ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG ĐIỆN – ĐIỆN TỬ

BÁO CÁO MÔN HỌC
ĐỒ ÁN THIẾT KẾ 2

ĐỀ TÀI: MẠCH TƯỚI CÂY TỰ ĐỘNG HIỂN THỊ LCD

GVHD	:	Ths. Tào Văn Cường	
Mã lớp	:	754459	
Nhóm sinh viên	:	Nguyễn Trần Tiến Duy	20213848
		Lê Hoàng Anh	20213797
		Nguyễn Hoàng Hiệp	20213912

Hà Nội, 06/2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

LỜI NÓI ĐẦU

Thế kỷ 21 mở ra một thời đại mới của khoa học công nghệ, đòi hỏi con người luôn luôn không ngừng tìm tòi học tập để tiến bộ. Thiết bị và công nghệ luôn được đổi mới tiên tiến hiện đại để phục vụ con người, góp phần nâng cao chất lượng cuộc sống. Sự phát triển không ngừng của khoa học kỹ thuật, đặc biệt là trong ngành Điện tử - Viễn thông, đã tạo ra nhiều ứng dụng trong công nghiệp. Trong lĩnh vực điều khiển, công nghệ chế tạo vi mạch lập trình đã mang lại những kỹ thuật điều khiển hiện đại với nhiều ưu điểm như kích thước mạch nhỏ gọn, giá thành thấp, độ tin cậy cao và tiêu thụ ít năng lượng, vượt trội hơn so với các mạch điều khiển sử dụng linh kiện rời. Hiện nay, công nghệ điều khiển được ứng dụng rộng rãi trong các thiết bị phục vụ đời sống hàng ngày như máy giặt, đồng hồ điện tử, ti vi... giúp cuộc sống con người ngày càng hiện đại và tiện nghi hơn.

Việc theo dõi môi trường sống xung quanh như nhiệt độ, độ ẩm điều chỉnh lại cho phù hợp với con người và môi trường sống. Việc theo dõi nhiệt độ độ ẩm cũng được ứng dụng vào các lĩnh vực như chăn nuôi và nông nghiệp. Vì vậy, em đã lựa chọn đề tài "Thiết kế mạch tưới cây tự động". Mục tiêu của đề tài là khai thác chức năng của Arduino Nano, cảm biến DHT11 để theo dõi nhiệt độ, độ ẩm và cảm biến độ ẩm của đất .

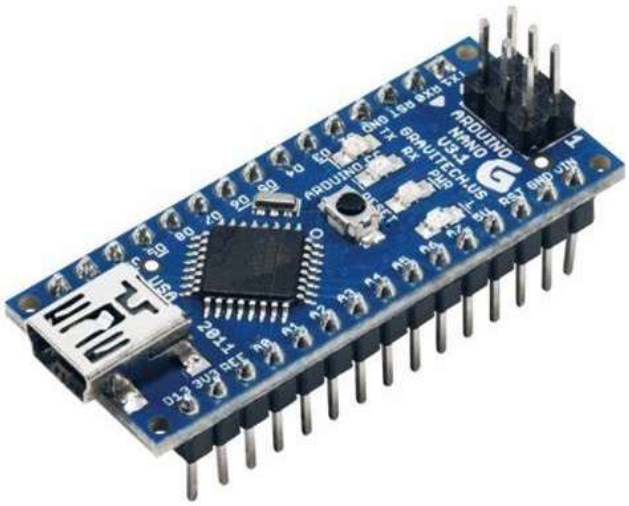
Em xin gửi lời cảm ơn chân thành đến thầy Cường đã trực tiếp hướng dẫn, định hướng, góp ý để em có thể hoàn thành đề tài này. Do trình độ và kinh nghiệm thực tế có hạn, đề tài không tránh khỏi những thiếu sót, em rất mong nhận được những ý kiến đóng góp, giúp đỡ chân tình, quý báu của quý thầy cô và các bạn sinh viên.

I. CƠ SỞ LÝ THUYẾT

1.1 Tìm hiểu Arduino Nano

1.1.1 Tổng quan về Arduino Nano

Arduino Nano là một bo mạch phát triển nhỏ gọn dựa trên vi điều khiển ATmega328P, sử dụng kiến trúc RISC (Reduced Instruction Set Computer) 8-bit với khả năng tiêu thụ điện năng thấp. Bo mạch này tích hợp đầy đủ các tính năng chuyển đổi tín hiệu Analog-to-Digital (ADC) và hỗ trợ xuất tín hiệu Digital-to-Analog thông qua PWM.
Với kiến trúc RISC tối ưu, Arduino Nano có thể thực thi hầu hết các lệnh trong một chu kỳ xung nhịp duy nhất, cho phép đạt hiệu suất xử lý lên đến 20 triệu lệnh trên giây khi hoạt động ở tần số tối đa 20MHz. Vi điều khiển ATmega328P trên Arduino Nano cung cấp nhiều chế độ tiết kiệm năng lượng linh hoạt, cho phép các nhà phát triển tối ưu hóa mức tiêu thụ điện năng phù hợp với từng ứng dụng cụ thể mà vẫn duy trì hiệu suất xử lý cao. Kích thước nhỏ gọn (18mm x 45mm) và khả năng tương thích hoàn toàn với môi trường phát triển Arduino IDE, Arduino Nano trở thành lựa chọn lý tưởng cho các dự án IoT, thiết bị nhúng và các ứng dụng yêu cầu không gian lắp đặt hạn chế.



