ĐẠI HỌC QUỐC GIA

ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

BÁO CÁO BÀI TẬP LỚN

MÔN THIẾT KẾ HỆ THỐNG NHÚNG

ĐỀ TÀI

THIẾT KẾ MÁY ĐO HUYẾT ÁP SỬ DỤNG CẢM BIẾN ÁP SUẤT

LỚP L02 --- NHÓM 05 --- HK241

GVHD: Ths. Nguyễn Trung Hiếu

STT	Sinh viên thực hiện	Mã số sinh viên
1	Huỳnh Kiến Hào	2210857
2	Phan Trần Thế Huy	2211259
3	Nguyễn Thị Ngọc Trân	2213593

Thành phố Hồ Chí Minh - 1/2025

DANH SÁCH THÀNH VIÊN

Danh sách thành viên Nhóm 5

STT	Họ và tên	Mã số sinh viên
1	Huỳnh Kiến Hào	2210857
2	Phan Trần Thế Huy	2211259
3	Nguyễn Thị Ngọc Trân	2213593

LỜI CẢM ƠN

Nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Trung Hiếu, người đã tận tình truyền đạt những kiến thức quý báu từ cơ bản đến nâng cao. Những bài học của thầy không chỉ giúp chúng em xây dựng nền tảng vững chắc mà còn là kim chỉ nam để nhóm hoàn thành bài tập lớn một cách chỉn chu nhất có thể.

Chúng em cũng xin bày tỏ lòng biết ơn sâu sắc đến anh Nguyễn Huy Hoàng, người đã không ngần ngại hỗ trợ, hướng dẫn tận tình trong quá trình thiết kế mạch trên Altium. Nhờ sự giúp đỡ và chỉ bảo hết mình của anh, nhóm mới có thể hoàn thiện mạch in PCB một cách thành công.

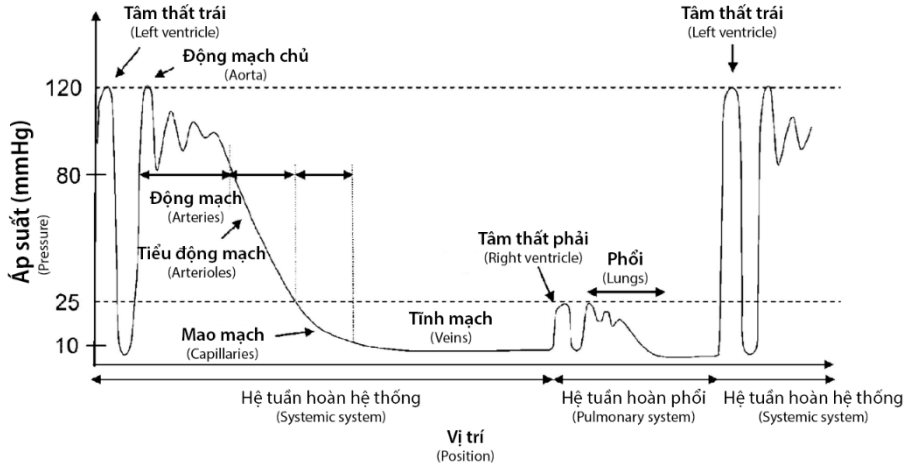
Cuối cùng, bài tập lớn về thiết kế mạch đo huyết áp này không chỉ là một thử thách mà còn là một cơ hội quý giá, mang lại cho chúng em nhiều bài học thực tiễn và kinh nghiệm sâu sắc trong việc thực hiện một đề tài thuộc môn Thiết kế Hệ thống Nhúng. Chúng em xin trân trọng cảm ơn!

MỤC LỤC

CHƯƠNG 1
LÝ THUYẾT VỀ HUYẾT ÁP

1.1. Huyết áp

Áp suất máu cần thiết để tống máu đi. Hình vẽ sau là đồ thị áp suất máu động mạch và tĩnh mạch tại các vị trí khác nhau của chu kỳ tuần hoàn, được ghi lại với một người đang nằm ngang.



Hình 1.1. Huyết áp dọc theo hệ tuần hoàn của một người nằm ngang.

Dao động của huyết áp dọc theo động mạch chính trong tuần hoàn hệ thống phản ánh dao động của áp suất của máu khi nó rời khỏi động mạch chủ, tại mức áp suất giữa 80mmHg ($P_{diastole}$), là huyết áp tâm trương, xảy ra khi tim nghỉ giữa các nhịp đập và máu vẫn tiếp tục lưu thông trong động mạch và 120mmHg ($P_{systole}$), là huyết áp tâm thu, xảy ra khi tim co bóp và bơm máu vào các động mạch.

1.2. Phương pháp đo huyết áp không xâm lấn

Đo biên độ dao động xuất hiện trong kỹ thuật đo áp suất vòng quấn, được tạo ra bởi sự giãn nở của thành động mạch mỗi khi máu được bơm qua động mạch.

Các đặc tính cụ thể của thể tích không khí nén trong vòng quấn được sử dụng để xác định và cảm nhận các giá trị huyết áp.

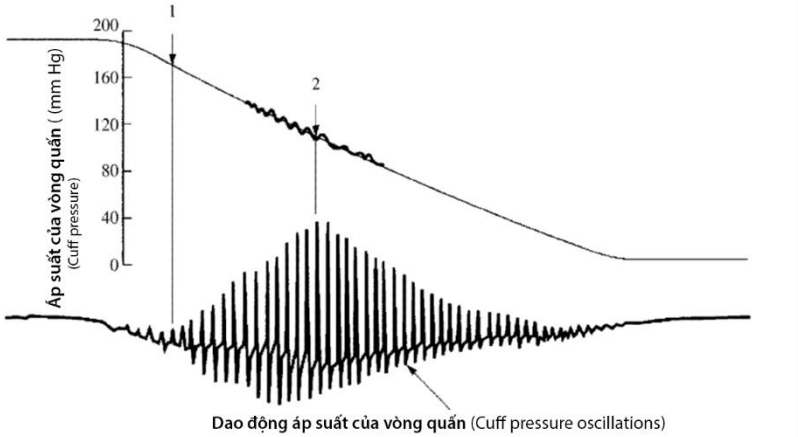
Tín hiệu áp suất vòng quấn tăng cường độ trong vùng huyết áp tâm thu, đạt cực đại khi áp suất vòng quấn bằng với áp suất động mạch trung bình. Khi áp suất vòng quấn giảm xuống dưới điểm này, cường độ tín hiệu giảm tỷ lệ với tốc độ giảm áp suất không khí trong vòng quấn.

Áp suất động mạch trung bình là chỉ số huyết áp đơn lẻ có độ chính xác cao nhất, so với huyết áp tâm thu và huyết áp tâm trương, vì nó được đo khi dao động áp suất vòng quấn đạt biên độ lớn nhất.

Khi áp suất vòng quấn được tăng lên mức cao hơn huyết áp tâm thu, có thể quan sát thấy mạch quay biến mất. Áp suất vòng quấn cao hơn mức huyết áp tâm thu khiến động mạch dưới vòng quấn bị tắc hoàn toàn. Tuy nhiên, ở các mức áp suất trên huyết áp tâm thu, những dao động áp suất nhỏ vẫn xuất hiện trong áp suất vòng quấn do các nhịp đập của động mạch dưới mép trên của vòng quấn, và các dao động này được truyền qua vòng quấn thông qua các mô lân cận.

Khi áp suất vòng quấn giảm chậm, ngay khi áp suất vòng quấn xuống dưới huyết áp tâm thu, máu bắt đầu tràn qua động mạch và các dao động trong áp suất vòng quấn trở nên lớn hơn. Áp suất "hơi cao hơn" huyết áp tâm thu được phát hiện bằng cách xác định sự thay đổi từ các dao động có biên độ nhỏ khi áp lực của vòng quấn "hơi cao hơn" huyết áp tâm thu và khi áp lực vòng quấn bắt đầu tăng biên độ (điểm 1). Khi vòng quấn tiếp tục xả hơi, biên độ của các dao động tăng lên, đạt mức cực đại, sau đó giảm dần khi áp suất vòng quấn giảm về 0. Điểm 2 là biên độ dao động vòng quấn lớn nhất, tương ứng với áp suất động mạch trung bình thực sự.

Vì không có sự chuyển đổi rõ ràng trong biên độ dao động khi áp suất vòng quấn xuống thấp hơn huyết áp tâm trương, các phương pháp thuật toán được sử dụng để dự đoán huyết áp tâm trương.



Hình 1.2. ***Một vòng quấn được bơm căng lên trên mức áp suất tâm thu và sau đó từ từ xả hơi****. Áp suất tâm thu được phát hiện (điểm 1), khi có sự chuyển đổi từ dao động biên độ nhỏ (trên mức áp suất tâm thu) sang biên độ dao động của vòng tăng dần. Dao động áp suất của vòng tăng lên mức tối đa (điểm 2) tại mức áp suất động mạch trung bình.*

CHƯƠNG 2

ĐẶC TẢ SẢN PHẨM (REQUIREMENTS)

2.1. Tên (Name)

Tên đề tài: Máy đo huyết áp sử dụng cảm biến áp suất

2.2. Mục đích (Purpose)

Đo và kiểm tra huyết áp của người dùng và xem mức huyết áp có nằm ngoài ngưỡng bình thường hay không?

2.3. Đầu vào và đầu ra (Inputs and Outputs)

Inputs	Outputs
- Cổ tay hoặc cánh tay người. - Cảm biến áp suất: để đo huyết áp. - Nguồn điện: Pin. - Nút nhấn start 1 (người bình thường) - Nút nhấn start 2 (người nghi ngờ huyết áp bất thường) - Nút nhấn stop	- Màn hình LCD chỉ số huyết áp tâm thu và huyết áp tâm trương của người đang đo, hiển thị dưới dạng mmHg (milimet thủy ngân). - Đèn LED đỏ nhằm cảnh báo sự bất thường của huyết áp. Cụ thể: + Đèn tắt: huyết áp bình thường. + Đèn bật: huyết áp cao,…

2.4. Tình huống sử dụng (Use cases)

Chế độ tự động tắt: Hệ thống tự động tắt sau một khoảng thời gian không sử dụng để tiết kiệm pin. Nếu không có hoạt động trong vòng 3 phút, máy sẽ tắt tự động.

Cảnh báo huyết áp cao/thấp: Máy sẽ cảnh báo khi huyết áp vượt quá các ngưỡng an toàn. Cụ thể:

Huyết áp cao: Nếu huyết áp đo được 130-139 mmHg (huyết áp tâm thu) hoặc 85-90 mmHg (huyết áp tâm trương), đèn LED màu đỏ sẽ sáng lên để cảnh báo.

Huyết áp thấp: Nếu huyết áp đo được thấp hơn 85 mmHg (huyết áp tâm thu) hoặc 60 mmHg (huyết áp tâm trương), đèn LED màu đỏ sẽ sáng lên để cảnh báo.

Điều chỉnh ngưỡng siết: Mỗi người đều có thể trạng cơ thể và tình trạng huyết áp khác nhau. Do đó, cần điều chỉnh ngưỡng siết cho phù hợp.

Thủ công – thông qua việc điều chỉnh kích thước của vòng quấn: người dùng có thể nới rộng hoặc thắt chặt vòng quấn trước khi bắt đầu đo để đảm bảo tính chính xác và thoải mái. Nếu không có dụng cụ để đo, có thể dùng ngón tay để kiểm tra; ở kích thước chuẩn vòng quấn chỉ có thể luồn hai ngón tay vào dưới mép của vòng quấn.

Loại vòng quấn	Chu vi cánh tay (cm)	Chiều rộng vòng quấn (cm)	Chiều dài vòng quấn (cm)
Trẻ sơ sinh	<6	3	6
Trẻ nhỏ	6-15	5	15
Trẻ em	16-21	8	21
Người lớn	22–26	10	24
	27–34	13	30
	35–44	16	38
Đo ở đùi bắp chân người lớn	45–52	20	42

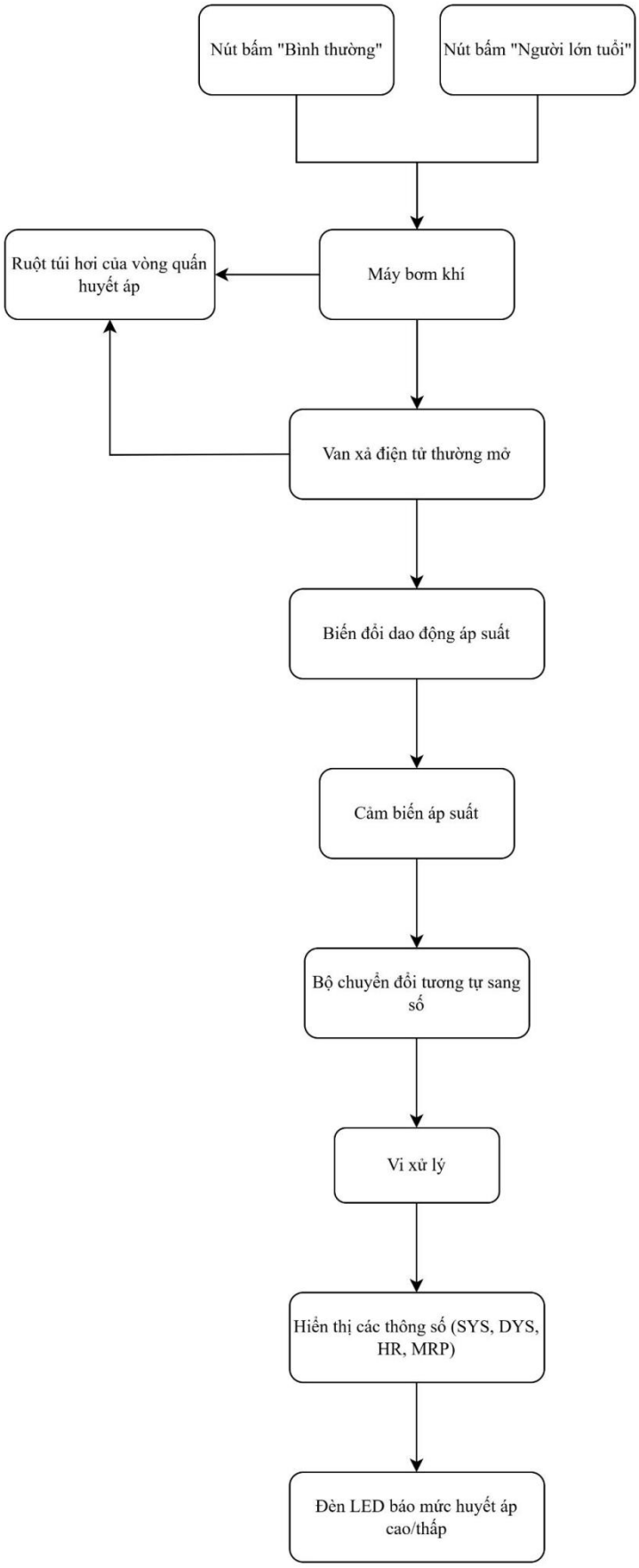
Bảng 2.4. Kích thước vòng quấn được Hiệp hội Tim mạch Hoa Kỳ khuyến nghị.

