Port, CSNpin_Pin, CEpin_Pin, hspi2);
343
344   NRF24_setAutoAck(false);
345   NRF24_setChannel(52);
346   NRF24_setPayloadSize(32);
347   NRF24_openReadingPipe(1, RxpipeAddrs);
348   NRF24_enableDynamicPayloads();
349   NRF24_enableAckPayload();
350   HAL_Delay(200);
351   NRF24_startListening();
352
353   // UART Recieve Interrupt
354   HAL_UART_Receive_IT(&huart1, &rx_data, 1);
355
356   // EOS Init with 5ms cycles interrupt
357   EOS_Init(5);
358
359   // Inittialize System state
360   sys_state = LOCK_STATE;
361
362   // Initialize System state
363   signal_state = BLUETOOTH;
364
365   // Initialize CAR Robot
366   car_init(&htim1);
367   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
368   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, GPIO_PIN_SET);
369   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, GPIO_PIN_SET);
370
371   /* USER CODE END 2 */
372
```

Hình 15: Đoạn mã lệnh khởi tạo trạng thái mặc định

Giải thích:

+ Khởi tạo NRF24L01: hàm begin sẽ thực hiện khởi tạo các port và pin cho module NRF24 bao gồm 5 chân: CE, CSN, SCK, MISO, MOSI. Hàm bật acknowldge từ slave gửi về master(trong đồ án không bật – false). Hàm đặt kênh tần số cho NRF24 là 52. Hàm đặt kích thước payload (dữ liệu được chứa để truyền/nhận) là 32 byte. Hàm mở cổng nhận. Hàm bật payload động khi cần sử dụng đến và hàm bật Ackpayload. Cuối cùng là hàm bật chức năng lắng nghe(chờ dữ liệu gửi đến).

+ Khởi tạo EOS với chu kỳ ngắt 5ms và khởi tạo các trạng thái mặc định.

```
375   while (1)
376   {
377     /* USER CODE END WHILE */
378
379     /* USER CODE BEGIN 3 */
380
381     // Unclocking CAR
382     if (sys_state == NORMAL_STATE)
383     {
384       // UART Bluetooth controlling car
385       if (signal_state == BLUETOOTH)
386       {
387         uart_handle();
388       }
389
390       // NRF24 controlling car
391       NRF24_handle();
392     }
393
394   }
395   /* USER CODE END 3 */
396 }
```

Hình 16: Đoạn mã lệnh chương trình trong vòng lặp vô hạn while(1)

Giải thích: Trong vòng lặp vô hạn, ta liên tục thực hiện quét trạng thái của hệ thống và trạng thái của tín hiệu. Mặc định là điều khiển bằng Bluetooth nếu có tín hiệu từ khối truyền thì sẽ chuyển sang tín hiệu NRF24.

```
114 // ----------------------- 1. UART Interrupt Callback -----------------------------
115 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
116 {
117   if (huart->Instance == huart1.Instance)
118   {
119     uart_flag = 1;
120     HAL_UART_Receive_IT(&huart1, &rx_data, 1);
121   }
122 }
123 //--------------------------------------------------------------------------------
```

Hình 17: Đoạn mã lệnh trong hàm xử lí ngắt UART

Giải thích: Hàm này sẽ thực thi khi có một kí tự được gửi đến thông qua Bluetooth -> UART -> vi điều khiển. Biến cờ flag đặt lên 1 để báo hiệu có 1 kí tự được gửi đến cho hàm uart_handle biết thực hiện các chức năng khác.

```
125 //-------------------- 2. UART Bluetooth Processing -------------------------------
126 void uart_handle(void)
127 {
128   if (uart_flag && (signal_state == BLUETOOTH))
129   {
130     switch (rx_data)
131     {
132       case 'S':
133       {
134         car_control(CAR_STOP_STATE, 0);
135         break;
136       }
137       case 'F':
138       {
139         car_control(CAR_FORWARD_STATE, car_speed);
140         break;
141       }
142       case 'B':
143       {
144         car_control(CAR_BACKWARD_STATE, car_speed);
145         break;
146       }
147       case 'L':
148       {
149         car_control(CAR_LEFT_STATE, car_speed);
150         break;
151       }
152       case 'R':
153       {
154         car_control(CAR_RIGHT_STATE, car_speed);
155         break;
156       }
157       default:
158       {
159         if ('0' <= rx_data && rx_data <= '9')
160         {
161           car_speed = (rx_data - '0')*10;
162         }
163         break;
164       }
165     }
166     uart_flag = 0;
167   }
168 }
169 //-----------------------------------------------------------------------------------
```

Hình 18: Đoạn mã lệnh hàm xử lý UART

Giải thích: Hàm này thực hiện điều khiển xe thông qua Bluetooth. Với các chức năng điều khiển hướng xe và tăng giảm tốc độ.

