

# I. Yêu Cầu

Xây dựng một sơ đồ mạch trên Proteus và lập trình, mạch gồm có:

- Sử dụng VĐK STM32
- Giao tiếp với matrix key 4x4
- Cảm biến nhiệt độ NTC
- Màn hình LCD 20x4
- Thẻ nhớ SD card
- Giao tiếp UART với máy tính

Hoạt động:

- Chương trình sử dụng FreeRTOS
- Chương trình gồm 2 chế độ hoạt động là Running và config
  - Trong chế độ Running:
    - + Động cơ quay thuận với tốc độ s1 trong thời gian t1, sau đó đảo chiều quay với tốc độ s2 trong thời gian t2 (tốc độ quy đổi theo độ rộng xung PWM cũng dc ko cần đọc encoder, thời gian tính theo giây)
    - + Cảm biến nhiệt độ NTC được đọc với chu kỳ 1s 1 lần
    - + Ghi vào thẻ nhớ SD card thông tin dưới dạng file, mỗi lần khởi động mạch sẽ tạo 1 file mới, mỗi lần ghi các thông tin gồm có: thời gian t1, thời gian t2, tốc độ s1 tốc độ s2, nhiệt độ NTC. (sử dụng thư viện FATFS hoặc một thư viện nào đó hỗ trợ đọc ghi định dạng file cũng dc)
    - + Hiển thị lên màn hình LCD các thông tin t1,t2,s1,s2,NTC
    - + Gửi lên UART các thông tin t1,t2,s1,s2,NTC định dạng giống như ghi vào file
  - Trong chế độ Config:
    - + Nhấn 1 tổ hợp 2 phím (tự chọn) trên matrix key để vào màn hình nhập password
    - + Nhập đúng password thì vào màn hình config
    - + Màn config dạng menu có 6 mục, vào 4 mục để cài đặt các thông số t1 t2 s1 s2, 1 mục vào để thay đổi password, một mục để save các thông tin đã bị thay đổi ở các mục còn lại vào thẻ nhớ flash (thẻ nhớ flash để 1 file lưu config các thông số kia)
    - + Trong màn hình config dùng 1 nút để di chuyển menu, 1 nút để enter, một nút exit, các nút số để nhập password và thông số

## II. Các linh kiện sử dụng

### 1. Nhiệt điện trở Thermistor và cảm biến NTC 16D7

#### Khái niệm:

Nhiệt điện trở hay điện trở nhiệt (**Thermistor**) là loại điện trở có trở kháng của nó thay đổi một cách rõ rệt dưới tác dụng nhiệt, hơn hẳn so với các loại điện trở thông thường. Từ thermistor được kết hợp bởi từ **thermal** (nhiệt) và **resistor** (điện trở).

#### Nguyên lý hoạt động:

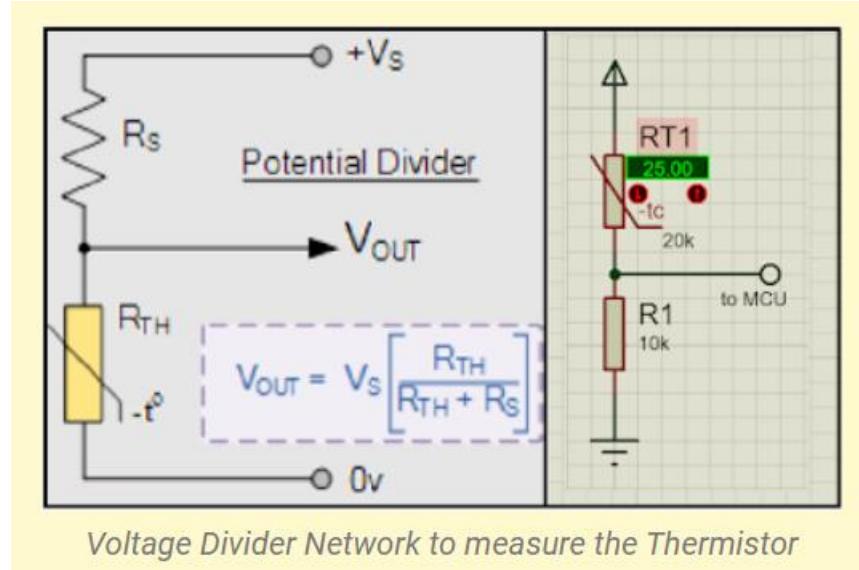
Các **điện trở thông thường** ít nhạy cảm với nhiệt độ hơn, do đặc tính vật lý của vật liệu làm ra chúng: thường là cacbon, vật liệu gốm, nikrom, tantalum nitride, ...

Trong khi đó **nhiệt điện trở** thường được tạo thành từ hỗn hợp các oxit kim loại như crom, cobalt, sắt, mangan, niken, được ép thành hạt, đĩa hoặc hình trụ và sau đó được bao bọc bằng vật liệu không thấm như epoxy hoặc thủy tinh. Đặc tính vật lý của các vật liệu này là thay đổi độ dẫn (điện trở) khi nhiệt độ tiếp xúc thay đổi.