Tự động – thông qua 2 nút điều chỉnh áp suất không khí bơm vào vòng quấn: có hai nút nhấn với hai ngưỡng siết khác nhau:

+ Nút xanh: ngưỡng siết dành cho người có huyết áp bình thường. Áp suất khi bơm không khí vào vòng quấn sẽ giao động từ 150-160 mmHg.

+ Nút đỏ: ngưỡng siết dành cho người có tình trạng huyết áp cao. Áp suất khi bơm không khí vào vòng quấn sẽ đạt 200 mmHg (trong ngưỡng cho phép của tổ chức Y tế quốc tế là <250mmHg).

2.5. Chức năng (Functions)



Hình 2.5. Sơ đồ khối chức năng hệ thống máy đo huyết áp bằng cảm biến áp suất

2.6. Hiệu suất (Performance)

Khía cạnh quan trọng nhất của một hệ thống giám sát huyết áp là độ chính xác.

Độ chính xác: huyết áp sai số 10mmHg.

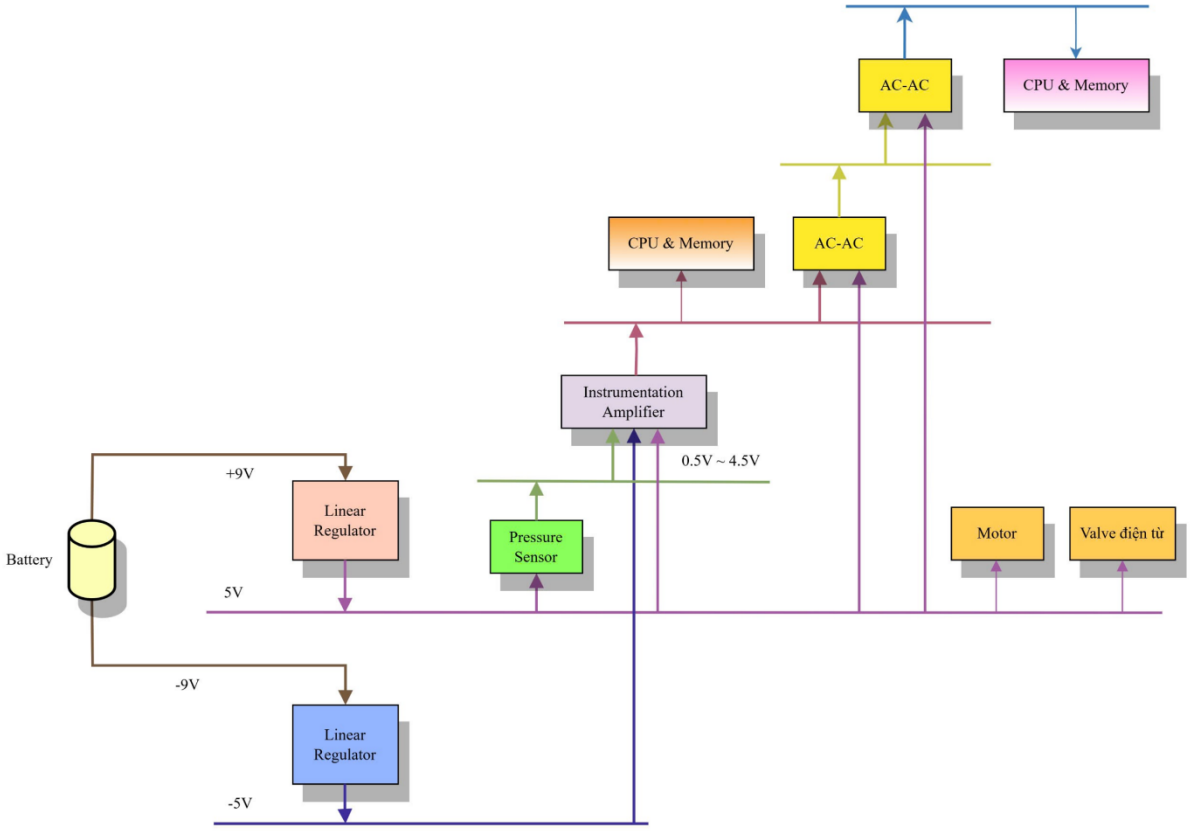
Hiệu suất năng lượng là rất quan trọng để kéo dài tuổi thọ pin hoặc đảm bảo hoạt động liên tục khi kết nối với nguồn điện.

2.7. Chi phí sản xuất (Manufacturing costs)

Giá thành dao động khoảng từ 300k - 1tr VND. Bao gồm các linh kiện: vi xử lý, cảm biến áp suất, bộ khuếch đại, màn hình LCD, LED, các nút nhấn,…

2.8. Nguồn cung cấp (Power)

Đây chỉ là thiết kế tham khảo, thiết kế thật sử dụng dựa trên dạng này.



Hình 2.8. Cây phân phối nguồn cho các khối trong máy đo huyết áp

2.9. Kích thước và trọng lượng (Physical size/weight)

Trọng lượng của máy dưới 1kg, gọn nhẹ, kích thước nhỏ vì các linh kiện sử dụng chiếm diện tích không nhiều, dễ dàng gấp gọn (vòng quấn và ống dẫn khí).

2.10. Lắp đặt (Installation)

Thiết bị máy đo huyết áp được thiết kế dưới dạng hộp nhựa trong suốt nhỏ gọn để vừa túi xách dễ dàng mang theo, ước tính khoảng *15 x 10 x 5 cm* và trọng lượng dưới *300g*, bên trong chứa toàn bộ các thành phần chính như vi xử lý, cảm biến áp suất, mạch khuếch đại, các van xả và bộ chuyển đổi tín hiệu. Bên ngoài của hộp được gắn màn hình LED hiển thị kết quả, cùng với các nút điều khiển và đèn báo.

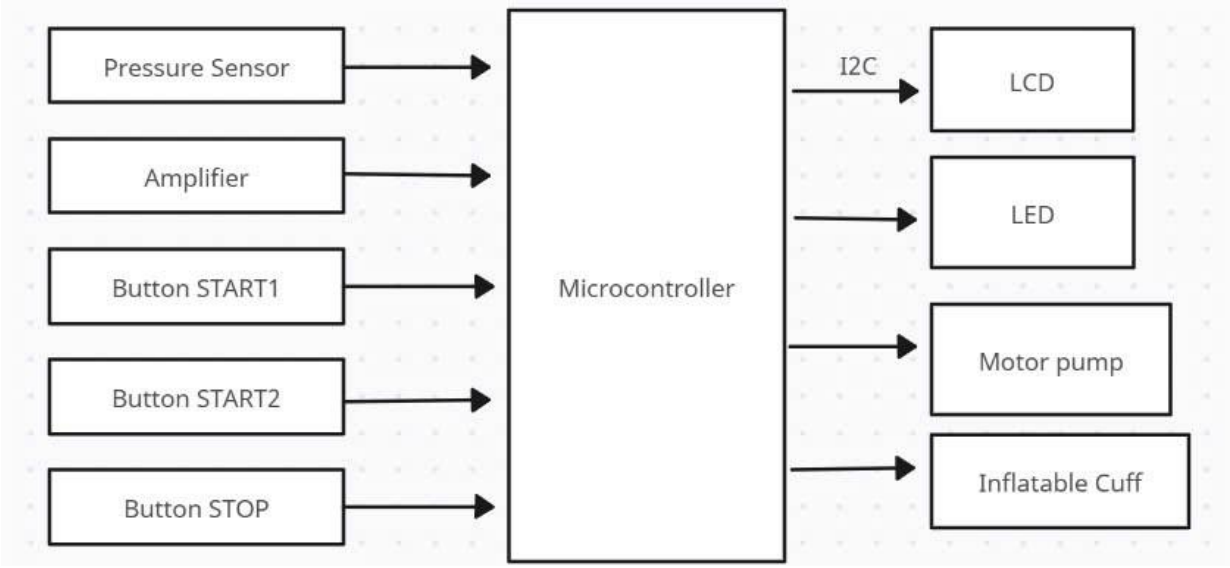
Thiết bị còn bao gồm một ống dẫn khí nối từ bao khí huyết áp đến hộp nhựa. Bao khí huyết áp được bọc bởi băng vải có khả năng điều chỉnh linh hoạt, giúp người dùng tự điều chỉnh độ chặt hoặc lỏng của bao quanh tay, đảm bảo sự thoải mái và độ chính xác khi đo huyết áp. Việc lắp đặt băng vải và bao quấn huyết áp rất đơn giản, giúp người dùng dễ dàng sử dụng mà không cần sự hỗ trợ từ bên ngoài.

CHƯƠNG 3

ĐẶC TẢ KỸ THUẬT (SPECIFICATIONS)

3.1. Các khối trong thiết kế

3.1.1. Sơ đồ khối



Hình 3.1.1. Sơ đồ khối của hệ thống

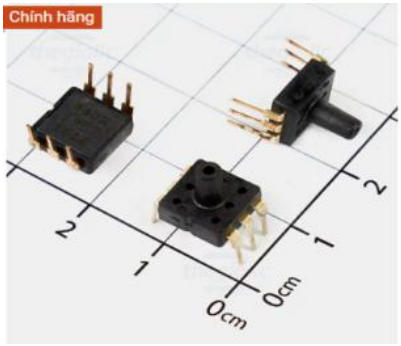
3.1.2. Giao tiếp giữa các thành phần

Quá trình bắt đầu bằng việc bơm không khí nén từ động cơ vào vòng bít bơm hơi (Inflatable cuff). *Áp suất không khí được thiết lập và kiểm soát từ chương trình trên vi điều khiển*. Trong quá trình bơm hơi này, cảm biến áp suất (Pressure Sensor) sẽ đo áp suất bên trong vòng bít và tạo ra tín hiệu đầu ra cho thành phần DC.

Đầu ra này sau đó được khuếch đại bởi một loại khuếch đại nhất định, cụ thể là op-amp có độ chính xác cao. Sau khi áp suất không khí đạt đến mức tối đa (giả sử đã được cài đặt trong chương trình), động cơ sẽ ngừng bơm không khí, và không khí đã được nén sẽ được xả từ từ ra khỏi vòng bít. Quá trình này được gọi là quá trình xẹp, trong đó vi điều khiển sẽ phân tích đầu ra để xác định huyết áp tâm thu và huyết áp tâm trương.

3.1.3. Chọn cảm biến áp suất (Pressure Sensor)

Cảm biến áp suất được sử dụng như một thành phần để phát hiện áp suất không khí đi qua nó.

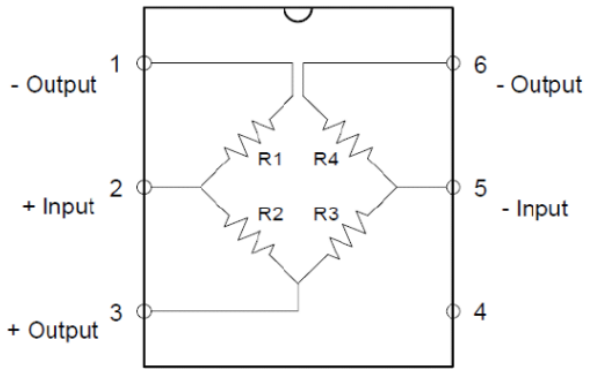


Hình 3.1.3.1. Cảm biến áp suất MPS20N0040D-D.

Không khí được nén từ động cơ bơm vào vòng bít phồng lên và đi qua thành phần này. Ban đầu, thành phần này sẽ tạo ra hai loại đầu ra, đó là tín hiệu DC và tín hiệu AC, nhưng chỉ có đầu ra DC sẽ được sử dụng và đo lường vì chỉ có nguồn DC để cấp nguồn. Đầu ra sẽ được khuếch đại bằng bộ khuếch đại hoạt động, điều này sẽ được giải thích trong phần tiếp theo. Lý do để khuếch đại tín hiệu đầu ra là vì nó quá nhỏ, khiến cho việc phát hiện và đo lường bởi vi điều khiển trở nên khó khăn.

Đại lượng	**Giá trị**
Điện áp hoạt động	5 (VDC)
Ngõ ra	0.5-4.5 (V)
Phạm vi đo	0-40 (kPa)
Trở kháng	4-6 (KΩ)

Bảng 3.1.3. Thông số kỹ thuật của cảm biến áp suất MPS20N0040D-D.



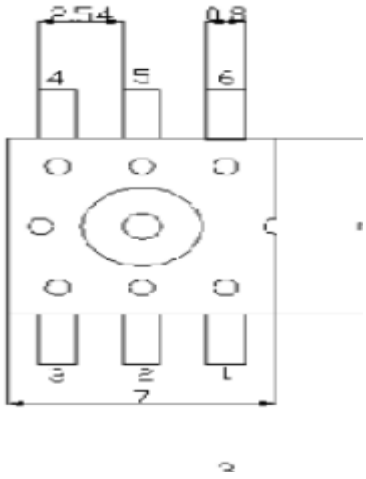
Hình 3.1.3.2. Sơ đồ mạch điện cảm biến áp suất MPS20N0040D-D.

Mô tả: Đây là loại cảm biến được thiết kế dạng cầu điện trở Wheatstone. Khi có áp suất khí bơm vào cuff đi đến cảm biến hoặc có một áp suất dòng chảy máu tác động lên cuff được ghi nhận lại trong cảm biến, nó làm biến dạng màng cảm biến, qua đó làm dãn hoặc nén các điện trở, gây ra sự thay đổi tín hiệu vi sai điện áp ở ngõ ra.

Ngoài ra, dải áp suất đo được của nó khoảng từ 0 đến 300mmHg nên rất phù hợp cho việc đo huyết áp. Một sự thay đổi nhỏ trong dao động của máu cũng được thể hiện ở điện áp vi sai ngõ ra.

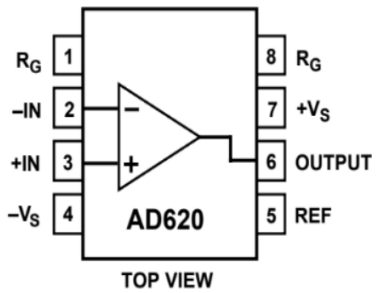
Theo như hình trên, ta có cách kết nối cảm biến như sau:

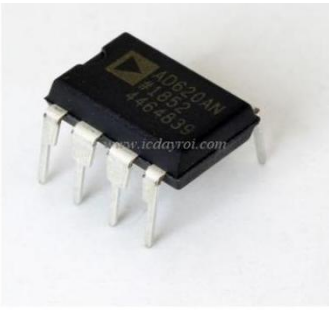
Chân – output được nối tắt với nhau để trở thành 1 cầu điện trở. Đưa điện áp ở ngõ vào là 5V ở chân + input, nối đất chân – input. Điện áp vi sai ở ngõ ra lúc này là điện áp giữa 2 chân + output và – output.