```
//------------------------------ 3. NRF24 Processing ------------------------------
void NRF24_handle(void)
{
  if(NRF24_available())
  {
    NRF24_read(myRxData, 32);
  }
  switch (myRxData[0])
  {
    case 'O':
    {
      signal_state = BLUETOOTH;
      break;
    }
    case 'S':
    {
      car_control(CAR_STOP_STATE, 0);
      break;
    }
    case 'F':
    {
      car_control(CAR_FORWARD_STATE, 95);
      break;
    }
    case 'B':
    {
      car_control(CAR_BACKWARD_STATE, 95);
      break;
    }
    case 'R':
    {
      car_control(CAR_RIGHT_STATE, 50);
      break;
    }
    case 'L':
    {
      car_control(CAR_LEFT_STATE, 50);
      break;
     }
  }
}
//-------------------------------------------------------------------------------
```

Hình 19: Đoạn mã lệnh hàm xử lí NRF24

Giải thích: đoạn mã lệnh này dùng để điều hướng xe theo tín hiệu NRF24. Với tốc độ được đặt sẵn trong suốt quá trình chạy không thể thay đổi.

```
214 //------------------------- 4. KEYPAD 4x4 Processing ------------------------------
215 void KEYPAD_Handle(uint8_t key)
216 {
217   switch (sys_state)
218   {
219     case NORMAL_STATE:
220     {
221       // No task
222       // Car is controlling
223       break;
224     }
225     case LOCK_STATE:
226     {
227       if (key == 'D')
228       {
229         // Push D key to fill password
230         sys_state = ENTER_LOCK_STATE;
231         HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_14);
232       }
233       break;
234     }
235     case ENTER_LOCK_STATE:
236     {
237       if ((key >= '0') && (key <= '9'))
238       {
239         if (key != key_last)
240         {
241           if (_index < 8)
242           {
243             pass_buff[_index++] = key;
244             HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_14);
245             key_last = key;
246             if (_index == 8)
247             {
248               pass_buff[_index] = '\0';
249             }
250           }
251         }
252       }
253       else if (key == 'A')
254       {
255         // Push A key to check password
256         if (strcmp((char *)pass_buff, (char *)pass) == 0)
257         {
258           // Password correct
259           sys_state = NORMAL_STATE;
260           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,GPIO_PIN_RESET);