#### Một số đặc điểm của Thermistor:

- **Rải giá trị đo :** Thermistor có rải đo nhiệt độ khá nhỏ, thường nằm trong rải chênh lệch 50 độ C so với nhiệt độ đặt trung tâm (-100°C đến 300°C).
- **Độ ổn định và độ nhạy cao,** giá thành rẻ so với các cảm biến nhiệt độ khác.
- **Độ bền cao,** kích thước nhỏ gọn thích hợp đo nhiệt độ tại một điểm.
- **Nhược điểm :** Không tuyến tính nên khó xử lý chính xác bằng vi điều khiển. Rải nhiệt độ khá hẹp, thời gian phản ứng chậm.
- Thermistor chia làm 2 loại chính là NTC và PTC.
  - Với PTC khi nhiệt độ tăng thì điện trở tăng, nhiệt độ giảm thì điện trở giảm.
  - Với NTC khi nhiệt độ tăng thì điện trở giảm, nhiệt độ giảm thì điện trở tăng

#### Giao thiếp Thermistor với vi điều khiển:



Do đầu ra của cảm biến là điện trở nên cách giao tiếp đơn giản nhất là sử dụng mạch phân áp.Sau đó điện áp đo được dễ nối đến chân ADC của Ví điều khiển, sau khi đo được giá trị số ADC\_Output ta có thể tính toán ngược lại giá trị của điện trở Thermistor(Rth) theo công thức như sau:

$$R_{th} = \left( \frac{(2^N - 1) * R}{ADC\_Output} \right) - R$$

Với N là độ phân giải của bộ ADC, R là điện trở nối tiếp với Rth

#### Tính toán giá trị nhiệt độ từ điện trở Rth

Thermistor có giá trị đầu ra thường là phi tuyến, hai nhà khoa học là John S. Steinhart và Stanley R. Hart đã nghiên cứu và tìm ra công thức chung về sự liên quan giữa điện trở và nhiệt độ của Thermistor, đó là phương trình Steinhart-Hart. Với 3 hệ số A,B,C

Còn một công thức rút gọn với hệ số B như sau:

#### B Parameter Equation

- NTC thermistors can also be characterised with the *B* parameter equation, which is essentially the Steinhart Hart equation with  $c=0$

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln \left( \frac{R}{R_0} \right)$$

## Cảm biến NTC 16D7

Em sử dụng NTC 16D7 Đây là cảm biến có điện trở 16ohm tại nhiệt độ phòng 25 độ C  
Theo bảng ta cũng biết được hệ số B của cảm biến B = 3050.

Do ở 25 độ C cảm biến có điện trở 16ohm nên em mắc mạch nối tiếp với điện trở 10ohm để tạo thành mạch chia áp.



Cảm biến nhiệt độ NTC 16D7

Part No.	Resistance at 25°C (Ω)	Max Steady State Current (A)	Thermal Dissipation Constant (mW/°C)	Thermal Time Constant (Sec.)	Beta Constant (K, 25/85°C, ±5%)	Max. Capacitance (μF)	
						120 Vac.	240 Vac.
2.5D - 7	2.5	3.0	9	35	2950	330	80
5D - 7	5.0	3.0	9	35	2950	330	80
8D - 7	8.0	2.5	9	35	2950	330	80
10D - 7	10.0	2.3	9	32	3000	330	80
15D - 7	15.0	2.0	12	30	3050	330	80
16D - 7	16.0	2.0	12	30	3050	330	80
22D - 7	22.0	1.5	10	32	3100	330	80
33D - 7	33.0	1.5	10	32	3100	330	80
50D - 7	50.0	1.2	8	28	3150	330	80

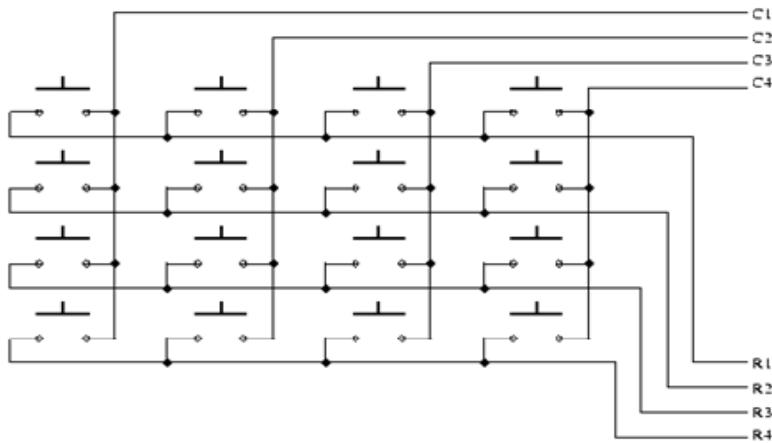
Bảng thông số của cảm biến NTC 16D7

## 2. Key pad 4x4

KeyPad là một thiết bị nhập chứa các nút bấm cho phép người dùng nhập các chữ số, chữ cái hoặc ký tự điều khiển. KeyPad không chứa tất cả bảng mã ASCII như keyboard vì thế nó thường được sử dụng trong các ứng dụng chuyên dụng và tương đối đơn giản, ở đó, số lượng nút nhấn thay đổi phụ thuộc vào ứng dụng. KeyPad  $4 \times 4$  là bàn phím gồm 16 nút nhấn, được xếp thành 4 hàng, mỗi hàng gồm 4 phím bấm.