Hình 1. Hình ảnh thực tế của Arduino Nano

Arduino Nano sở hữu tập lệnh phong phú với 32 thanh ghi làm việc đa năng trên vi điều khiển ATmega328P. Toàn bộ 32 thanh ghi này được kết nối trực tiếp với ALU (Arithmetic Logic Unit), cho phép truy cập đồng thời hai thanh ghi độc lập trong một chu kỳ xung nhịp duy nhất. Kiến trúc RISC này mang lại hiệu suất xử lý vượt trội, nhanh gấp nhiều lần so với các vi điều khiển CISC truyền thống.Với 32KB bộ nhớ Flash với khả năng đọc trong khi ghi (bootloader chiếm 0.5KB), 1KB bộ nhớ EEPROM, và 2KB bộ nhớ SRAM - gấp đôi so với ATmega16; 14 chân digital I/O (6 chân hỗ trợ PWM), 8 chân analog input, giao tiếp

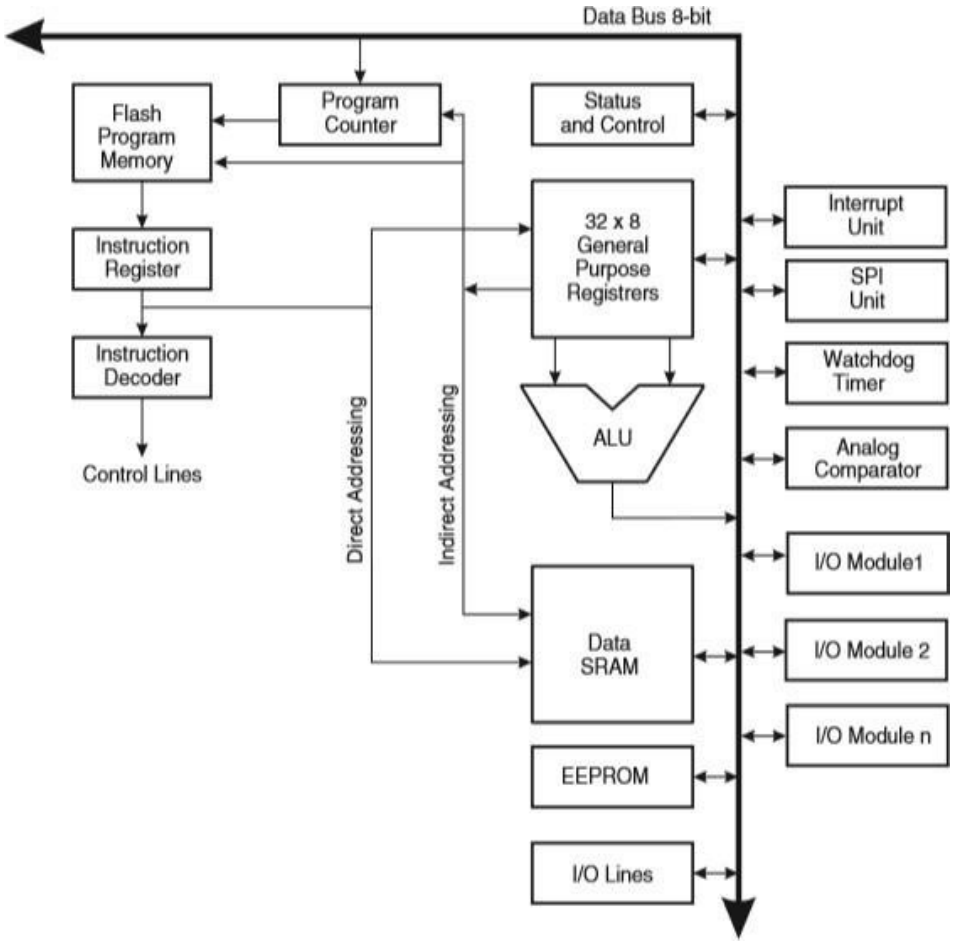
UART/USART tích hợp, hỗ trợ giao thức I2C và SPI, cùng với các ngắt nội và ngắt ngoại linh hoạt. Bộ ADC 10-bit với 6 kênh (8 kênh trên phiên bản SMD), 3 bộ Timer/Counter đa chức năng hỗ trợ PWM và các chế độ đếm khác nhau, hoạt động ổn định ở tần số 16MHz với nguồn cung cấp 5V, hoặc 8MHz với nguồn 3.3V.