3.1.4. Chọn Amplifier (Bộ khuếch đại)

Bộ khuếch đại hoạt động là một loại bộ khuếch đại vi sai, trong đó điện thế đầu ra thường được nhân với một giá trị nhất định để lớn hơn điện thế đầu vào và được gọi là độ khuếch đại. Trong đề tài này, yêu cầu cho op-amp là tiêu thụ điện năng thấp, độ chính xác cao và độ ồn đầu ra thấp. Điều này là vì, giống như các thiết bị đo lường khác, dự án này phải có độ chính xác cao để duy trì sự ổn định và độ tin cậy. Do đó, nhóm chọn IC khuếch đại AD620AN.





Hình 3.1.4.1 & 3.1.4.2 Sơ đồ mạch điện và ảnh thực tế IC khuếch đại AD620AN.

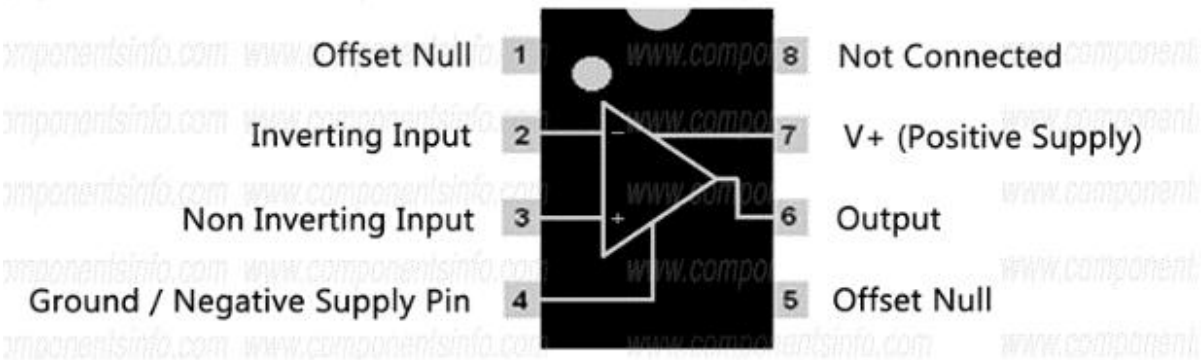
Điện áp cấp nguồn - Tối đa:	36 V

Điện áp cấp nguồn - Tối thiểu:	4.6 V
Dòng cấp nguồn vận hành:	1.3 mA
Nhiệt độ làm việc tối thiểu:	- 55 C
Nhiệt độ làm việc tối đa:	+ 125 C
Kiểu gắn:	Đục lỗ (Through Hole)
Đóng gói / Vỏ bọc:	PDIP-8
Đóng gói:	Tube
Loại bộ khuếch đại:	Khuếch đại dụng cụ (Instrumentation Amplifier)
Nhãn hiệu:	Thiết bị tương tự (Analog Devices)
Điện áp hai nguồn cấp:	+/- 2.3 V đến +/- 18 V
Độ khuếch đại V/V:	1 V/V đến 10000 V/V

Bảng 3.1.4. Thông số kỹ thuật của IC khuếch đại AD620AN

3.1.5. LM741

LM741 là một bộ khuếch đại đa năng. Chúng ta có thể sử dụng nó khuếch đại đảo hoặc không đảo.



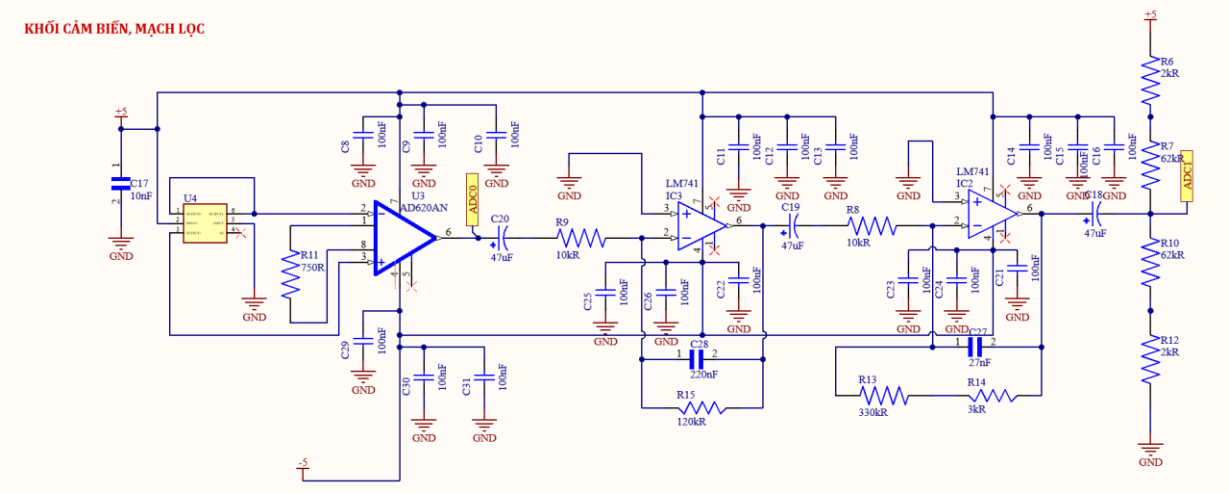
Hình 3.1.5.1. Sơ đồ chân LM741

Số chân	Tên chân	Thông tin chi tiết
1	(Offset N1)	Dùng để điều chỉnh offset cho tín hiệu đầu vào
2	Đầu vào đảo	Nhận tín hiệu điện áp trong mạch khuếch đại đảo
3	Đầu vào không đảo	Nhận tín hiệu điện áp trong mạch khuếch đại không đảo

4	-Vcc	Là chân tham chiếu đất hoặc một nguồn tín hiệu tham chiếu khác
5	Offset N2	Dùng để điều chỉnh offset cho tín hiệu đầu vào
6	Đầu ra	Là tín hiệu đầu ra của mạch khuếch đại
7	+ Vcc	Chân cấp nguồn điện áp dương
8	NC	Chân không cần kết nối

Bảng 3.1.5. Bảng liệt kê dưới các mô tả của các chân

Thiết kế một bộ lọc thông dải bậc 2 dựa trên 2 opamp LM741:



Hình 3.1.5.2. Khối xử lý tín hiệu

Bộ lọc tầng 1 có:

- Tần số cắt thấp: $f_{low} = \frac{1}{2\pi.47.10^{-6}.10.10^{3}} = 0.338(Hz)$

- Tần số cắt cao: $f_{high} = \frac{1}{2\pi.220.10^{-9}.120.10^{3}} = 6.029(Hz)$

- Độ lợi tầng 1: $G_1 = -\frac{120.10^{3}}{10.10^{3}} = -12$

Bộ lọc tầng 2 có:

- Tần số cắt thấp: $f_{low} = \frac{1}{2\pi.47.10^{-6}.10.10^{3}} = 0.338(Hz)$

- Tần số cắt cao: $f_{high} = \frac{1}{2\pi.27.10^{-9}.333.10^{3}} = 17.7(Hz)$

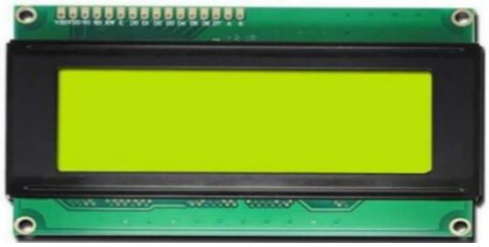
- Độ lợi tầng 2: $G_1 = -\frac{333.10^3}{10.10^3} = -33.3$

Độ lợi sau cùng của 2 tầng là : $G = G_1.G_2 = -12.(-33.3) = 399.6$

Dải thông cuối cùng mà bộ lọc này lọc được là $[0.338Hz : 6.03Hz]$. Việc thiết kế cho bộ lọc này lọc được tín hiệu trong khoảng tần số này là do tín hiệu nhịp tim của con người dao động trong khoảng 0.5 – 2Hz. Nó có thể loại bỏ nhiễu từ các tín hiệu có tần số nằm ngoài vùng của nhịp tim, được mở rộng thêm một khoảng để chắc chắn rằng tần số nhịp tim đo được nằm trong khoảng này.

3.1.6. Chọn màn hình LCD

Màn hình đóng vai trò giúp người dùng dễ dàng quan sát chính xác các số liệu huyết áp tâm trương và huyết áp tâm thu. Nhóm quyết định chọn màn hình LCD1602 cho thiết kế này vì có thể hiển thị rõ ràng 2 thông số huyết áp ở 2 dòng và 16 ký tự mỗi dòng.



Hình 3.1.6.1. Ảnh thực tế màn hình LCD1602.

Điện áp hoạt động	5VDC
Dòng điện tiêu thụ	350uA - 600uA
Nhiệt độ hoạt động	30°C đến 75°C
Kích thước	96 x 60 mm
Màu màn hình	Xanh lá
Màu kí tự hiển thị	Đen
Điều khiển đèn LED nền	Biến trở hoặc PWM
Điều khiển kí tự hiển thị	6 chân tín hiệu
Ngôn ngữ hỗ trợ	Tiếng Anh và Tiếng Nhật
Giao tiếp	SPI, I2C

Bảng 3.1.5. Thông số kỹ thuật của màn hình LCD1602.

* Giao tiếp LCD và I2C

Khối PCF8574 gắn với LCD giúp giao tiếp I2C với MCU. Trên đó có 3 chân được dùng để chọn địa chỉ I2C cho LCD là A2A1A0.

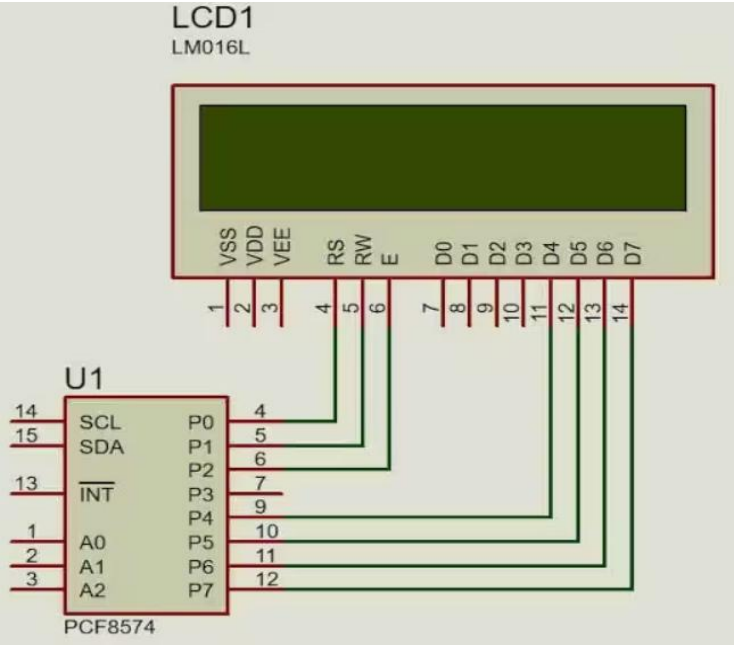
Trên module PCF8574 này, 3 chân được kết nối lên mức cao, tức là A2A1A0 = 111, đối chiếu với bảng address reference trong datasheet của PCF8574, địa chỉ của LCD là 4E (Write) và 4F (Read).

Giao tiếp giữa LCD và PCF8574 là giao tiếp song song 4 bit data, được gửi đi bằng cách gửi từng nibble cao và thấp.

7.3.3 Address Reference

INPUTS			I²C BUS TARGET 8-BIT READ ADDRESS	I²C BUS TARGET 8-BIT WRITE ADDRESS
A2	A1	A0		
L	L	L	65 (decimal), 41 (hexadecimal)	64 (decimal), 40 (hexadecimal)
L	L	H	67 (decimal), 43 (hexadecimal)	66 (decimal), 42 (hexadecimal)
L	H	L	69 (decimal), 45 (hexadecimal)	68 (decimal), 44 (hexadecimal)
L	H	H	71 (decimal), 47 (hexadecimal)	70 (decimal), 46 (hexadecimal)
H	L	L	73 (decimal), 49 (hexadecimal)	72 (decimal), 48 (hexadecimal)
H	L	H	75 (decimal), 4B (hexadecimal)	74 (decimal), 4A (hexadecimal)
H	H	L	77 (decimal), 4D (hexadecimal)	76 (decimal), 4C (hexadecimal)
H	H	H	79 (decimal), 4F (hexadecimal)	78 (decimal), 4E (hexadecimal)

Hình 3.1.6.2. Datasheet address PCF8574



Hình 3.1.6.2. Hình ảnh giao tiếp LCD và I2C

3.1.7. Vòng quấn (Inflatable cuff)

Vòng quấn là một phần không thể thiếu trong việc đo huyết áp, thường được đặt vừa vặn quanh cổ tay hoặc cánh tay trên, ở độ cao gần giống như tim trong khi người được đo ngồi với cánh tay được hỗ trợ. Việc chọn kích thước băng đo huyết áp phù hợp cho người cần đo là rất quan trọng. Khi một vòng quấn quá nhỏ dẫn đến áp lực quá cao, trong khi một vòng quấn quá lớn dẫn đến áp lực quá thấp, vì vậy nó có bốn kích cỡ, dành cho trẻ em cho đến người lớn béo phì. Ngoài ra, nó được làm từ một chất liệu không co giãn, và phần ống tay được sử dụng lớn hơn khoảng 20% so với cánh tay.

Hình 3.1.6. Hình ảnh minh họa vòng quấn.

3.1.8. Động cơ bơm không khí (motor pump)

Động cơ bơm khí 370 Air Pump Motor 12VDC có kích thước nhỏ gọn được sử dụng để bơm, thổi khí với độ ồn thấp.

Hình 3.1.7. Ảnh thực tế động cơ bơm khí 370. Air Pump Motor 12VDC.