## Cấu tạo

KeyPad  $4 \times 4$  gồm: 8 đầu vào/ra và 16 phím nhấn kết nối theo sơ đồ dưới đây:

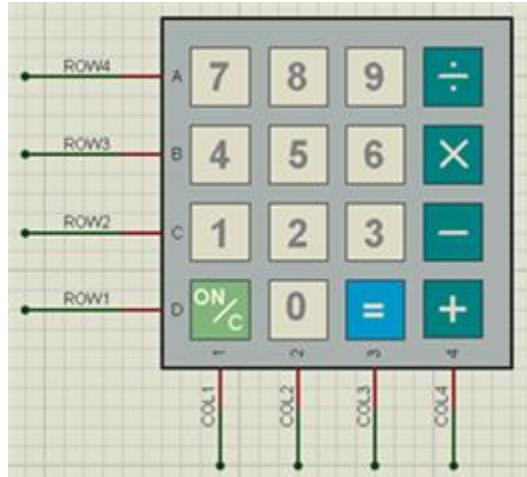


Các phím bấm được chia thành 4 hàng và 4 cột, 1 đầu của nút bấm được nối với đầu vào cột(C), đầu kia được nối với đầu vào hàng(R).

## Nguyên lý hoạt động

Để làm việc với KeyPad  $4 \times 4$ , người lập trình thường sử dụng giải thuật “quét phím”. Giải thuật này yêu cầu VĐK liên tục đưa các tín hiệu đầu ra ở hàng (hoặc cột) và thu lại đầu vào ở cột (hoặc hàng), nếu phím được bấm, đầu phát tín hiệu sẽ được kết nối với đầu thu, từ đó xác định được phím đã bấm.

Việc lựa chọn đầu ra/vào hình thành 2 phương pháp quét phím: theo chiều dọc và theo chiều ngang. Trong báo cáo này, tín hiệu xuất ra ở các hàng và thu lại ở các cột. Giả sử một nút ‘2’ được nhấn, khi đó đường C và 2 được nối với nhau. Nếu đường C được nối với GND, khi đó, điện áp



ở chân số 2 sẽ mang điện áp 0V. Tương tự như thế với các phím cùng hàng C. Áp dụng thuật toán vào mô hình KeyPad dưới đây:

## Thuật toán

**Bước 1:** set các chân ROW1, ROW2, ROW3, ROW4 như các chân Output và giữ chúng ở mức cao, các chân COL1, COL2, COL3, COL4 như các chân input có điện trở kéo lên.

**Bước 2:** đưa tín hiệu đầu ra ở các chân ROW1 = 1, ROW2 = 1, ROW3 = 1 và ROW4 = 1. Kiểm tra tín hiệu ở các chân COL1, COL2, COL3, COL4 luôn bằng 1 dù có phím nào được nhấn hay không.

**Bước 3:** đưa tín hiệu đầu ra ở các chân ROW1 = 0, ROW2 = 1, ROW3 = 1 và ROW4 = 1.

Kiểm tra COL1, 2, 3 và 4, nếu phím thuộc hàng 1 được nhấn sẽ giá trị COL nhận được bằng 0, ví dụ: ON/C được nhấn, COL1 = 0, COL2, 3, 4 = 1. Nếu phím thuộc các hàng khác (2, 3, 4) được nhấn, các chân COL1, COL2, COL3, COL4 luôn bằng 1.

Bằng thao tác trên, sẽ xác định phím đã được nhấn nếu nó ở hàng 1.

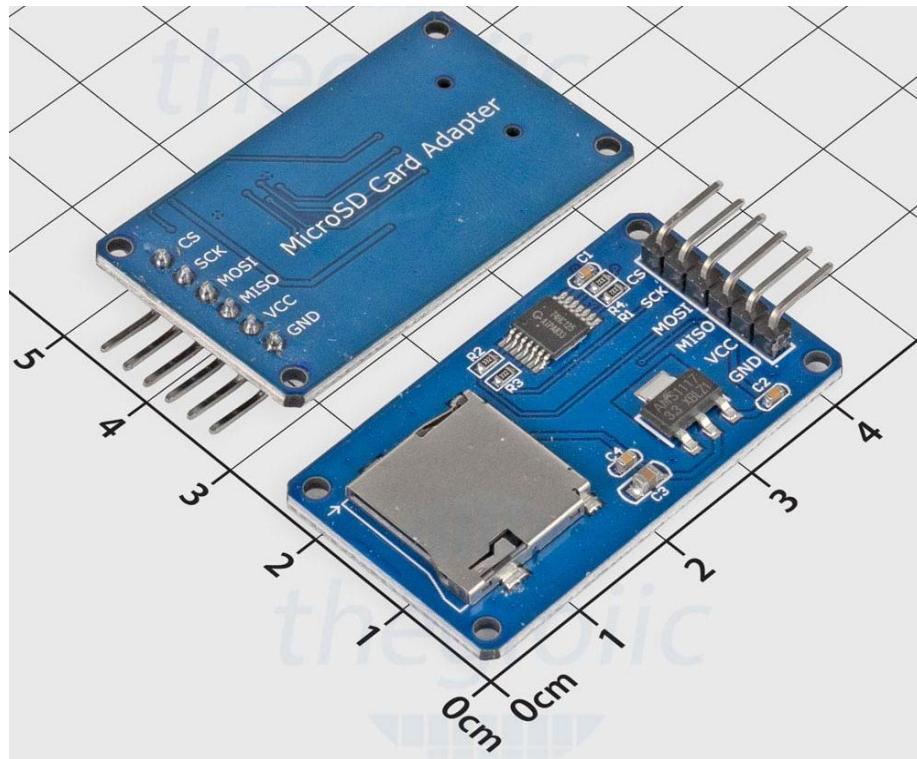
**Bước 4:** tiếp tục đưa tín hiệu đầu ra ở các chân ROW1 = 1, ROW2 = 0, ROW3 = 1, ROW4 = 1 để xác định phím bấm được nhấn nếu nó ở hàng 2.