261         }
262         else
263         {
264           sys_state = LOCK_30S_STATE;
265           _index = 0;
266         }
267         HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_14);
268       }
269       break;
270     }
271     case LOCK_30S_STATE:
272     {
273       if (++time_led_err >=20)
274       {
275         HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_13);
276         time_led_err = 0;
277       }
278       // waiting 30 second
279       if (++time_lock_30s >= 1000)
280       {
281         memset(pass_buff, 0, PASS_LEN);
282         sys_state = LOCK_STATE;
283         HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
284         HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, GPIO_PIN_SET);
285         time_lock_30s = 0;
286       }
287       break;
288     }
289     default:
290
291       break;
```

Hình 20: Đoạn mã lệnh hàm thực thi KEYPAD

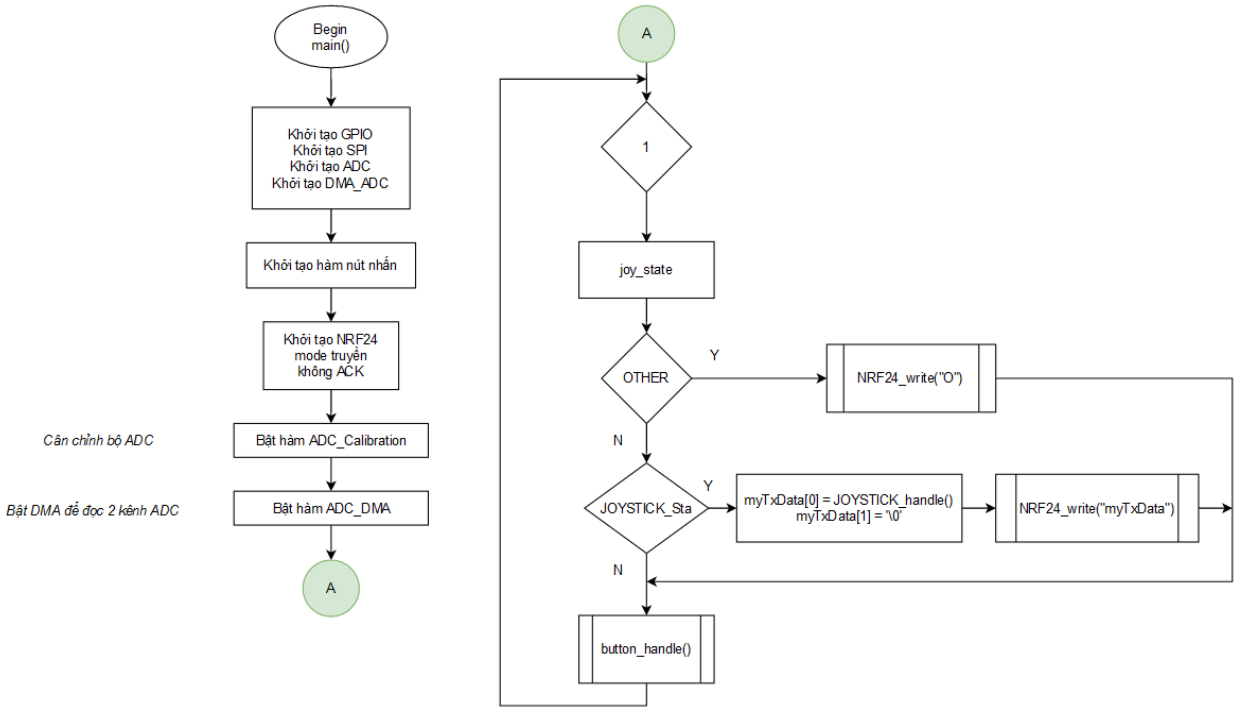
Giải thích: hàm này thực hiện đọc giá trị biến key truyền vào và xử lí nhập vào mảng đệm để kiểm tra mật khẩu có đúng hay không. Ngoài ra còn có thêm một số hiệu ứng LED báo hiệu trên mạch của xe thông qua các hàm HAL_GPIO_Writepin và HAL_GPIO_Togglepin.

```
296 //------------------------ 5. SysTick Interrupt ----------------------------------
297 void SysTick_Handler(void)
298 {
299   HAL_IncTick();
300   key = KEYPAD_GetKey();
301   KEYPAD_Handle(key);
302 }
303 //--------------------------------------------------------------------------------
```

Hình 21: Đoạn mã lệnh hàm ngắt SysTick do EOS gọi đến

Giải thích: khi SysTick timer đếm được 5ms sẽ gọi hàm ngắt này 1 lần và thực hiện quét KEYPAD và thực thi các hàm KEYPAD.

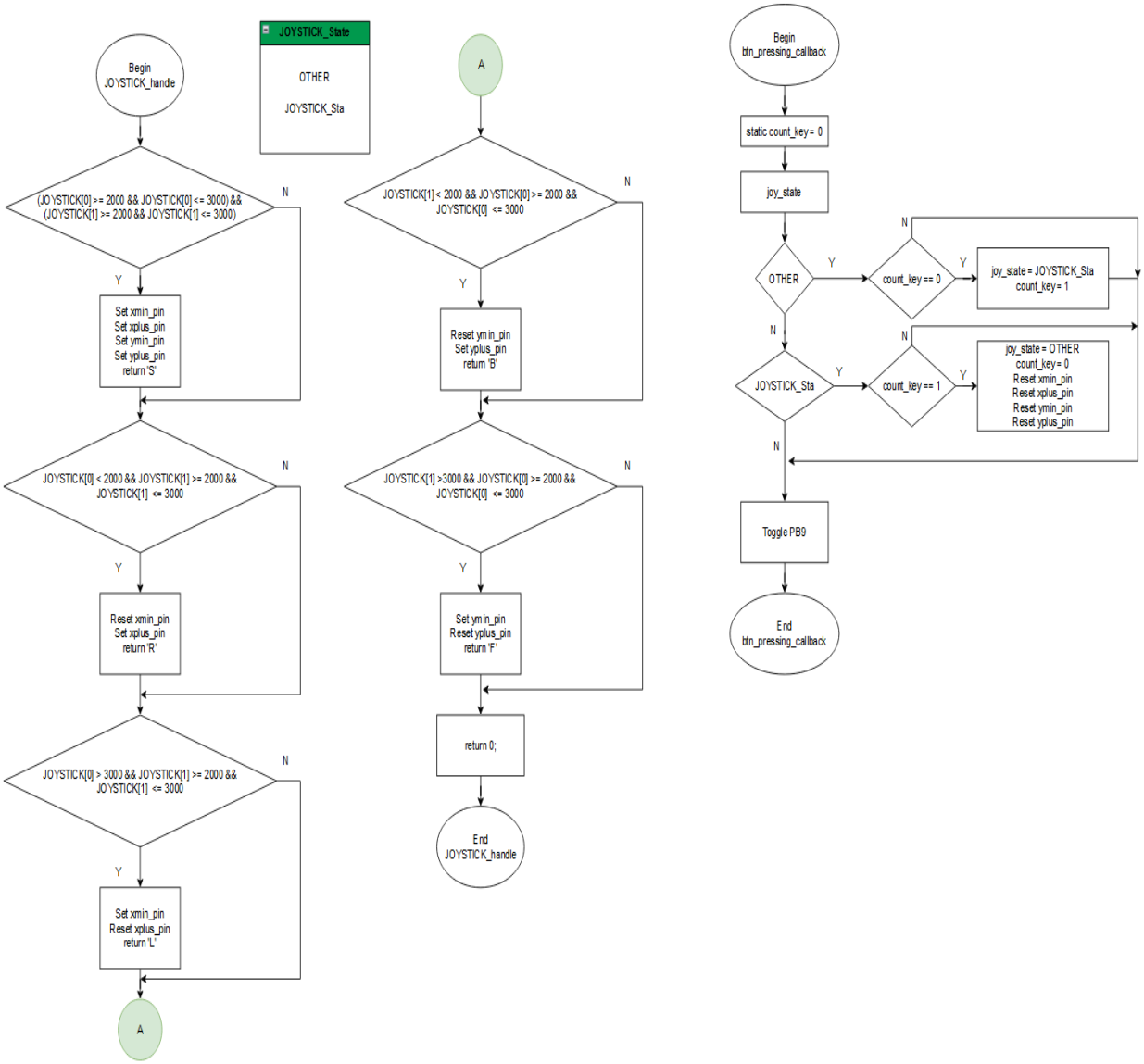
4.3 Lưu đồ giải thuật trạm phát – Transmit Block:



Hình 22: Lưu đồ giải thuật chương trình chính trong file main.c

Giải thích: Hệ thống sẽ quét qua các trạng thái để thực hiện truyền kí tự qua trạm nhận Receive Block. Nếu vào trạng thái OTHER, trạm truyền sẽ truyền kí tự 'O' tức cho phép điều khiển xe bằng Bluetooth. Nếu vào trạng thái JOYSTICK thì sẽ truyền các kí tự chuyển hướng S, R, L, F, B thông qua NRF24 đến trạm nhận Receive Block.

Đồng thời, quét qua hàm đọc nút nhấn để lấy sự kiện cho hàm btn_pressing_callback xử lí chuyển đổi qua lại giữa điều khiển bằng Bluetooth và NRF24.