Tên model	370 Air Pump Motor 12VDC
Điện áp sử dụng	Max 12VDC

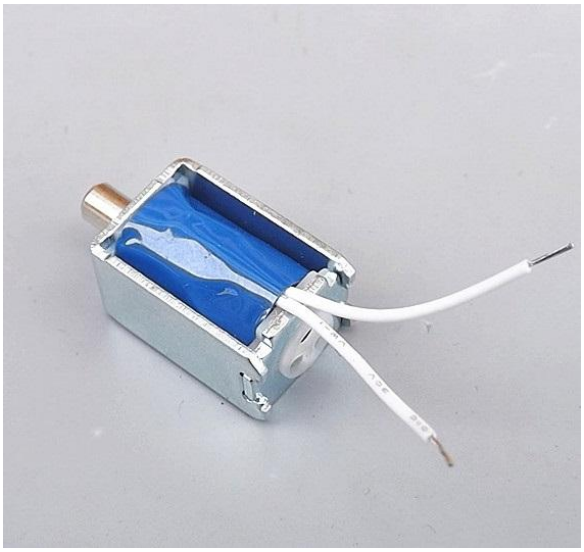
Chức năng	Thổi khí thông qua 1 đầu ra
Dòng điện tiêu thụ	< 600mA
Lưu lượng khí bơm	0.5~1.5 LPM
Độ ồn	< 65dB

Bảng 3.1.7. Thông số kỹ thuật của động cơ bơm khí 370.

Cần cấp cho motor pump nguồn 5V, dòng 17mA

3.1.9. Valve đóng mở khí

Sử dụng van để đóng/mở dòng khí thường áp dụng trong trường hợp đo huyết áp không xâm lấn với kỹ thuật bơm và xả khí để điều chỉnh áp suất trong vòng bít (cuff).



Hình 3.1.9. Valve đóng mở khí

Valve loại thưởng mở, cấp cho valve nguồn 5V, dòng 44mA

3.1.10. Nút nhấn 4 chân

Cần 3 nút nhấn START1, START2, STOP để thực thi 3 chức năng tương ứng

*Thông số kỹ thuật

- Số chân: 4 chân.
- Dòng điện chịu đựng: 2A.
- Màu sắc: vàng.
- Hình dạng: vuông.
- Kích thước 12 x 12 x 7.3mm

Hình 3.1.10. Nút nhấn 4 chân

3.2. Chọn thông số cho bộ ADC từ cảm biến áp suất

Cảm biến áp suất MPS20N0040D-D

Phạm vi đo áp suất	0 - 40kpa (tương đương 0 - 300 mmHg)
Điện áp đầu ra toàn thang đo (full scale output voltage)	50 - 100 mV
Nguồn điện	5 VDC hoặc dòng điện không đổi 1mA
Độ chính xác tuyến tính	0.25% FS
Trở kháng đầu vào	4 - 6 kΩ
Trở kháng đầu ra	4 - 6 kΩ
Độ tuyến tính	0,3% F.S
Độ trễ	0.7% F.S
dải điện áp đầu ra	50 mV đến 100 mV ⇔ 0 kPa đến 40 kPa (áp suất tối đa tương đương với 300 mmHg)

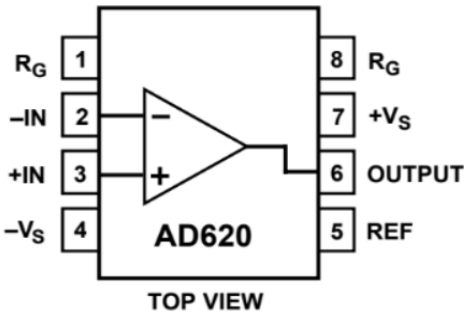
Bảng 3.2.1. Thông số kỹ thuật về tín hiệu của cảm biến áp suất MPS20N0040D-D.

Tầm đo huyết áp**:** 0 mmHg đến 300 mmHg.

Độ tuyến tính (sự thay đổi điện áp cho mỗi mmHg) là:

$$\frac{\Delta mV}{\Delta mmHg} = \frac{50\ mV}{300\ mmHg} = 0.167\ mV / mmHg$$

Từ các thông tin trên, để khuếch đại tín hiệu từ 0-50 mV (từ cảm biến) thành tín hiệu có thể xử lý bởi ADC với mức điện áp tham chiếu (Vref). Nhóm quyết định sử dụng IC khuếch đại ADC620A.



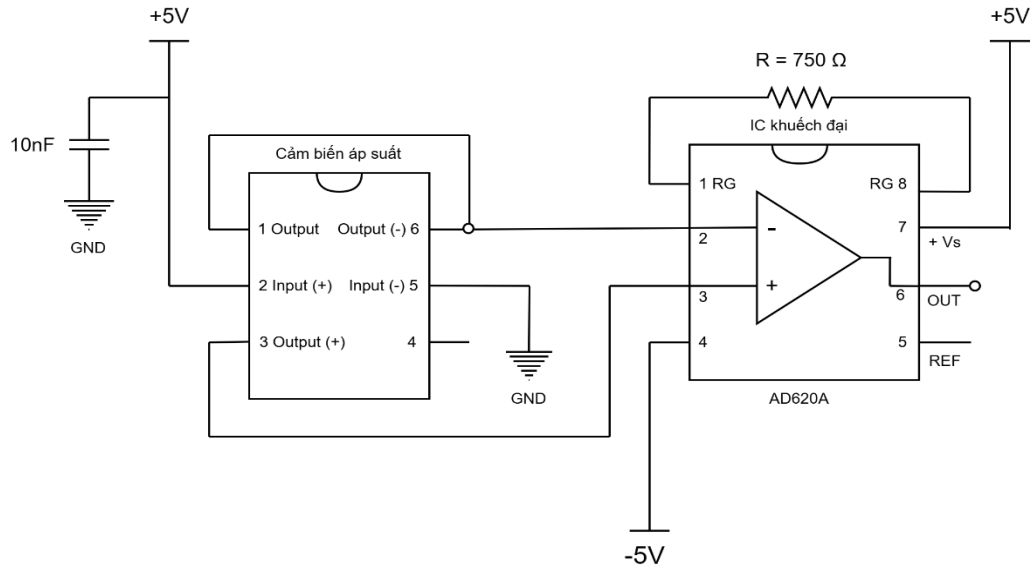
Hình 3.2.2. Sơ đồ chân IC khuếch đại ADC620A 12VDC.

Với tổng sự thay đổi điện áp của cảm biến là 50mV, tổng sự thay đổi điện áp mong muốn sau khi khuếch đại là 50mV.

⇨ *Hệ số khuếch đại cần là:* $\frac{3.3V}{50\ mV} = 66$.

Điện trở được sử dụng để xác định độ lợi của bộ khuếch đại theo phương trình là:

$$R_G = \frac{49.4\ k\Omega}{G-1} = \frac{49.4\ k\Omega}{66-1} = 760\Omega$$ => chọn 750Ω.



Hình 3.2.3. Sơ đồ mạch điện kết nối giữa cảm biến áp suất và IC khuếch đại.

Điện áp đầu ra của cảm biến cho mỗi 1 mmHg là 0.167 mV**.**

Sau khi khuếch đại

=> $V_{out\_khuếch\ đại}$ (1mmHg)=0.167 mV/mmHg × 66 = 11.022 (mV/mmHg).

Độ phân giải của ADC được tính bằng: Độ phân giải $= \frac{V_{ref}}{2^N}$.

Trong đó:

- Vref: Điện áp tham chiếu của ADC
- N: số bit của ADC

Với tín hiệu đầu ra sau khi khuếch đại là từ 0V đến 3.3V, thì Vref có thể chọn là 3.3V để tận dụng tối đa dải đo của ADC.

Chọn số bit N của ADC

- N = 8 bit → Độ phân giải $= \frac{V_{ref}}{2^N} = \frac{3.3V}{2^8} = 12.8\ mV$.

=> 12.8 mV không thể phân biệt sự thay đổi 11.022 mV cho 1mmHg.

- N = 10 bit → Độ phân giải $= \frac{V_{ref}}{2^N} = \frac{3.3V}{2^{10}} = 3.22\ mV$.

=> Độ phân giải vừa đủ.

- N = 12 bit → Độ phân giải $= \frac{V_{ref}}{2^N} = \frac{3.3V}{2^{12}} = 0.806\ mV$.

=> Độ phân giải nhỏ, đảm bảo kết quả đo chính xác hơn.

3.3. Chọn vi điều khiển

3.3.1. Giới thiệu STM32F103C8T6

STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng như F0,F1,F2,F3,F4….. Stm32f103 thuộc họ F1 với lõi là ARM COTEX M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.

STM32F103C8T6 là bo một mạch phát triển sử dụng MCU STM32F103C8T6 lõi ARM STM32. Bo mạch này phát triển hệ thống tối thiểu chi phí thấp, được thiết kế nhỏ gọn, hoạt động vô cùng ổn định, các chân ngoại vi được đưa ra ngoài giúp dễ dàng kết nối, giao tiếp. Bo mạch phù hợp cho người học muốn tìm hiểu vi điều khiển STM32 với lõi ARM Cortex-M3 32-bit.

3.3.2. Đặc điểm nổi bật

Chế độ tiết kiệm năng lượng: Hỗ trợ nhiều chế độ tiết kiệm năng lượng như Sleep, Stop, và Standby, giúp tối ưu hóa tiêu thụ điện năng cho các ứng dụng yêu cầu hoạt động pin dài hoặc tiêu thụ năng lượng thấp.

Đóng gói QFP48: Với kích thước đóng gói QFP48, vi điều khiển này dễ dàng tích hợp vào các thiết kế nhỏ gọn và tiết kiệm không gian.

Timer và PWM: Hỗ trợ lên đến 7 bộ đếm/bộ định thời (Timer), trong đó có các timer hỗ trợ PWM, rất hữu ích trong các ứng dụng điều khiển động cơ và phát tín hiệu.

Giao tiếp nối tiếp: Có nhiều giao diện giao tiếp như UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit) và CAN (Controller Area Network) cho các ứng dụng yêu cầu truyền thông dữ liệu.

ADC 12-bit: Bao gồm 2 bộ chuyển đổi tín hiệu từ tương tự sang số (ADC) 12-bit, với khả năng chuyển đổi nhanh và độ phân giải cao, phù hợp cho các ứng dụng cảm biến.

3.3.3. Thông số kỹ thuật

MCU: STM32F103C8T6

Core: ARM 32 Cortex-M3 CPU

Tần số: 72MHz

Bộ nhớ Flash: 64Kb

SRAM 20Kb

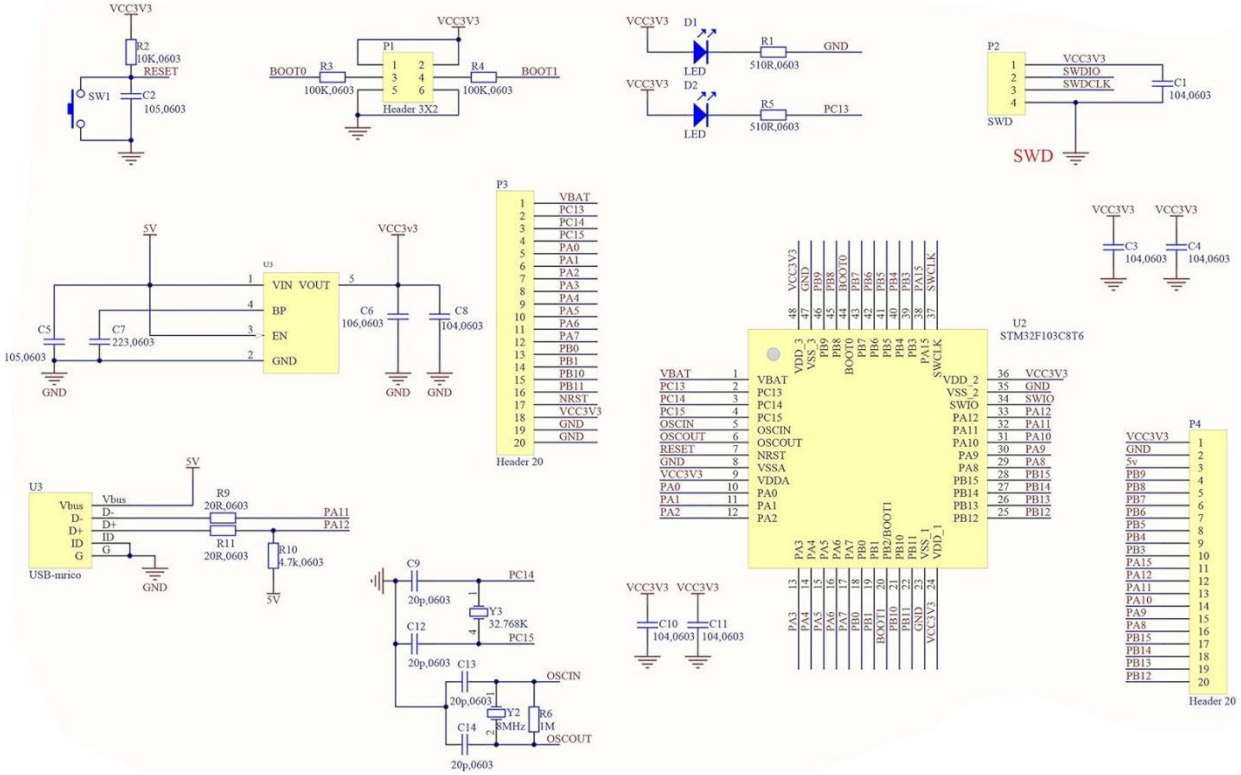
Điện áp I/O: 2.0~3.6 VDC

Thạch anh: 4~16MHz

Cổng MiniUSB dùng để cấp nguồn và giao tiếp.