**Bước 5:** thực hiện quá trình dịch chân đầu ra mang điện áp mức 0 một cách liên tục và xác định phím được bấm.

### 3. Module SD Card với SPI & Thư viện FATFS

Một số dòng vi điều khiển STM32 có **phần cứng SDIO (SDMMC) tích hợp**, được thiết kế đặc biệt để giao tiếp với thẻ SD với **tốc độ tối đa**. Tuy nhiên, thẻ SD vẫn có thể được sử dụng thông qua **giao tiếp SPI**, một giao thức có mặt trên **tất cả vi điều khiển STM32** cũng như hầu hết các vi điều khiển khác.

SPI Có giao tiếp nối tiếp đa năng, có thể dùng để giao tiếp với thẻ SD trên các vi điều khiển tầm thấp. Tốc độ chậm nhưng giao tiếp đơn giản hơn SDIO.

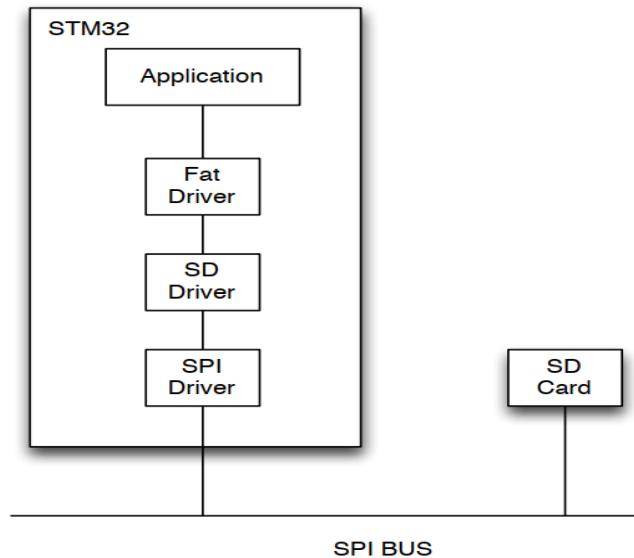


FatFS là một loại module hệ thống tệp FAT chung cho các hệ thống nhúng. fatFS được viết phù hợp với ANCI C và tách biệt với lớp I/O nên nó độc lập với phần cứng. FatFS nằm ở tầng Middleware giữa tầng hardware và Application trong hệ thống nhúng. Do đó, nó độc lập với các nền tảng và thiết bị lưu trữ. Nó có thể được kết hợp vào các bộ vi điều khiển nhỏ với tài nguyên hạn chế, chẳng hạn như 8051, PIC, AVR, ARM, Z80, RX, v.v.

Các lệnh FATFS chúng ta hay sử dụng chủ yếu là:

- f\_mount(): Register/Unregister a work area
- f\_open(): Open/Create a file
- f\_close(): Close a file
- f\_read(): Read a file

- `f_write()`: Write a file
- `f_lseek()`: Move read/write pointer, Expand a file size
- `f_truncate()`: Truncate a file size
- `f_sync()`: Flush cached data
- `f_opendir()`: Open a directory

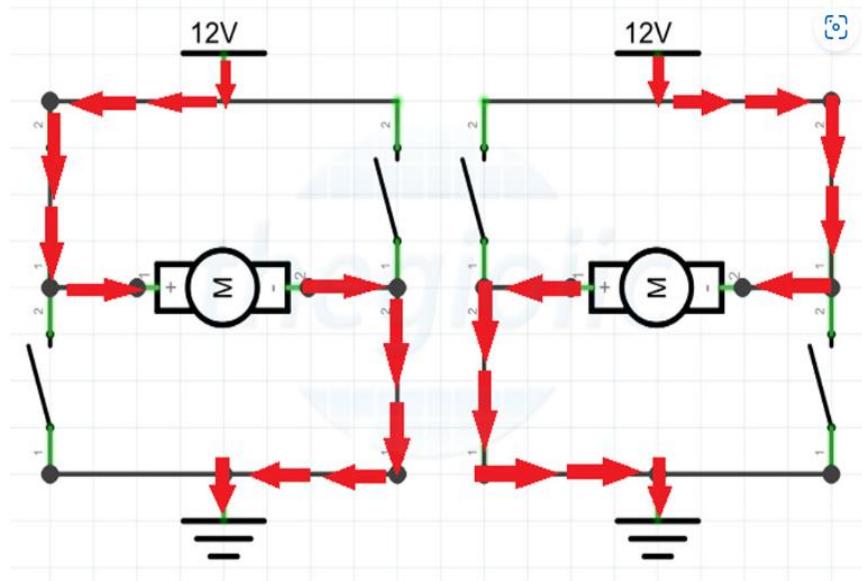


#### 4. LD298N và Motor DC 5V

Module L298 là một mạch điều khiển động cơ một chiều DC cùng lúc. L298 là IC điều khiển cầu kép toàn kỳ có dải điện áp hoạt động rộng, xử lý dòng tải có mức tối đa 3A. Bao gồm điện áp bảo hòa thấp và bảo vệ quá nhiệt. Có cấu tạo từ hai mạch cầu H transistor.