Hình 23: Lưu đồ giải thuật chương trình con trong file main.c

Giải thích: Chương trình con này thực hiện so sánh các giá trị do bộ ADC đọc được ở cột x và y từ Joystick để bật/tắt các LED hiển thị hướng đi tương ứng và trả về kí tự để sử dụng truyền đi cho trạm nhận. Hình bên trái, nhận nút nhấn và chuyển đổi qua lai giữa 2 trạng thái OTHER(tương ứng Bluetooth) và JOYSTICK_Sta(tương ứng NRF24).

4.4 Các đoạn mã lệnh chính trạm truyền – Transmit Block:

```
uint8_t JOYSTICK_handle(void)
{
  if ((JOYSTICK[0] >= 2000 && JOYSTICK[0] <= 3000) && (JOYSTICK[1] >= 2000 && JOYSTICK[1] <= 3000))
  {
    HAL_GPIO_WritePin(xmin_GPIO_Port, xmin_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(xplus_GPIO_Port, xplus_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(ymin_GPIO_Port, ymin_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(yplus_GPIO_Port, yplus_Pin, GPIO_PIN_SET);
    return 'S';
  }

  if (JOYSTICK[0] < 2000 && JOYSTICK[1] >= 2000 && JOYSTICK[1] <= 3000)
  {
    HAL_GPIO_WritePin(xmin_GPIO_Port, xmin_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(xplus_GPIO_Port, xplus_Pin, GPIO_PIN_SET);
    return 'R';
  }

  if (JOYSTICK[0] > 3000 && JOYSTICK[1] >= 2000 && JOYSTICK[1] <= 3000)
  {
    HAL_GPIO_WritePin(xmin_GPIO_Port, xmin_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(xplus_GPIO_Port, xplus_Pin, GPIO_PIN_RESET);
    return 'L';
  }

  if (JOYSTICK[1] < 2000 && JOYSTICK[0] >= 2000 && JOYSTICK[0] <= 3000)
  {
    HAL_GPIO_WritePin(ymin_GPIO_Port, ymin_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(yplus_GPIO_Port, yplus_Pin, GPIO_PIN_SET);
    return 'B';
  }

  if (JOYSTICK[1] > 3000 && JOYSTICK[0] >= 2000 && JOYSTICK[0] <= 3000)
  {
    HAL_GPIO_WritePin(ymin_GPIO_Port, ymin_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(yplus_GPIO_Port, yplus_Pin, GPIO_PIN_RESET);
    return 'F';
  }
  return 0;
}
```

Hình 24: Đoạn mã lệnh hàm JOYSTICK_handle

Giải thích: Giá trị ADC được lưu trong 2 phần tử của mảng. Sau đó so sánh với các mức 2000 và 3000 vì giá trị ADC khi chưa gạt Joystick nằm khoảng từ 2300 đến 2500. Nếu giá trị ADC của trục Y lớn hơn 3000 thì sẽ gửi tín hiệu chạy thẳng 'F', nếu nhỏ hơn 2000 thì sẽ gửi tín hiệu chạy lùi 'B'. Tương tự với trục X lớn hơn 3000 thì sẽ gửi tín hiệu rẽ trái 'L', nếu nhỏ hơn 2000 thì sẽ gửi tín hiệu rẽ phải 'R'.