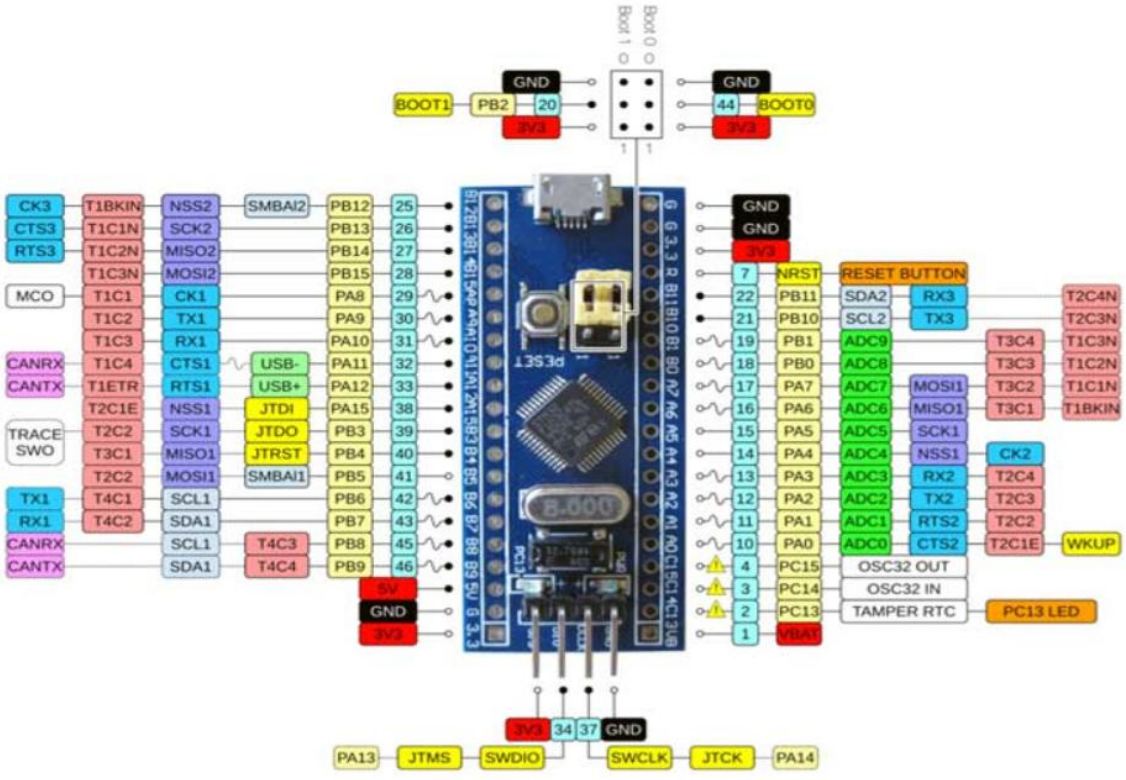
Kích thước: 5.3 x 2.2cm

3.3.4. Sơ đồ nguyên lý của STM32F103C8T6 Blue Pill (schematic)



Hình 3.3.4. Schematic của STM32F103C8T6 Blue Pill

3.3.5. Sơ đồ chân Pin/ Pout



Hình 3.3.5. STM32F103C8T6 Pinout

CHƯƠNG 4

PHẦN CỨNG (SCHEMATIC - PCB)

Phần cứng thiết kế gồm các khối như sau:

+ Khối nguồn: Mạch nguồn 5V và mạch nguồn -5V

+ Khối cảm biến, mạch lọc: cảm biến áp suất MPS20N0400D-D, khuếch đại vi sai AD620AN, mạch lọc thông dải dùng 2 opamp LM741

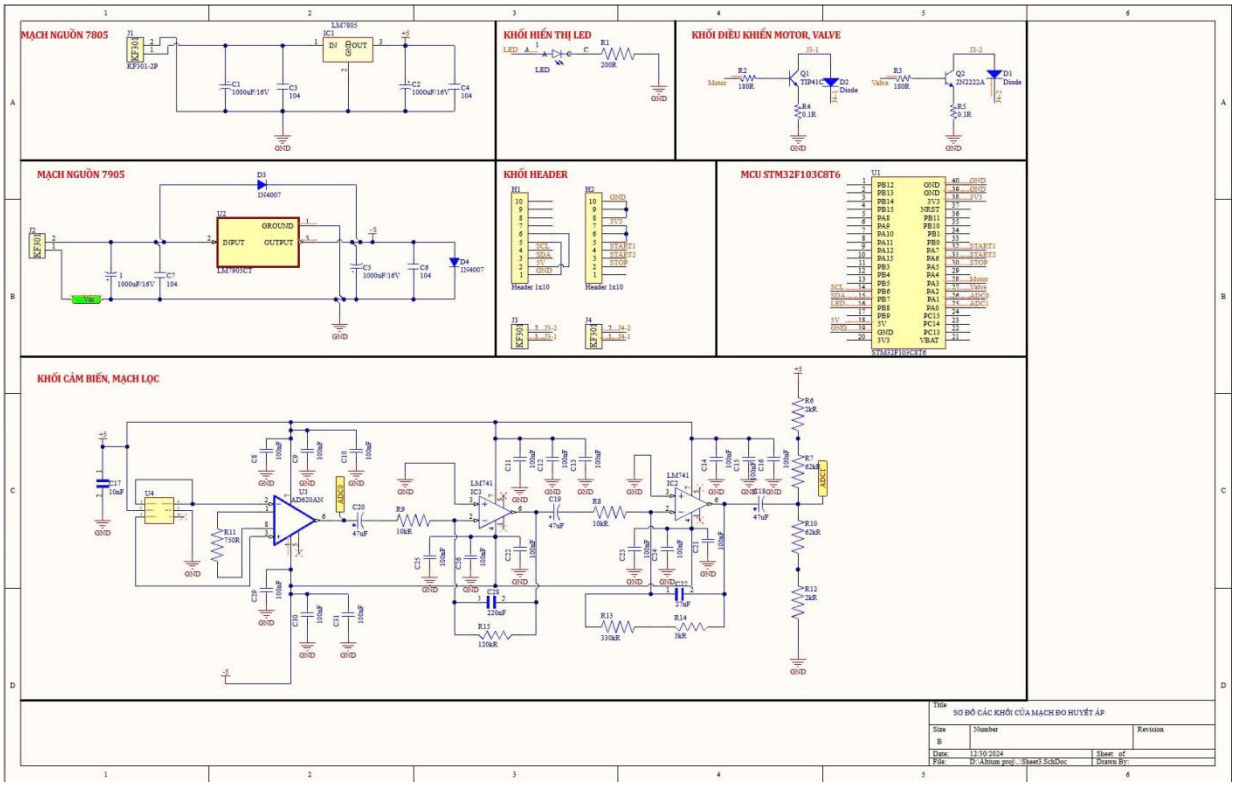
+ Khối điều khiển motor và valve

+ Khối vi điều khiển

+ Khối header

+ Khối hiển thị LED

4.1. Kết quả Schematic (sơ đồ nguyên lý)



Hình 4.1.1. Sơ đồ nguyên lý (schematic)

Mô tả từng khối trong sơ đồ nguyên lý:

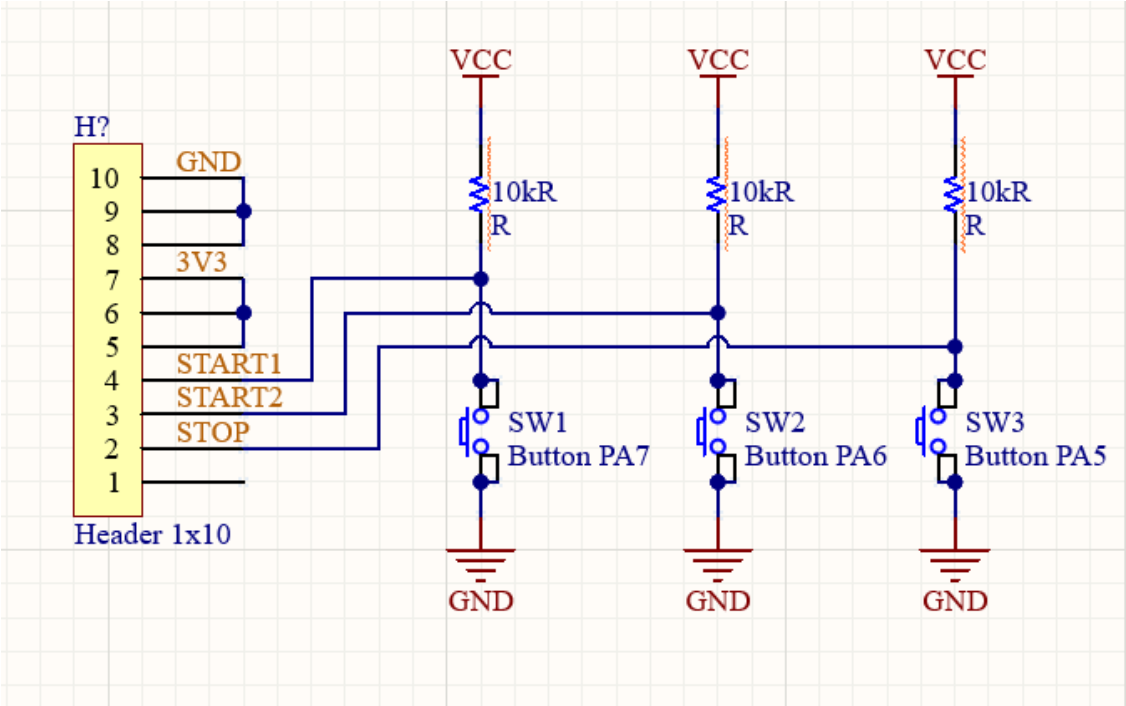
+ Mạch nguồn 7805: đầu vào pin 9V-12V đầu ra 5V, cấp dòng tối đa 1A.

+ Mạch nguồn 7905: điện áp đầu vào có thể từ -35V đến 5V, đầu ra -5V, cấp dòng tối đa 1.5A.

+ LM741 lọc thông dải, độ khuếch đại, mạch chia áp nâng offset lên một khoảng 3.3. $\frac{62+2}{62+2+62+2}=1.65V$ với 4 điện trở: 2 trở 62kR, 2 trở 2kR…

+ Mạch BJT lái dòng motor và valve, đưa mức điện áp thấp ra chân B của BJT, khi đó BJT trong trạng thái không dẫn điện, cực C của BJT được kéo lên nguồn 5V. Còn khi xuất mức 1 ra chân B thì BJT ở trạng thái dẫn điện, chân C được kéo về 0V làm cho motor và valve hoạt động.

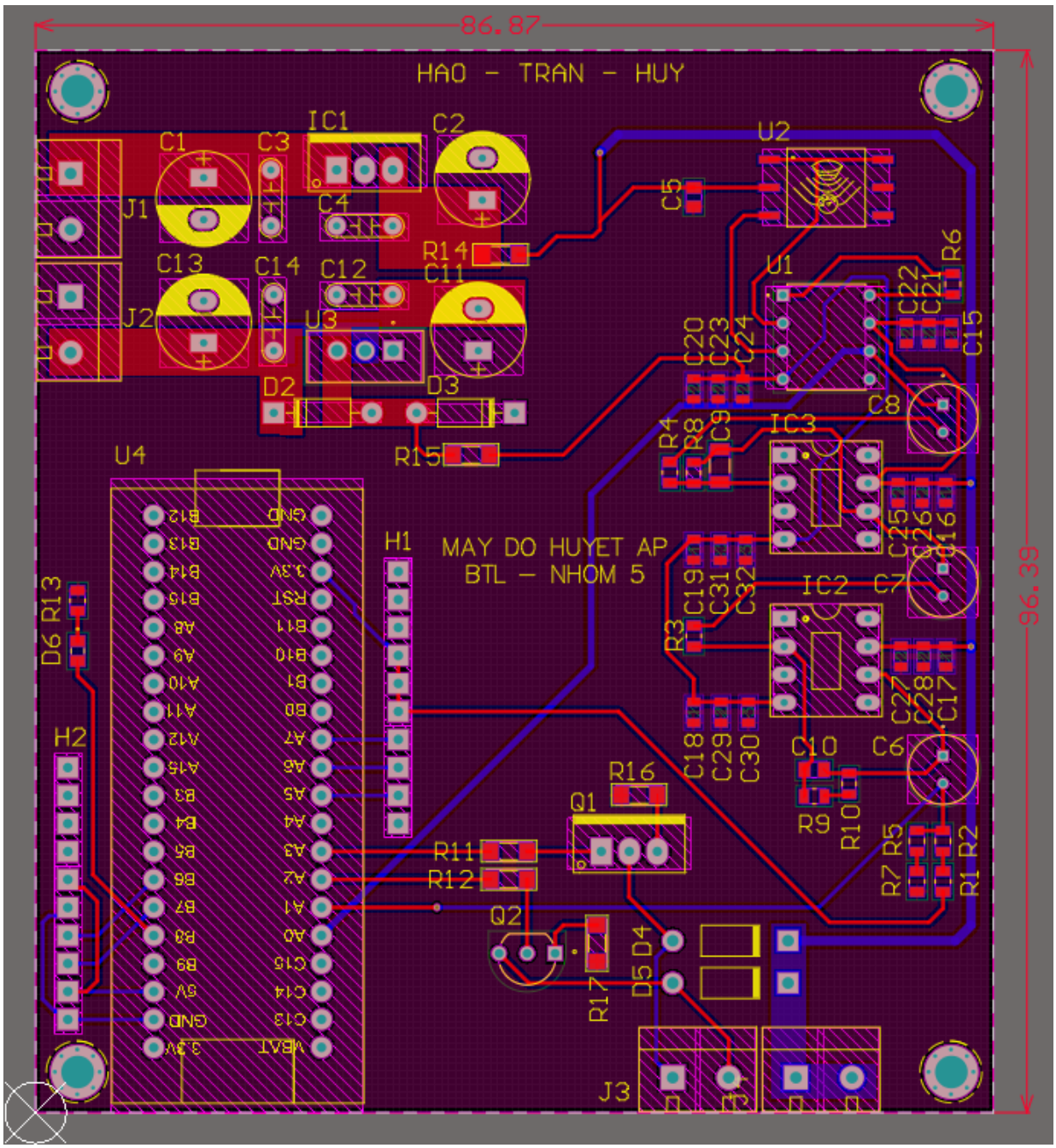
+ Mạch điều khiển nút nhấn , điện trở kéo lên, 3 nút nhấn START1, START2, STOP như (chức năng như phần trình bày phía trên)