**Nguyên lý hoạt động của mạch cầu H**

Động cơ DC hòn đảo chiều quay khi thay đổi chiều dòng điện chạy vào động cơ. Do đó ta hoàn toàn có thể đổi chiều cấp điện cho động cơ để làm thay đổi chiều quay. Hình bên dưới là sơ đồ mạch cầu H đơn thuần sử dụng 4 công tắc nguồn. Các công tắc nguồn hoàn toàn có thể sửa chữa thay thế bằng relay hoặc những khóa bán dẫn công suất



## 5. Một số linh kiện khác

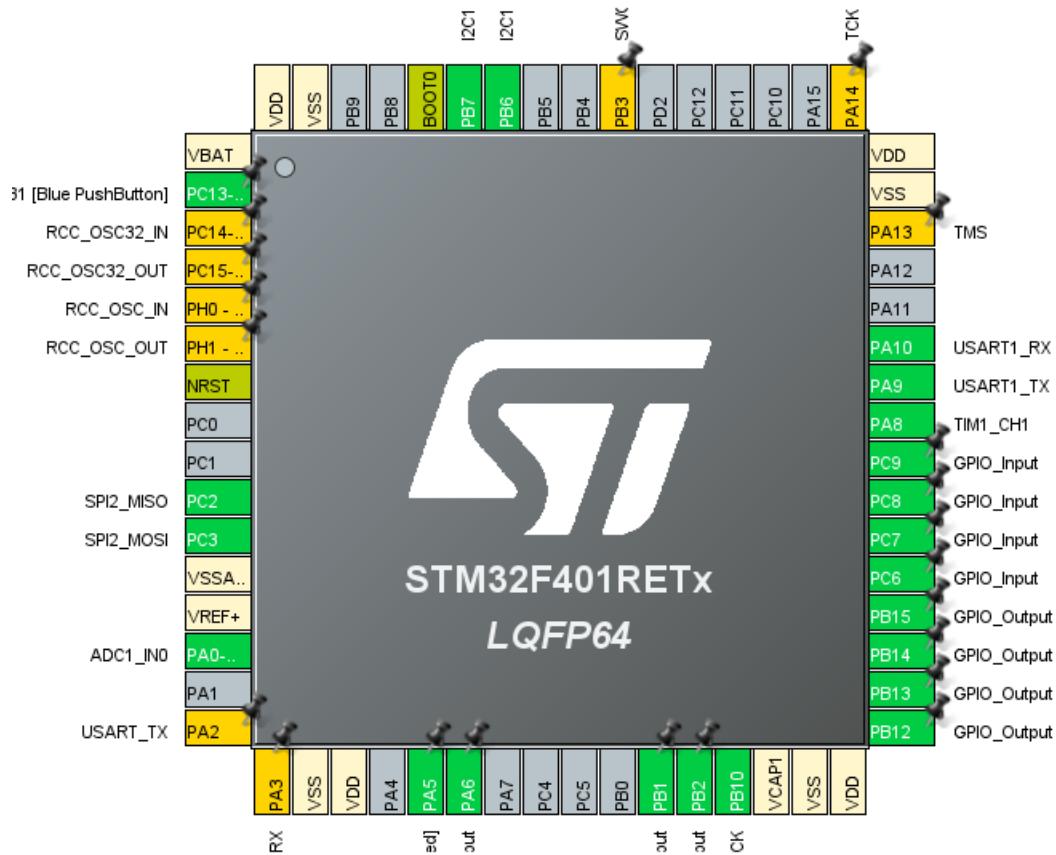
Vi điều khiển trung tâm : STM32F401RE

LCD 16X4 có I2C

Điện trở 10Ohm

Module USB to TLL

### III. Kết nối phần cứng



- PC9,PC8,PC7,PC6: hàng của key pad
- PC15,PC14,PC13,PC12: cột của key pad
- PA10,PA9: UART nối USB to TTL
- PB7,PB6: I2C nối LCD
- PA0: ADC đọc điện áp mạch chia áp của NTC
- PA6: CS của SD card
- PC2,PC3 : MISO, MOSI của sd card
- PA8: PWM 1 channel 1 nối L289N
- PB1,PB2: nối IN1,IN2 của LN298N

### IV. Thiết kế chương trình

Xác định số lượng task

Với yêu cầu của đề bài em xác định có **7 task** như sau:

- **Sensor task** : đọc dữ liệu từ cảm biến , hiển thị lên LCD
- **Motor task** : điều khiển motor
- **Sd card task** : ghi dữ liệu vào sd card
- **Print task** : truyền dữ liệu lên UART
- **Scan key pad task** : đọc tín hiệu từ key pad
- **Menu config task** : xử lý khi ở tab config các thông số
- **Pass word task** : xử lý khi ở tab nhập pass word

## Xác định mức ưu tiên của các task

- **Sensor task** : Mức ưu tiên bình thường ( đảm bảo nhiệt độ được lấy 1s 1 lần)
- **Motor task** : Mức ưu tiên cao nhất ( đảm bảo motor phải chạy liên tục phản hồi đúng chiều quay với thời gian đã định)
- **Sd card task** : Mức ưu tiên thấp
- **Print task** : Mức ưu tiên thấp
- **Scan key pad task** : Mức ưu tiên cao nhất ( đảm bảo phải phản hồi kịp thời tín hiệu nút bấm từ người dùng )
- **Menu config task** : Mức ưu tiên thấp
- **Pass word task** : Mức ưu tiên thấp

## Tính toán byte stack của các task

Em sử dụng hàm uxTaskGetStackHighWaterMark () hàm này sẽ trả về số lượng word còn lại trong stack của 1 task