```
121 void  btn_pressing_callback(Button_Typdef *ButtonX)
122 {
123   static uint8_t count_key = 0;
124   switch (joy_state)
125   {
126     case OTHER:
127     {
128       if (count_key == 0)
129       {
130         joy_state = JOYSTICK_Sta;
131         count_key = 1;
132       }
133       break;
134     }
135     case JOYSTICK_Sta:
136     {
137       if (count_key == 1)
138       {
139         joy_state = OTHER;
140         count_key = 0;
141         HAL_GPIO_WritePin(xmin_GPIO_Port, xmin_Pin, GPIO_PIN_RESET);
142         HAL_GPIO_WritePin(xplus_GPIO_Port, xplus_Pin, GPIO_PIN_RESET);
143         HAL_GPIO_WritePin(ymin_GPIO_Port, ymin_Pin, GPIO_PIN_RESET);
144         HAL_GPIO_WritePin(yplus_GPIO_Port, yplus_Pin, GPIO_PIN_RESET);
145       }
146       break;
147     }
148   }
149   HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_9);
150 }
151
```

Hình 25: Đoạn mã lệnh xử lí nút nhấn trên Joystick

Giải thích: Hàm này với chức năng khi nhấn nút 1 lần sẽ chuyển đổi qua lại giữa 2 trạng thái OTHER và JOYSTICK_Sta. Nếu nhấn lần đầu thì sẽ chuyển sang trạng thái JOYSTICK_Sta và nếu nhấn lần tiếp theo sẽ chuyển qua trạng thái OTHER và bật 4 LED sáng lên. Tương tự lặp lại ở các lần sau.

```
206   while (1)
207   {
208     /* USER CODE END WHILE */
209
210     /* USER CODE BEGIN 3 */
211     switch (joy_state)
212     {
213       case OTHER:
214       {
215         NRF24_write("O", 32);
216         HAL_Delay(50);
217         break;
218       }
219       case JOYSTICK_Sta:
220       {
221         myTxData[0] = (char)JOYSTICK_handle();
222         myTxData[1] = '\0';
223         NRF24_write(myTxData, 32);
224         HAL_Delay(50);
225         break;
226       }
227     }
228     button_handle(&but1);
229
230   }
231   /* USER CODE END 3 */
```

Hình 26: Đoạn mã lệnh trong siêu vòng lặp while(10)

Giải thích: Đoạn này sẽ thực hiện quét liên tục các trạng thái và quét phím nhấn. Nếu trạng thái là OTHER sẽ liên tục gửi kí tự 'O' để trạm nhận chuyển trạng thái sang Bluetooth. Nếu trạng thái là JOYSTICK sẽ thực hiện gửi các kí tự trả về từ hàm JOYSTICK_handle đến trạm nhận để điều hướng xe.