Hình 4.1.2. Mạch điều khiển nút nhấn

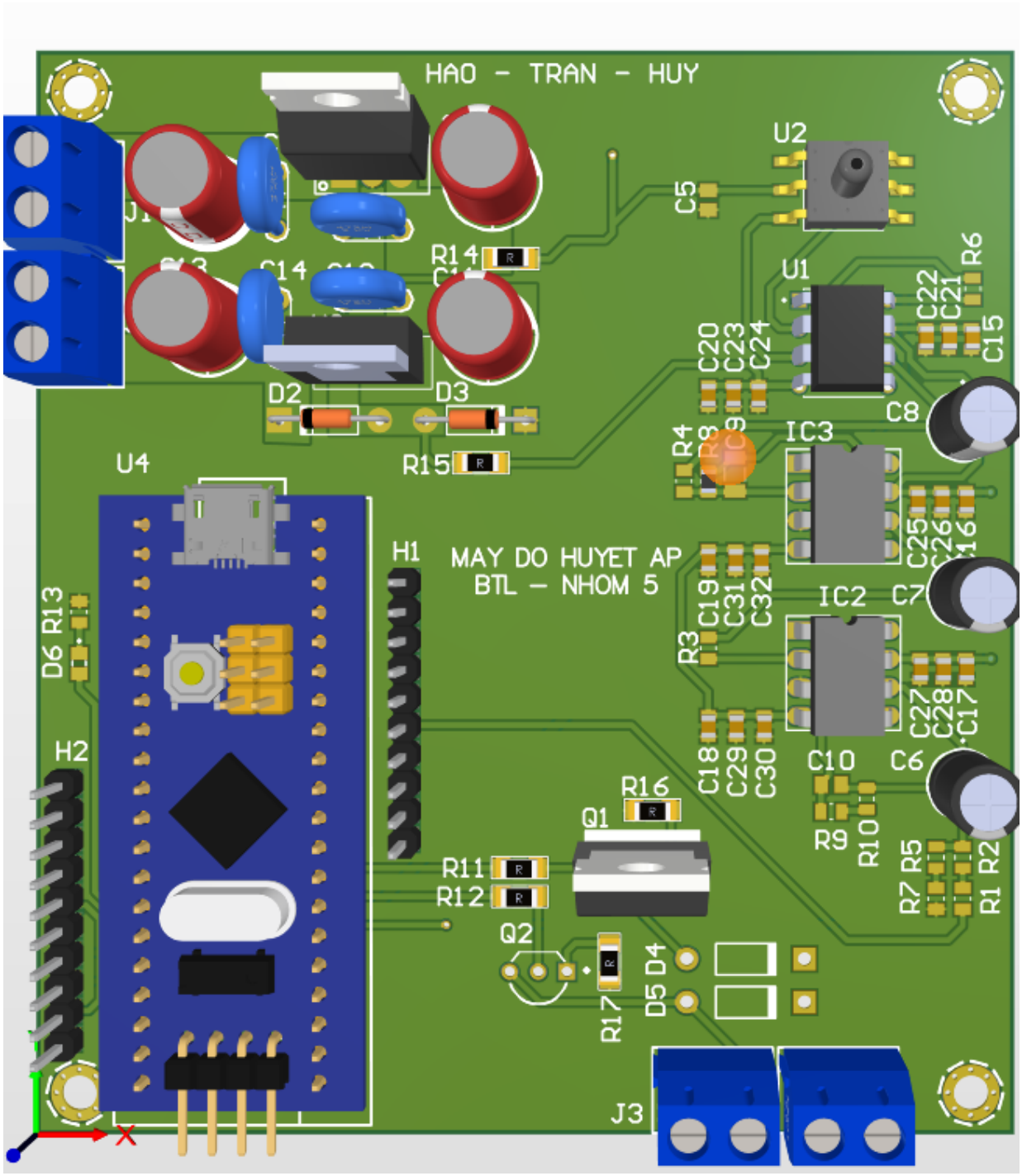
4.2. Kết quả PCB (thực hiện layout 2 lớp)

- Mạch đi dây (layout)



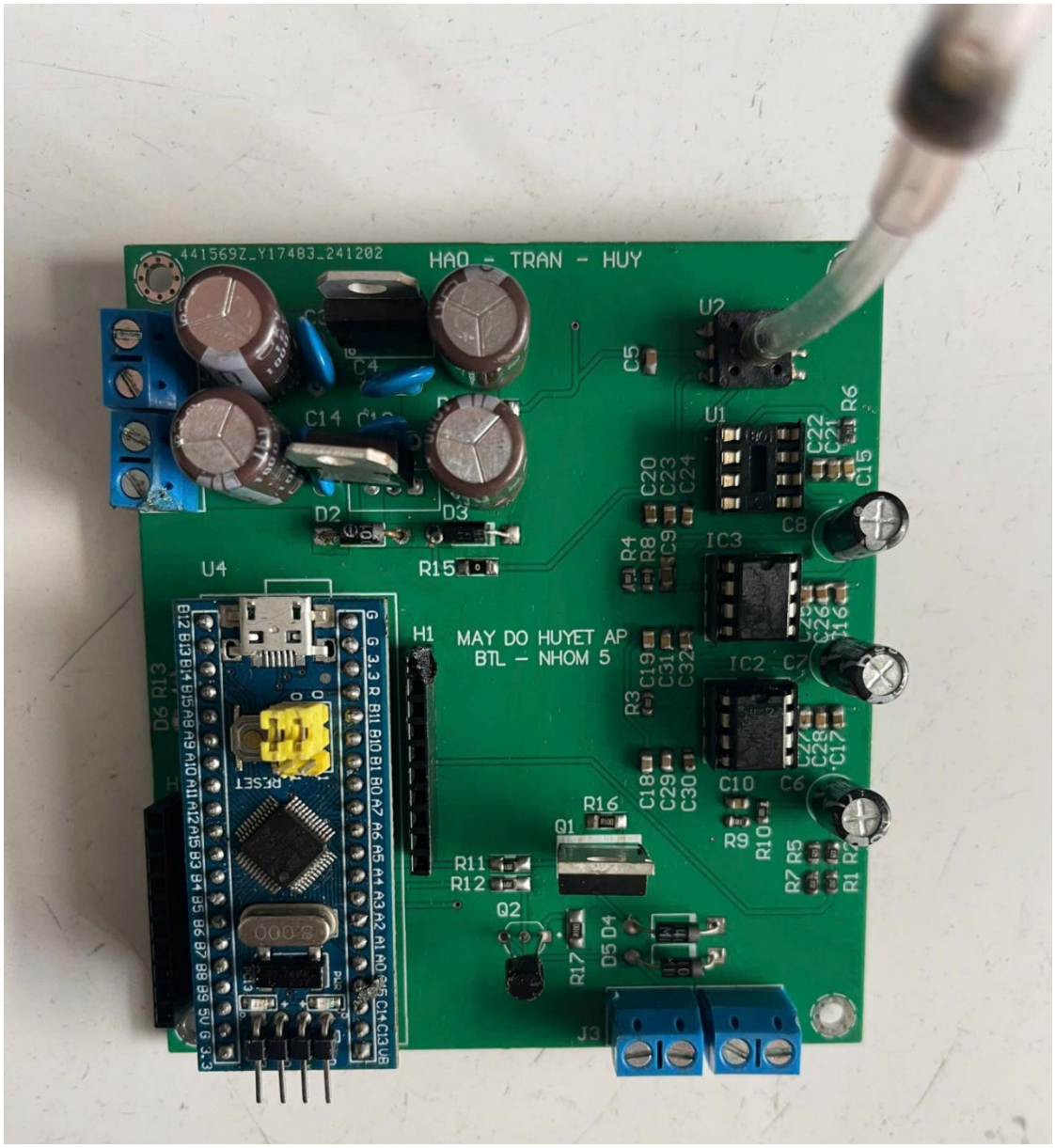
Hình 4.2.1. Mạch đi dây layout

- Chế độ 3D

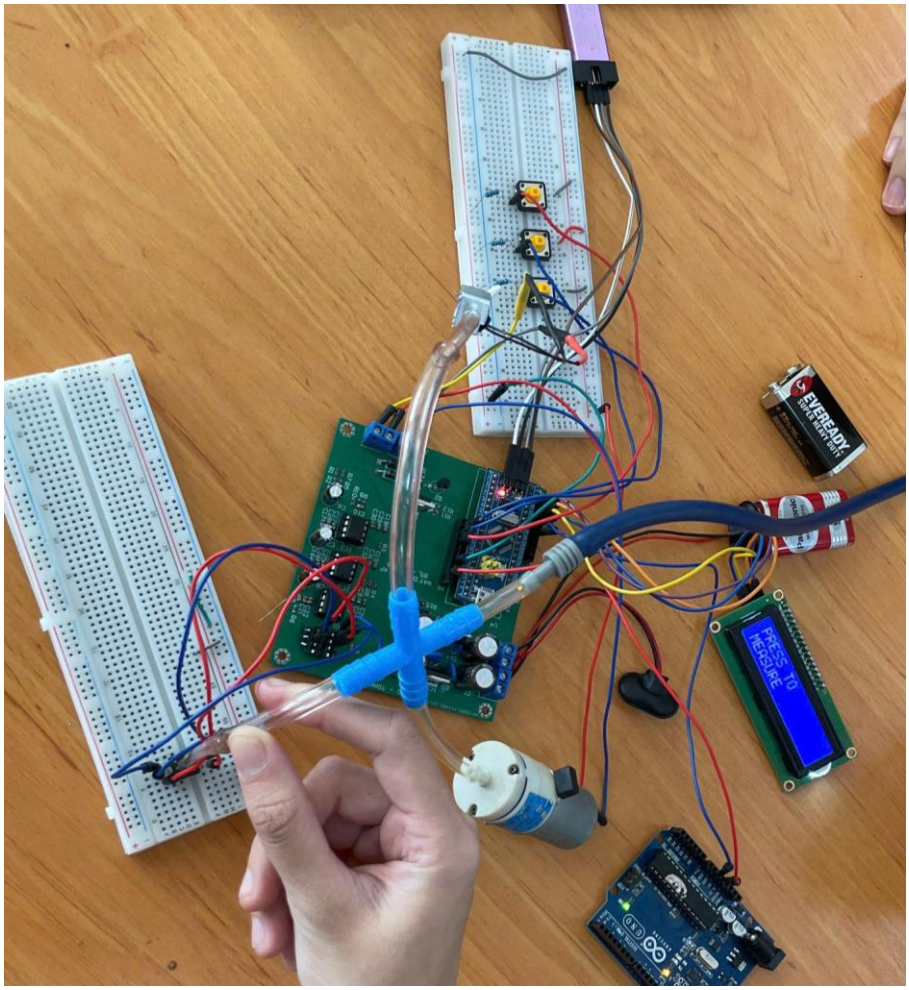


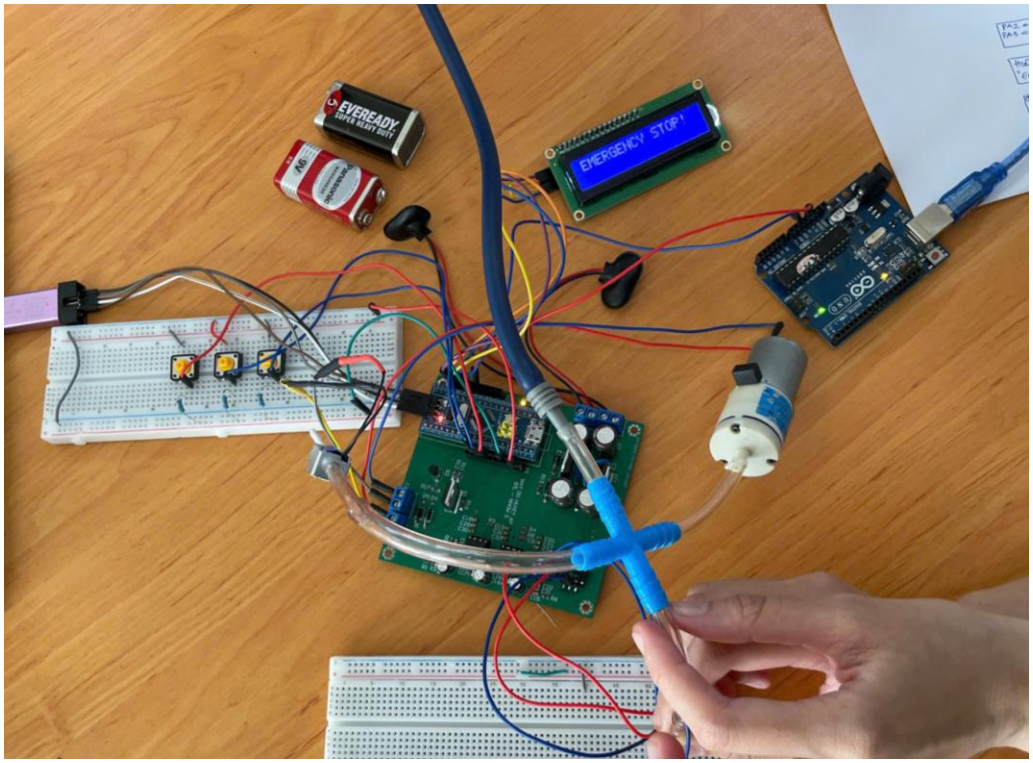
Hình 4.2.3. Mạch layout xem ở chế độ 3D

- Hình ảnh mạch thực tế:


441569Z_Y17483_241202
HAO - TRAN - HUY
MAY DO HUYET AP
BTL – NHOM 5
U4
U2
U1
IC3
IC2
H1
J3
Q1
Q2
R15
R16
D2
D3
C4
C5
C6
C8
C10
C14
C15
RESET

- Hình ảnh mạch chạy thực tế:





- Test mạch bơm hơi:

Link google drive: Test mạch bơm hơi - Google Drive

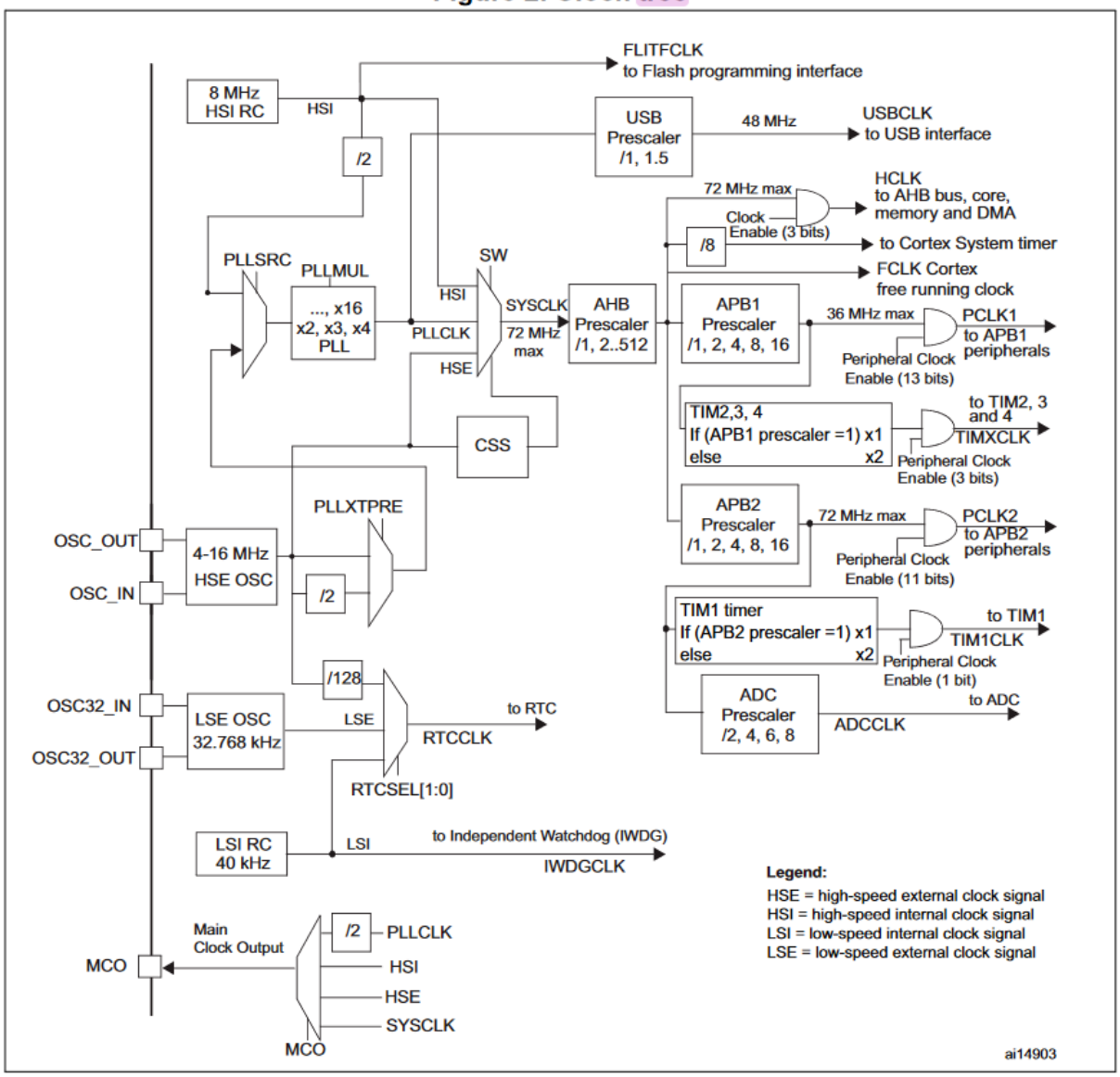
CHƯƠNG 5

PHẦN MỀM

5.1. STM32CubeIDE

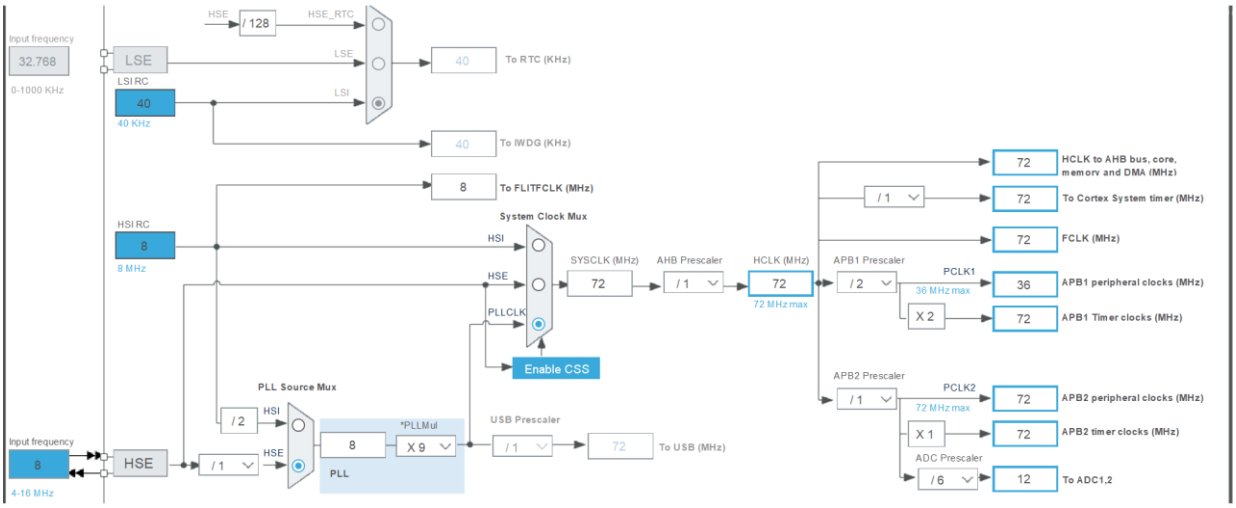
5.1.1. Sử dụng nguồn clock

Figure 2. Clock tree



- Dùng nguồn cấp từ HSE – High Speed Internal (là nguồn clock ngoài 8Mhz từ thạch anh được gắn trên bluepill).

Các thông số được chọn trong project:



5.1.2. ADC

ADC (Analog to Digital Convert) là bộ chuyển đổi tương tự sang số. Điện áp Vin được so sánh với điện áp mẫu Vref (giá trị lớn nhất), sau đó được chuyển đổi thành số .

Bộ ADC trong STM32F103C8T6 có độ phân giải 12 bit. Vref được chọn là nguồn 3.3V.

Sử dụng 2 kênh IN0 và IN1 trong ADC1 của STM32, sử dụng bộ DMA chuyển đổi để CPU làm việc khác.

Cả 2 kênh đều có sampling time là 55.5 cycles. Khi đó ta có thời gian cho một chuyển đổi là:

$$T_{conv} = (55.5+12.5).\frac{1}{12.10^6} = 5.67\mu s .$$

Một số hàm liên quan đến ADC:

– HAL_ADC_Start(ADC_HandleTypeDef* hadc): cho phép ADC bắt đầu chuyển đổi

– HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout): polling chờ cho chuyển đổi hoàn tất với thời gian timeout.

– HAL_ADC_GetValue(ADC_HandleTypeDef* hadc): trả về giá trị adc của con trỏ hadc.

– HAL_ADC_Stop(ADC_HandleTypeDef* hadc): stop chuyển đổi adc.

– HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length): giá trị của ADC chuyển đổi được lưu vào các địa chỉ kế tiếp nhau trong mảng pData.

– HAL_ADC_Stop_DMA(ADC_HandleTypeDef* hadc): dừng chế độ chuyển đổi ADC bằng DMA.

5.1.3. I2C

Kết nối LCD I2C với STM32 Blue Pill và lập trình bằng thư viện STM32CubeIDE và HAL.

5.1.4. Ngắt Timer

Timer trên STM32 là các bộ định thời có thể được cấu hình để đếm lên hoặc đếm xuống, thực hiện các ngắt định kỳ, đo độ dài xung, phát xung PWM, hoặc nhiều chức năng khác.

STM32F103C8T6 có nhiều bộ timer (TIM1, TIM2, TIM3, TIM4, ...), trong đó:

TIM1: Timer nâng cao, hỗ trợ PWM và nhiều tính năng phức tạp.

TIM2-TIM4: Timer cơ bản và tổng quát, thường dùng cho các ứng dụng định thời và đo đạc.

*Cài đặt ngắt timer bằng STM32CubeIDE:

Bước 1: Cấu hình Timer trong CubeMX

1. Tạo project: Mở STM32CubeIDE, tạo một project mới và chọn vi điều khiển STM32F103C8T6.
2. Bật Timer: Trong phần cấu hình Pinout & Configuration, chọn một bộ timer (ví dụ: TIM2) và đặt chế độ Internal Clock.
3. Cấu hình thông số Timer: Trong tab Configuration, chọn TIM2 và đặt thông số:

- Prescaler: Giá trị này chia tần số của clock (APB1) trước khi cấp cho timer.

- Counter Period (ARR - Auto Reload Register): Giá trị tối đa mà bộ đếm có thể đếm được trước khi reset và tạo ngắt.

4. Bật ngắt Timer:

Bật tùy chọn Update Interrupt trong mục NVIC Settings của TIM2.

Bước 2: Nhấn Generate Code để sinh mã tự động.

Bước 3: Viết code xử lý ngắt

Hàm callback mặc định được gọi khi xảy ra ngắt.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {

}
```

5.2. Cấu trúc files trong project, định nghĩa trong các file

5.2.1. Source file: main.c, i2c-lcd.c, func.c

* Hàm main.c:

Khởi tạo các biến toàn cục, các biến cục bộ, gọi các hàm init của các ngoại vi, gọi các hàm khởi tạo ngắt, gọi hàm init_lcd.

Gọi hàm setup: Khởi tạo lại các biến timecount, timing, Vref, DC_gain, pressure, x1, x2, stop_count, hiển thị LCD.

Gọi các hàm trạng thái bắt đầu , bơm hơi, xả hơi, reset bằng switch case nhờ vào biến currentState.

Định nghĩa lại hàm callback cho ngắt timer1 mỗi 1ms:

+ Khi nút start1 nhấn thì x1 = 1.

+ Khi nút start2 nhấn thì x1 = 2.

+ Khi nút stop nhấn thì x2 = 1.

+Thay đổi các biến volatile chỉ thời gian.

* Hàm i2c-lcd.c:

Định nghĩa các hàm cho LCD như init_lcd, lcd_clear, lcd_send_string, lcd_send_data, lcd_send_cmd.

* Hàm func.c:

Định nghĩa các hàm cho các trạng thái: set_up, start_state, inflate1_state, inflate2_state, reset_state.

5.2.2. Header file: main.h, i2c-lcd.h, func.h

* Hàm main.h:

Define lại tên các GPIO port và GPIO pin, các trạng thái.

* Hàm i2c-lcd.h:

Định nghĩa prototype cho các hàm được sử dụng ở i2c-lcd.c

* Hàm func.h:

Định nghĩa prototype cho các hàm được sử dụng ở func.c

5.2.3. Các trạng thái chính đo huyết áp

1. void start_state(*uint8_t* *currentState, float *maxpressure)

Là hàm trạng thái bắt đầu, nó khởi tạo biến timing, đọc trạng thái nút nhấn để chuyển sang trạng thái tương ứng.

2. void inflate1_state(*uint8_t* *currentState, float *pressure, float Vref, float DC_gain)

Tính pressure, hiện lên LCD mỗi 200ms, nếu nút nhấn stop được nhấn thì tắt motor, valve, hiển thị lên LCD dòng cảnh báo “EMERGENCY STOP!”, nếu nút stop không nhấn, chuyển qua hàm inflate2_state.

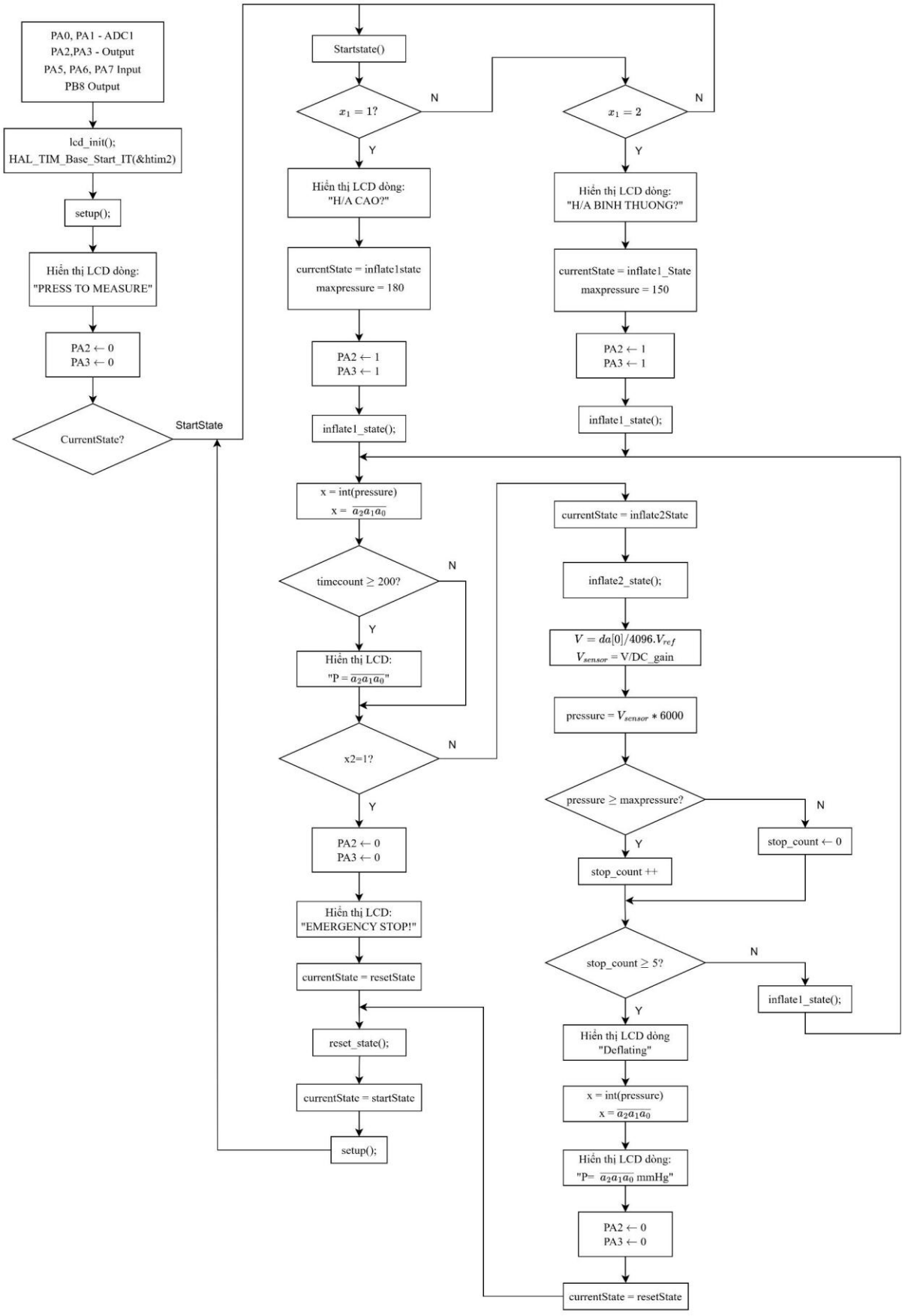
3. void inflate2_state(float Vref, float DC_gain, float *pressure, float maxpressure, *uint8_t* *currentState, *uint8_t* *stop_count)

Đọc ADC bằng DMA, tính pressure, nếu áp suất trong cuff lớn hơn áp suất max thì xả hơi, hiển thị lên LCD dòng “Deflating”, hiển thị áp suất hiện tại, chuyển qua hàm reset_state.

4. void reset_state(*uint8_t* *currentState, float *Vref, float *DC_gain, *uint8_t* *stop_count, float *pressure)

Chuyển về start_state, gọi hàm setup.

5.3. Lưu đồ giải thuật



5.4. Code

- Hàm main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "i2c-lcd.h"
#include "func.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
```

```
DMA_HandleTypeDef hdma_adc1;
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim2;
/* USER CODE BEGIN PV */
volatile uint8_t timing, x1, x2;
volatile uint16_t timecount = 0;
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_I2C1_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint16_t da[1];
/* USER CODE END 0 */
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
              uint8_t currentState = 0;
              float DC_gain;
              float  maxpressure, pressure;
              uint8_t stop_count;
              float Vref;
              int a = 0;
  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
```

```
  HAL_Init();
  /* USER CODE BEGIN Init */
  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_TIM2_Init();
  MX_I2C1_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
       HAL_TIM_Base_Start_IT(&htim2);
       lcd_init();
//     lcd_clear();
//     lcd_put_cur(0,0);
//     lcd_send_string("P = ");
//     HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2, 1);
//     HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3, 1);
//     timing = 40;
//setup(&maxpressure, &meas_state, &former, &TH_sys, &TH_rate,&TH_dias, &time_pulse,
&systolic, &diastolic, &total_pulse_period,
//            &pulse_per_min, &sys_count, &count_average,&countpulse, &Vref, &DC_gain,
&accum_data, &press_data, &count);
       setup(&Vref, &DC_gain, &stop_count, &pressure);
//lcd_send_string("NGUYEN");
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
//       start_state(&currentState, &maxpressure);
//       if(x1 == 1 && a == 0)                          //nút start1 nhấn
//           {
//                   a = 1;
```

```
//            lcd_clear();
//            lcd_put_cur(0,0);
//            lcd_send_string("H/A CAO?");
//            HAL_Delay(1000);
//            maxpressure = 150;
//            currentState = inflate1State;
//            timecount = 0;
//            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
//            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
      // bật motor, valve
//          }
//       HAL_ADC_Start_DMA(&hadc1, (uint32_t *)(da), 1 );
//       HAL_Delay(1);
//            HAL_ADC_Stop_DMA(&hadc1);
//            float V = (float)da[0]/4096 * (Vref);
//            float Vsensor = V/DC_gain;
//            pressure = Vsensor * 6500;
//            int x = (int) pressure;
//            char a2 = x / 100;
//            char a1 = (x / 10) % 10;
//            char a0 = x % 10;
//            if(timecount >= 200) //DONG HO BAM GIAY, SAU 200MS HIEN LCD SYS
//            {
//                  lcd_clear();
//                  lcd_put_cur(1, 0);
//                  lcd_send_string("P = ");
//                  lcd_send_data(a2 + 0x30);
//                  lcd_send_data(a1 + 0x30);
//                  lcd_send_data(a0 + 0x30);
//                  timecount = 0;
//            }
         switch(currentState)
                    {
                      case startState:
                            start_state(&currentState, &maxpressure);
                       break;
```

```
                    case inflate1State:
                        inflate1_state(&currentState, &pressure, Vref, DC_gain);
                    break;
                    case inflate2State:
                        inflate2_state(Vref, DC_gain, &pressure, maxpressure,
&currentState, &stop_count);
                    break;
                case resetState:
                        reset_state(&currentState, &Vref, &DC_gain, &stop_count,
&pressure);
                            break;
                }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}
/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{
  /* USER CODE BEGIN ADC1_Init 0 */
  /* USER CODE END ADC1_Init 0 */
  ADC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN ADC1_Init 1 */
  /* USER CODE END ADC1_Init 1 */
```

```
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */
  /* USER CODE END ADC1_Init 2 */
}
/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{
  /* USER CODE BEGIN I2C1_Init 0 */
  /* USER CODE END I2C1_Init 0 */
  /* USER CODE BEGIN I2C1_Init 1 */
```

```
  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */
  /* USER CODE END I2C1_Init 2 */
}
/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{
  /* USER CODE BEGIN TIM2_Init 0 */
  /* USER CODE END TIM2_Init 0 */
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  /* USER CODE BEGIN TIM2_Init 1 */
  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7200-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