Trước hết phải define INCLUDE\_uxTaskGetStackHighWaterMark lên 1 trong FreeRTOSconfig.h

```
#define INCLUDE_uxTaskGetStackHighWaterMark 1
```

Sau khi tính toán em chọn được số lượng word của các stack như sau:

- **Sensor task** : 188 words => chọn uxStackDepth = 250 words
- **Motor task** : 152 words => chọn uxStackDepth = 200 words
- **Sd card task** : 197 words => chọn uxStackDepth = 250 words
- **Print task** : 96 words => chọn uxStackDepth = 150 words
- **Scan key pad task** : 146 words => chọn uxStackDepth = 200 words
- **Menu config task** : 152 words => chọn uxStackDepth = 200 words
- **Pass word task** : 48 words => chọn uxStackDepth = 100 words

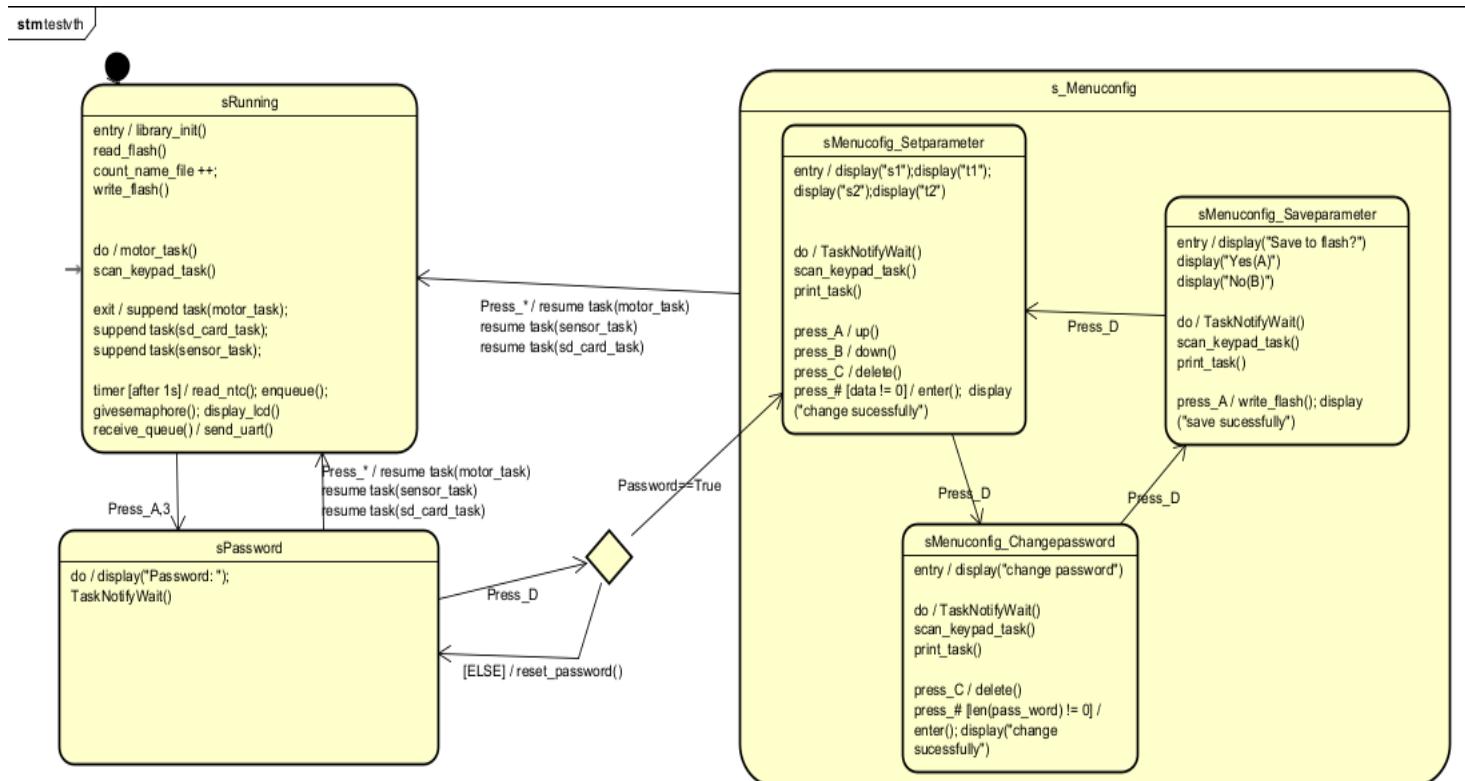
## Thiết kế chương trình

Để thiết kế chương trình em sử dụng state machine. Hệ thống hoạt động với 5 state như sau:

- **sRunning** : Đây là trạng thái bình thường của hệ thống các sensor task, motor task, sd card task và print task sẽ hoạt động ở trạng thái này.
- **sPassword** : Xử lý khi nhập mật khẩu
- **sMenuconfig\_Setparameter** : Xử lý khi muốn thay đổi các thông số s1,t1,s2,t2
- **sMenuconfig\_Changepassword** : Xử lý khi muốn thay đổi pass word
- **sMenuconfig\_Saveparameter** : Xử lý khi muốn lưu các biến vào flash

Và các sự kiện : Nút bấm, timer delay, receive queue, take semaphore,....