5. Kết quả (2đ)

5.1 Kết quả của đề tài:

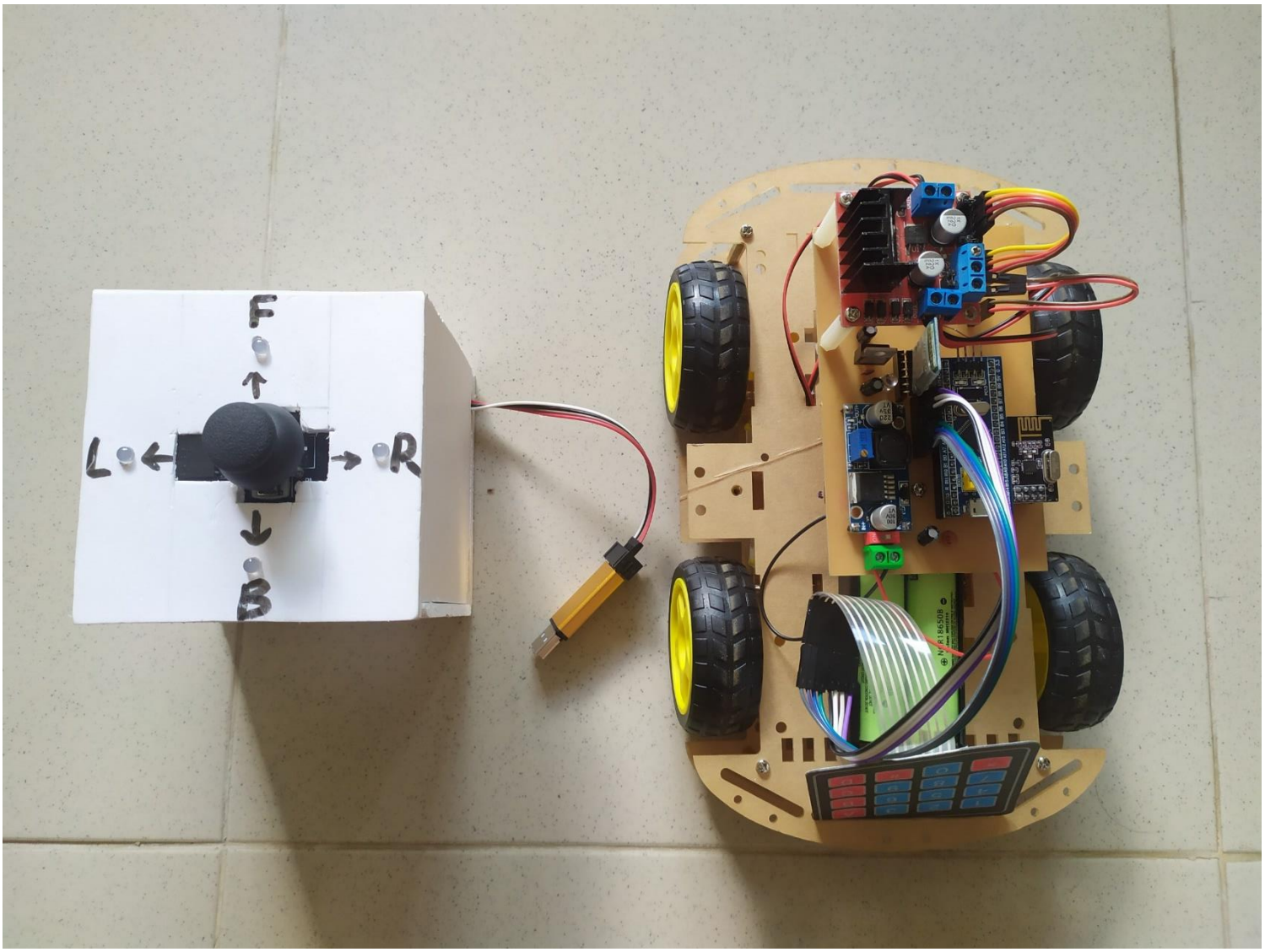
- Có áp dụng lập trình nhiều trạng thái để xử lí.
- Có áp dụng hệ điều hành nhúng EOS để xử lí các tác vụ cần thiết, cũng như sử dụng quy tắc timeout (trạng thái khóa xe 30s nếu nhập mật khẩu sai).
- Về tính chuyển động của xe:
  + Chế độ Bluetooth: đáp ứng chậm hơn nhiều so với mong muốn thực tế. Thời gian trễ trong việc gửi tín hiệu rất lớn và dễ bị nhiễu.
  + Chế độ NRF24: đáp ứng nhanh và chính xác. Xe có thể di chuyển tốt trong bán kính < 15m. Hầu như không có độ trễ ở các tiến trình điều hướng xe. Độ ổn định cao hơn so với Bluetooth. Rất phù hợp cho việc thiết kế xe điều khiển cỡ nhỏ như đồ chơi trẻ em hoặc cao hơn là nghiên cứu các hệ thống cảm biến trong xe thông qua chuẩn giao tiếp này.
- Hiểu thêm về DMA cho đọc nhiều kênh ở bộ ADC và 2 giao thức truyền dữ liệu quan trọng trong đề tài là UART và SPI.