```
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}
```

```
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2|GPIO_PIN_3, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);

  /*Configure GPIO pins : PA2 PA3 */
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PA5 PA6 PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PB8 */
```

```
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
       if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7) == GPIO_PIN_RESET)
              {
                     x1 = 1;
              }
              if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6) == GPIO_PIN_RESET)
              {
                     x1 = 2;
              }
              if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5) == GPIO_PIN_RESET)
              {
                     x2 = 1;
              }
              if(timing > 0) --timing;
              timecount++;
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
```

```
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

- Hàm func.c

```
/*
 * func.c
 *
 *  Created on: Dec 31, 2024
 *      Author: Admin
 */
#include "main.h"
#include "func.h"
#include "i2c-lcd.h"
#include "stm32f1xx_hal.h"
```

```
extern ADC_HandleTypeDef hadc1;
extern volatile uint8_t timing, timecount, x1, x2;
extern uint16_t da[1];
void setup(float *Vref, float *DC_gain, uint8_t *stop_count, float *pressure)
{
            timecount = 0;
            timing=40;
            *Vref = 3.3;
            *DC_gain=66;
            *pressure = 0;
            x1 = 0; //START NHAN, X1=1 ... START2 NHAN, X1=2
            x2 = 0; //STOP NHAN X2=1   ...
            *stop_count = 0;
            lcd_clear();
            lcd_put_cur(0, 0);
            lcd_send_string("PRESS TO");
            lcd_put_cur(1, 0);
            lcd_send_string("MEASURE");
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
}
void start_state(uint8_t *currentState, float *maxpressure)
{
    timing=40;          //thời gian lấy mẫu 40ms
    if(x1 == 1)                          //nút start1 nhấn
    {
       lcd_clear();
       lcd_put_cur(0,0);
       lcd_send_string("H/A CAO?");
       HAL_Delay(1000);
       *maxpressure = 150;
       *currentState = inflate1State;
       timecount = 0;
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);                    //
bật motor, valve
```

```
    }
    if(x1 == 2)                          //nút start2 nhấn
       {
       lcd_clear();
       lcd_put_cur(0,0);
       lcd_send_string("H/A BINH THUONG?");
       HAL_Delay(1000);
              *maxpressure = 180;
              *currentState = inflate1State;
              timecount = 0;
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
       // bật motor, valve
       }
}
void inflate1_state(uint8_t *currentState, float *pressure, float Vref, float DC_gain)
{
       HAL_ADC_Start_DMA(&hadc1, (uint32_t *)(da), 1 );
       HAL_Delay(1);
       HAL_ADC_Stop_DMA(&hadc1);
       float V = (float)da[0]/4096 * (Vref);
       float Vsensor = V/DC_gain;
       *pressure = Vsensor * 6000;
       int x = (int) *pressure;
       char a2 = x / 100;
       char a1 = (x / 10) % 10;
       char a0 = x % 10;
       if(timecount >= 200) //DONG HO BAM GIAY, SAU 200MS HIEN LCD SYS
       {
              lcd_clear();
              lcd_put_cur(1, 0);
              lcd_send_string("P = ");
              lcd_send_data(a2 + 0x30);
              lcd_send_data(a1 + 0x30);
              lcd_send_data(a0 + 0x30);
              lcd_send_string("mmHg");
```

```
            timecount = 0;
        }
        if(x2 == 1)   //nút stop nhấn
        {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);  //tắt motor,
valve
            lcd_clear();
            lcd_put_cur(0,0);
            lcd_send_string("EMERGENCY STOP!");
            *currentState = resetState;
        }
        else
        {
            *currentState = inflate2State;
        }
}
void inflate2_state(float Vref, float DC_gain, float *pressure, float maxpressure,
uint8_t *currentState, uint8_t *stop_count)
{
    if(*pressure>=maxpressure) *stop_count++;
    else *stop_count = 0;
  if(*stop_count>=5){              //nếu đã vượt qua mức max và ổn định thì xả hơi
    lcd_clear();
    lcd_put_cur(0,0);
    lcd_send_string("Deflating"); //TRANG THAI XA HOI
    int x = (int) (*pressure);
       char a2 = x / 100;
       char a1 = (x / 10) % 10;
       char a0 = x % 10;
       lcd_clear();
       lcd_put_cur(1, 0);
            lcd_send_string("P = ");       //HIEN SYS
            lcd_send_data(a2 + 0x30);
            lcd_send_data(a1 + 0x30);
            lcd_send_data(a0 + 0x30);
            lcd_send_string("mmHg");
```

```
            //turn off motor but keep the valve
               HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2, 0);
               HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3, 0);
            HAL_Delay(1000);
            *currentState = resetState;
          }
          else
          {
            *currentState = inflate1State;       //nếu chưa vượt quá mức max thì quay
lại bơm hơi
          }
        }
void reset_state(uint8_t *currentState, float *Vref, float *DC_gain, uint8_t
*stop_count, float *pressure)
{
        HAL_Delay(1000);
        *currentState = startState;
        setup(Vref, DC_gain, stop_count, pressure);
}
```

- i2c-lcd.c

```
/** Put this in the src folder **/
#include "i2c-lcd.h"
extern I2C_HandleTypeDef hi2c1;  // change your handler here accordingly
#define SLAVE_ADDRESS_LCD 0x4E // change this according to ur setup
void lcd_send_cmd (char cmd)
{
  char data_u, data_l;
        uint8_t data_t[4];
        data_u = (cmd&0xf0);
        data_l = ((cmd<<4)&0xf0);
        data_t[0] = data_u|0x0C;  //en=1, rs=0 -> bxxxx1100
        data_t[1] = data_u|0x08;  //en=0, rs=0 -> bxxxx1000
        data_t[2] = data_l|0x0C;  //en=1, rs=0 -> bxxxx1100
        data_t[3] = data_l|0x08;  //en=0, rs=0 -> bxxxx1000
        HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100);
}
```

```
void lcd_send_data (char data)
{
	char data_u, data_l;
	uint8_t data_t[4];

	data_u = (data&0xf0);
	data_l = ((data<<4)&0xf0);
	data_t[0] = data_u|0x0D;  //en=1, rs=0 -> bxxxx1101
	data_t[1] = data_u|0x09;  //en=0, rs=0 -> bxxxx1001
	data_t[2] = data_l|0x0D;  //en=1, rs=0 -> bxxxx1101
	data_t[3] = data_l|0x09;  //en=0, rs=0 -> bxxxx1001
	HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100);
}
void lcd_clear (void)
{
	lcd_send_cmd (0x80);
	for (int i=0; i<70; i++)
	{
		lcd_send_data (' ');
	}
}
void lcd_put_cur(int row, int col)
{
    switch (row)
    {
        case 0:
            col |= 0x80;
            break;
        case 1:
            col |= 0xC0;
            break;
    }
    lcd_send_cmd (col);
}
void lcd_init (void)
{
```

```
	// 4 bit initialisation
	HAL_Delay(50);  // wait for >40ms
	lcd_send_cmd (0x30);
	HAL_Delay(5);  // wait for >4.1ms
	lcd_send_cmd (0x30);
	HAL_Delay(1);  // wait for >100us
	lcd_send_cmd (0x30);
	HAL_Delay(10);
	lcd_send_cmd (0x20);  // 4bit mode
	HAL_Delay(10);
  // dislay initialisation
	lcd_send_cmd (0x28); // Function set --> DL=0 (4 bit mode), N = 1 (2 line display) F = 0 (5x8 characters)
	HAL_Delay(1);
	lcd_send_cmd (0x08); //Display on/off control --> D=0,C=0, B=0  ---> display off
	HAL_Delay(1);
	lcd_send_cmd (0x01);  // clear display
	HAL_Delay(1);
	HAL_Delay(1);
	lcd_send_cmd (0x06); //Entry mode set --> I/D = 1 (increment cursor) & S = 0 (no shift)
	HAL_Delay(1);
	lcd_send_cmd (0x0C); //Display on/off control --> D = 1, C and B = 0. (Cursor and blink, last two bits)
}
void lcd_send_string (char *str)
{
	while (*str) lcd_send_data (*str++);
}
```

- main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
```

```
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */
/* USER CODE END ET */
/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
/* USER CODE END EC */
/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */
/* USER CODE END EM */
/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);
/* USER CODE BEGIN EFP */
```

```
/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */

#define valve_Pin GPIO_PIN_2

#define valve_GPIO_Port GPIOA


#define motor_Pin GPIO_PIN_3

#define motor_GPIO_Port GPIOA


// Define GPIO pins for buttons

#define bt_start1_Pin GPIO_PIN_5

#define bt_start1_GPIO_Port GPIOA


#define bt_start2_Pin GPIO_PIN_6

#define bt_start2_GPIO_Port GPIOA


#define bt_stop_Pin GPIO_PIN_7

#define bt_stop_GPIO_Port GPIOA


// Define ADC pins

#define ADC0_Pin GPIO_PIN_0

#define ADC0_GPIO_Port GPIOA


#define ADC1_Pin GPIO_PIN_1

#define ADC1_GPIO_Port GPIOA


// Define motor states

#define on GPIO_PIN_SET

#define off GPIO_PIN_RESET


// Define states for motor control

#define startState 0

#define inflate1State 1  // Bơm hơi

#define inflate2State 2

#define deflateState 3   // Xả hơi

#define displayState 4
```

```
#define resetState 5


// Define states for Measure control

#define Sys_Measure 6    // ?o tâm thu

#define Sys_Cal 7        // Tính toán tâm thu

#define Rate_Measure 8   // ?o tỷ lệ


#define dias_Measure 9   // ?o tâm trương

#define dias_Cal 10      // Tính toán tâm trương

/* USER CODE END Private defines */

#ifdef __cplusplus

}

#endif

#endif /* __MAIN_H */
```

- func.h

```
/*

 * func.h

 *

 *  Created on: Dec 31, 2024

 *      Author: Admin

 */

#ifndef INC_FUNC_H_

#define INC_FUNC_H_

#include "main.h"

#include "i2c-lcd.h"

void setup(float *Vref, float *DC_gain, uint8_t *stop_count, float *pressure);

void start_state(uint8_t *currentState, float *maxpressure);

void inflate1_state(uint8_t *currentState, float *pressure, float Vref, float
DC_gain);

void inflate2_state(float Vref, float DC_gain, float *pressure, float maxpressure,
uint8_t *currentState, uint8_t *stop_count);

void reset_state(uint8_t *currentState, float *Vref, float *DC_gain, uint8_t
*stop_count, float *pressure);

#endif /* INC_FUNC_H_ */
```

- i2c-lcd.h

```
#include "main.h"
```

```
void lcd_init (void);   // initialize lcd

void lcd_send_cmd (char cmd);  // send command to the lcd

void lcd_send_data (char data);  // send data to the lcd

void lcd_send_string (char *str);  // send string to the lcd

void lcd_put_cur(int row, int col);  // put cursor at the entered position row (0 or
1), col (0-15);

void lcd_clear (void);
```

TÀI LIỆU THAM KHẢO

1. Xuan Minh (2018). Giới thiệu về STM32F103C8T6. Truy cập từ: https://laptrinharmst.blogspot.com/2018/02/bai-00-gioi-thieu-ve-stm32f103c8t6.html

2. Mecsu (2023). Bộ mạch vi điều khiển STM32F103C8T6 Blue Pill. Truy cập từ: https://mecsu.vn/ho-tro-ky-thuat/bo-mach-vi-dieu-khien-stm32f103c8t6-bluepill.goy

3. Lê Phú Ngọc (2018). Máy đo huyết áp với Arduino. Truy cập từ: http://arduino.vn/tutorial/6241-may-do-huyet-ap-voi-arduino