Dưới đây là state machine của hệ thống (state\_machine\_testvht.pdf sẽ nhìn được rõ hơn)



## Giải thích chi tiết chương trình và các task

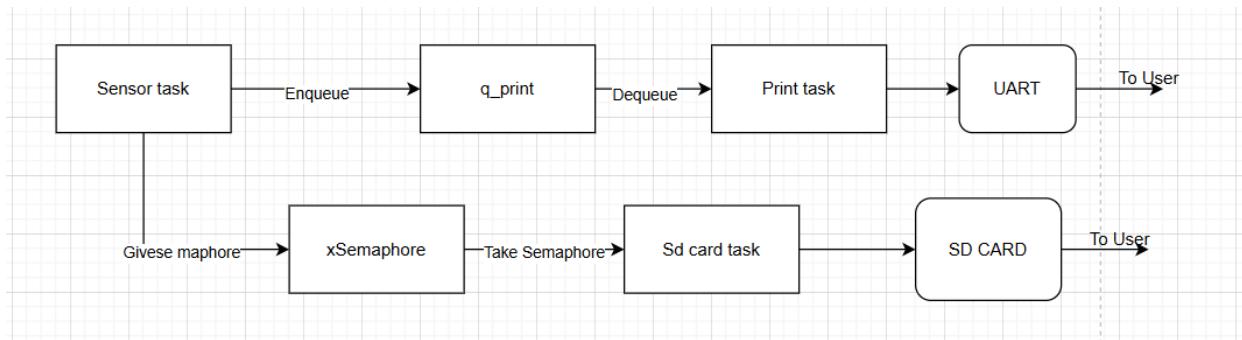
- **trạng thái sRunning:**

Motor task sẽ chạy liên tục

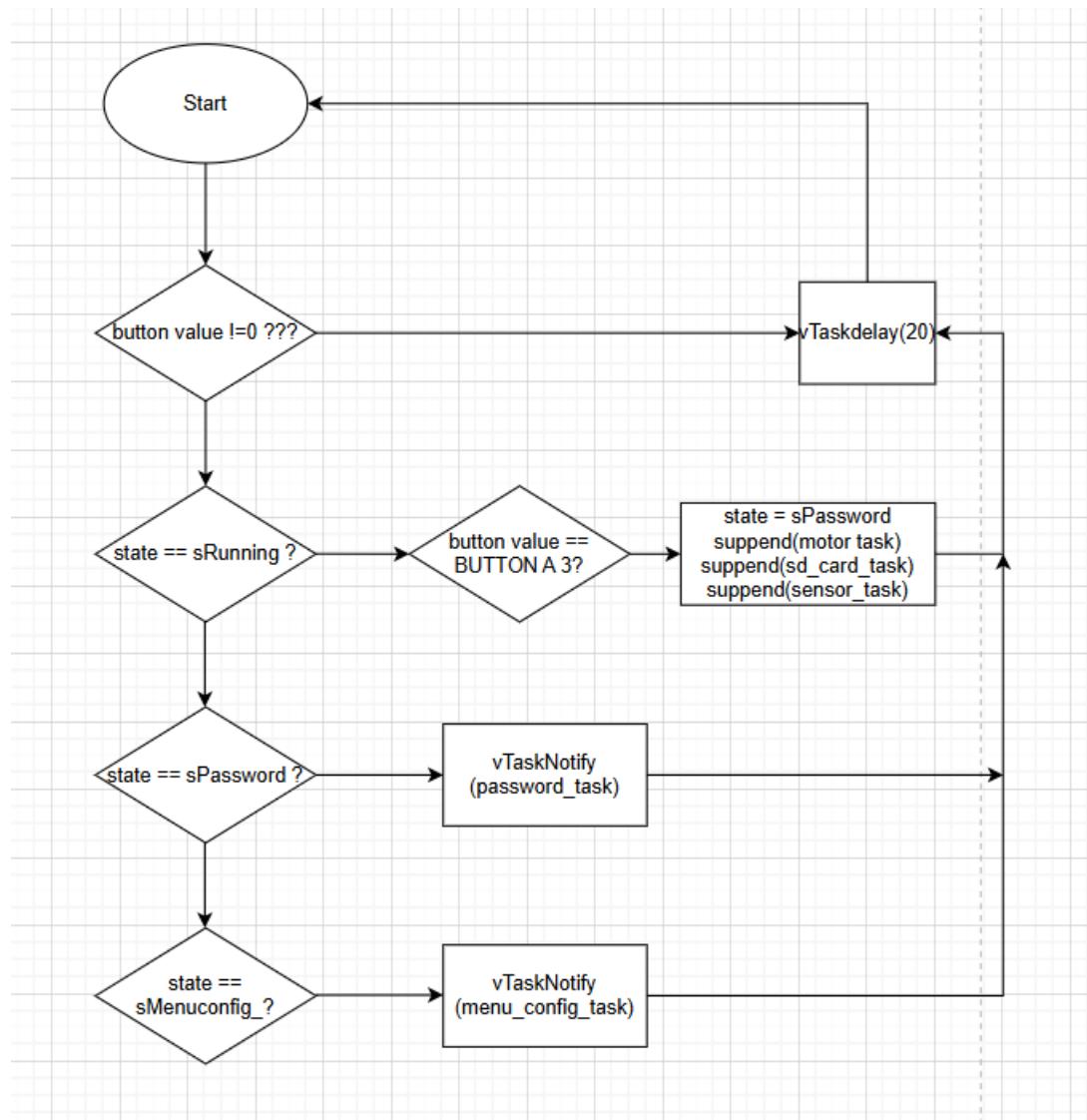
Scan key pad task sẽ chạy với chu kỳ 20ms 1 lần đọc tín hiệu bàn phím mà không gây trễ, đồng thời cũng không làm ảnh hưởng quá nhiều đến hiệu suất hệ thống.