5.2 Khắc phục đề tài:

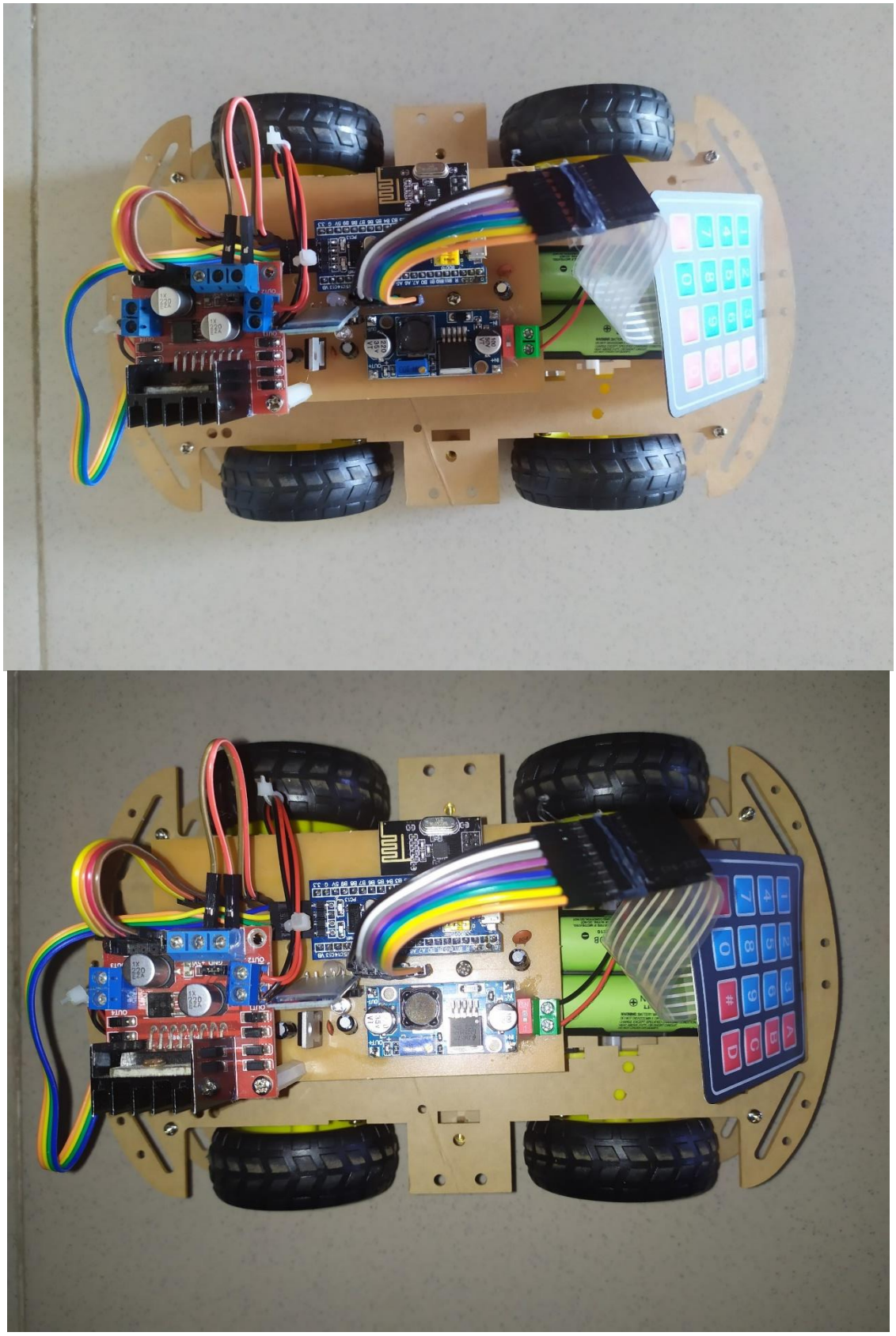
- Điều khiển thông qua Bluetooth: có thể module giá rẻ chưa ổn định cao. Việc xử lí tín hiệu còn thô sơ chưa qua bộ lọc nhiễu nào. Vị trí lắp đặt module trên board mạch chưa hợp lí.
  Đề xuất giải pháp: có thể dùng module Bluetooth khác để thử nghiệm như các chuẩn Bluetooth Low Energy (BLE).
- Điều khiển thông qua NRF24L01: khoảng cách còn truyền còn ngắn hơn so với mong muốn là 30m.
  Đề xuất giải pháp: sử dụng module có gắn thêm anten rời để tăng khuếch đại sóng RF có thể kể đến như: NRF24L01 + PA/LNA 2.4GHz Anten rời.
- Sử dụng một bộ nguồn riêng cho trạm điều khiển, không cắm mạch nạp ST Link v2.
- Thiết kế lại mạch PCB tích hợp thêm sẵn KEYPAD trên mạch, không dùng KEYPAD rời.