Khi nhận được nút bấm scan key pad task sẽ kiểm tra xem có phải 2 nút A và 3 được nhấn đồng thời hay không. Nếu đúng sẽ chuyển sang trạng thái **sPassword** và **đồng thời Suspend 3 task**: Motor task, Sensor task, và Sd card task. (Các nút khác sẽ không xử lý)

Sensor task sẽ đọc ADC 1 giây 1 lần khi đọc xong nó đồng thời Enqueue và Give semaphore lúc này 2 task sẽ nhận được thông báo và chạy như sơ đồ bên dưới.



#### - Lưu đồ thuật toán của scan\_keypad\_task



Task này sẽ được chạy với chu kỳ 20ms trong cả chế độ running và config. Nó liên tục quét phím nếu phím được nhấn sẽ kiểm tra với các trạng thái tương ứng và gửi notify đến các task .

### - **trạng thái sPassword**

Khi ở trạng thái sPassword hàm scan key pad vẫn được chạy nếu nút bấm được nhấn nó sẽ gửi xTaskNotify đến pass word task kèm theo đó là giá trị của nút nhấn. Tức là task pass word task chỉ chạy khi có notify từ scan key pad task.

Khi nhấn các nút số giá trị của Pass word sẽ được hiển thị

Nhấn button C sẽ xóa 1 giá trị đi

Nhấn D sẽ check giá trị pass word vừa nhập nếu đúng sẽ chuyển sang trạng thái **sMenuconfig\_Setparameter**

Nhấn button \* sẽ thoát khỏi trạng thái sPassword và Resume lại các task Motor task, Sensor task, và Sd card task và quay lại trạng thái sRunning

- **trạng thái sMenuconfig\_Setparameter**

Khi ở trạng thái **sMenuconfig\_Setparameter** hàm scan key pad vẫn được chạy nếu nút bấm được nhấn nó sẽ gửi xTaskNotify đến pass word task kèm theo đó là giá trị của nút nhấn. Tức là task menu config task chỉ chạy khi có notify từ scan key pad task.

Khi bấm button A,B thì con trỏ sẽ di chuyển đi lên và xuống tương ứng với các hàng s1,t1,s2,t2

Khi bấm các nút số thì giá trị số sẽ hiển thị tùy theo vị trí của con trỏ tương ứng  
Khi bấm button C thì giá trị số sẽ bị xóa

Khi bấm button D thì sẽ di chuyển sang trạng thái

**sMenuconfig\_Changepassword**

Khi bấm button \* thì sẽ thoát khỏi chế độ config quay lại trạng thái sRunning và resume lại các task ở state này

Khi bấm button # thì sẽ lưu các giá trị thay đổi s1,t1,s2,t2 nếu các giá trị này khác 0

- **trạng thái sMenuconfig\_Changepassword**

**Khi ở trạng thái sMenuconfig\_Changepassword**

hàm scan key pad vẫn được chạy nếu nút bấm được nhấn nó sẽ gửi xTaskNotify đến pass word task kèm theo đó là giá trị của nút nhấn.

Khi bấm button C thì giá trị số sẽ bị xóa

Khi bấm button \* thì sẽ thoát khỏi chế độ config quay lại trạng thái sRunning và resume lại các task ở state này

Khi bấm button D thì sẽ di chuyển sang trạng thái

Khi bấm button # thì sẽ lưu pass word mới nếu len của nó khác 0

- **trạng thái sMenuconfig\_Saveparameter**

**Khi ở trạng thái sMenuconfig\_Changepassword**

hàm scan key pad vẫn được chạy nếu nút bấm được nhấn nó sẽ gửi xTaskNotify đến pass word task kèm theo đó là giá trị của nút nhấn.

Nếu nhấn button A sẽ lưu các giá trị mới nhất của các thông số s1,s2,t1,t2 và pass word vào flash

## V. Những khó khăn gặp phải

- LCD bị hỏng : trong quá trình làm không hiểu sao LCD của em bị lỗi không thể hiển thị được em fix mãi không được  
=> Lỗi do phần cứng, mua LCD mới 😞
- ADC đọc không ổn định : Do đặt đầu đo của ADC gần các nguồn điện nên giá trị đọc được không ổn định bị nhảy liên tục
  - Flash thiňn thoảng bị lỗi không thể xóa được sector dẫn đến dừng hệ thống  
=> Kiểm tra nếu xóa lỗi thì sẽ xóa lại
  - Sd card giao tiếp không ổn định lúc ghi được lúc không  
=> Em vẫn chưa tìm ra nguyên nhân của lỗi này
- Quản lý các task làm sao cho hiệu quả: Do vẫn còn yếu kinh nghiệm nên em vẫn chưa biết sử dụng Semaphore, Mutex, Task Notify, Queue, Suspend để quản lý hoạt động các task sao cho hiệu quả
  - => Tập trung đọc hiểu các tài liệu và tìm hiểu ưu nhược điểm của từng phương pháp áp dụng vào hệ thống

## VI. Kết quả

- Code : [tuyenXuly/test\\_vht](#)
- Video demo : <https://youtu.be/dvsYLmfiH3o>