5.3 Một số hình ảnh và video demo đề tài:

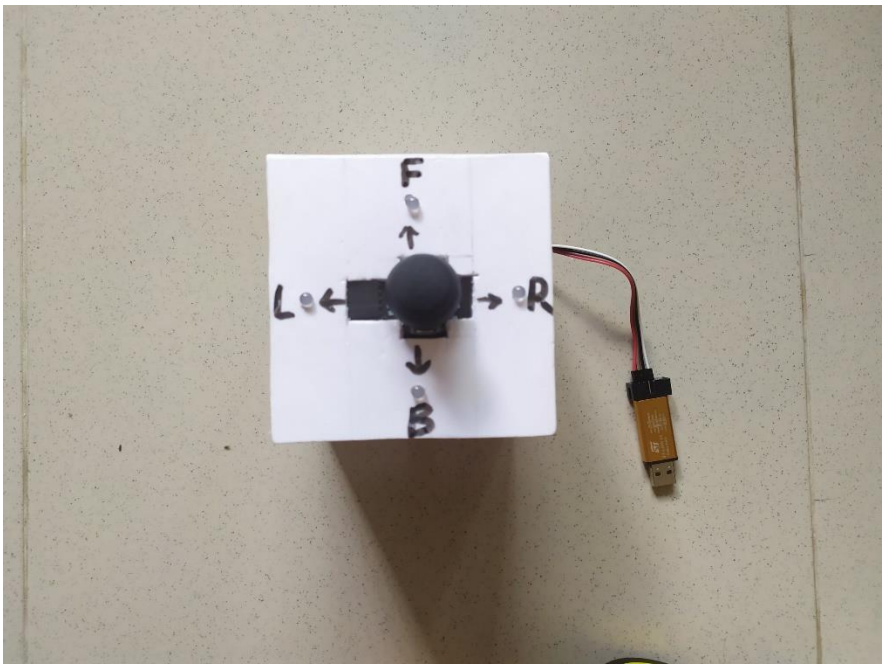
5.3.1 Hình ảnh thực tế:



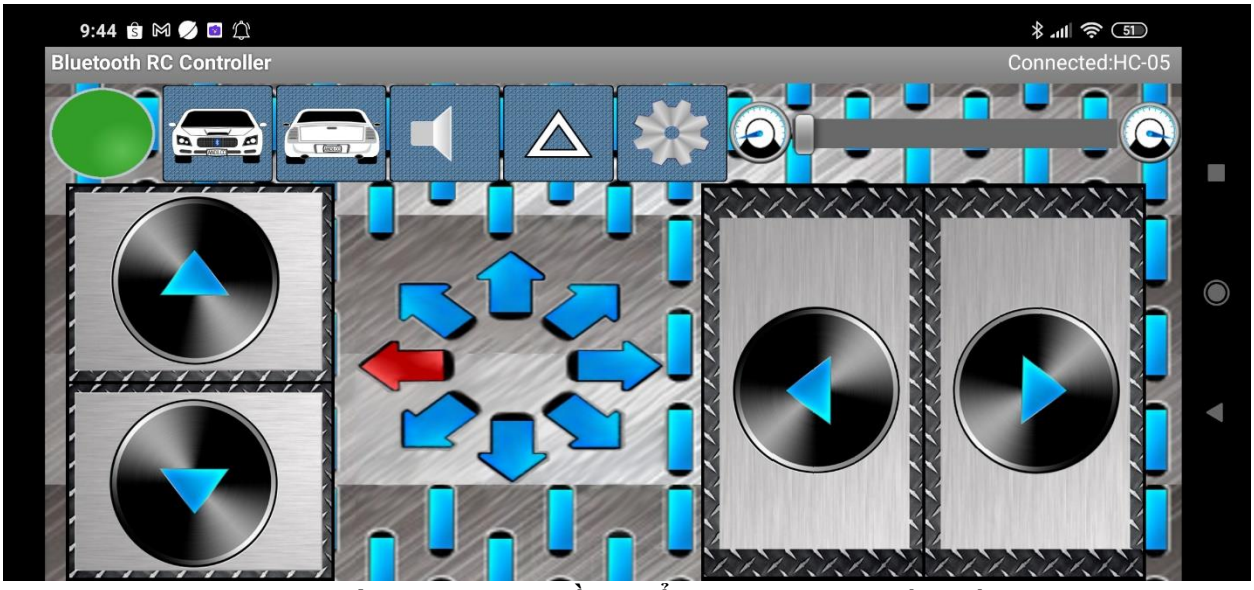
Hình 27. Khung xe, hệ tthống mạch điện và bộ điều khiển



Hình 28. Khung xe hoàn chỉnh



Hình 29. Trạm điều khiển sử dụng Joystick



Hình 30. Trạm điều khiển sử dụng App trên điện thoại

5.3.2 Link video demo:

Link demo ở chế độ NRF24L01: **https://youtu.be/idwfzNZjZEc**

Link demo ở chế độ Bluetooth: https://www.youtube.com/watch?v=eXh3cRqOpSA

Hết