Arduino Nano được hỗ trợ toàn diện bởi hệ sinh thái Arduino IDE, cùng với hàng nghìn thư viện mã nguồn mở, công cụ debug, và cộng đồng phát triển lớn trên toàn thế giới. Khả năng tương thích với các công cụ phát triển chuyên nghiệp như Atmel Studio và PlatformIO giúp Arduino Nano trở thành lựa chọn lý tưởng cho cả người mới học và các nhà phát triển chuyên nghiệp.

Cấu trúc nhân

CPU trong Arduino Nano (vi điều khiển ATmega328P) đóng vai trò trung tâm, đảm bảo sự hoạt động chính xác của chương trình. Nó có khả năng truy cập bộ nhớ, thực hiện các phép tính toán logic và số học, điều khiển các thiết bị ngoại vi như Timer, ADC, UART, và xử lý các ngắt (interrupts) để đáp ứng nhanh với các sự kiện bên ngoài.

Cấu trúc tổng quát: AVR sử dụng cấu trúc Harvard, tách biệt bộ nhớ và các bus cho chương trình và dữ liệu. Các lệnh được thực hiện chỉ trong một chu kì xung clock. Bộ nhớ chương trình được lưu trong bộ nhớ Flash.



Hình 1 Sơ đồ khối của kiến trúc AVR MCU

- **ALU**: ALU làm việc trực tiếp với các thanh ghi chức năng chung. Các phép toán được thực hiện trong một chu kì xung clock. Hoạt động của ALU được chia thành 3 loại: đại số, logic, theo bit.
- **Thanh ghi trạng thái**: Thanh ghi trạng thái là thanh ghi có 8 bit lưu giữ trạng thái của ALU sau các phép tính số học và logic.

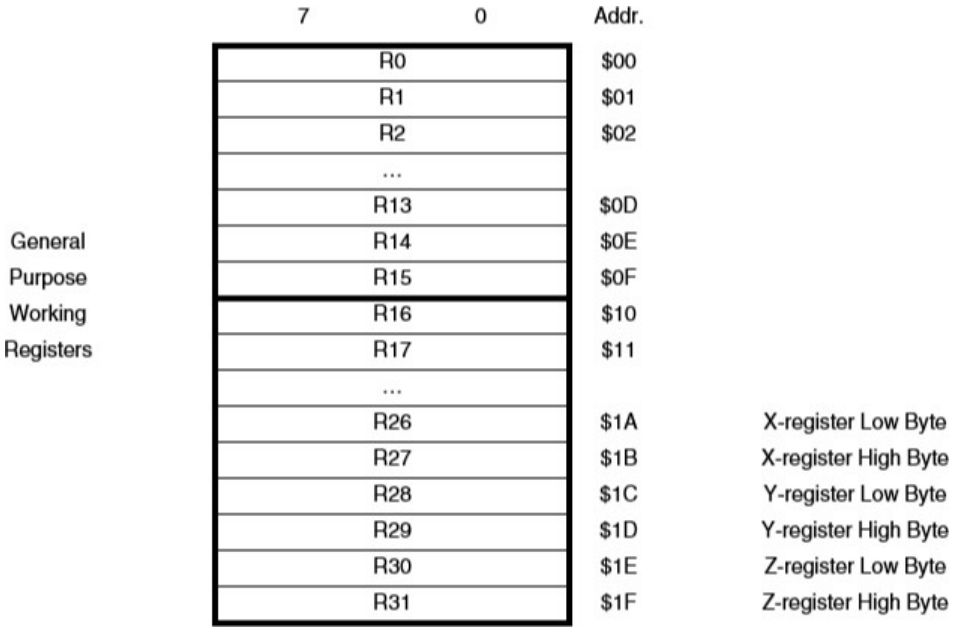
Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 2 Thanh ghi trạng thái

- **Các thanh ghi chức năng chung**

Trong đó:

- C: Carry Flag – cờ nhớ (nếu phép toán có nhớ, cờ sẽ được thiết lập)
- Z: Zero Flag – cờ Zero (nếu kết quả phép toán bằng 0)
- N: Negative Flag (nếu kết quả phép toán là âm)
- V: Two's complement overflow indicator (Cờ được thiết lập khi tràn số bù 2)
- H: Half Carry Flag
- T: Transfer bit được sử dụng làm nơi trung gian trong các lệnh BLD, BST)
- I: Global Interupt Enable/Disable Flag (Bit cho phép ngắt toàn cục. Nếu bit này ở trạng thái logic 0 thì không có một ngắt nào được phục vụ)



Hình 3 . Các thanh ghi chức năng chung

- **Con trỏ ngăn xếp (SP):** Là một thanh ghi 16-bit nhưng cũng có thể được xem như hai thanh ghi chức năng đặc biệt 8 bit. Có địa chỉ trong các thanh ghi chức năng đặc biệt là $3E (trong bộ nhớ RAM là $5E) và có nhiệm vụ trỏ tới vùng nhớ trong RAM chứa ngăn xếp.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Hình 4 Thanh ghi con trỏ ngăn xếp

- **Quản lý ngắt**

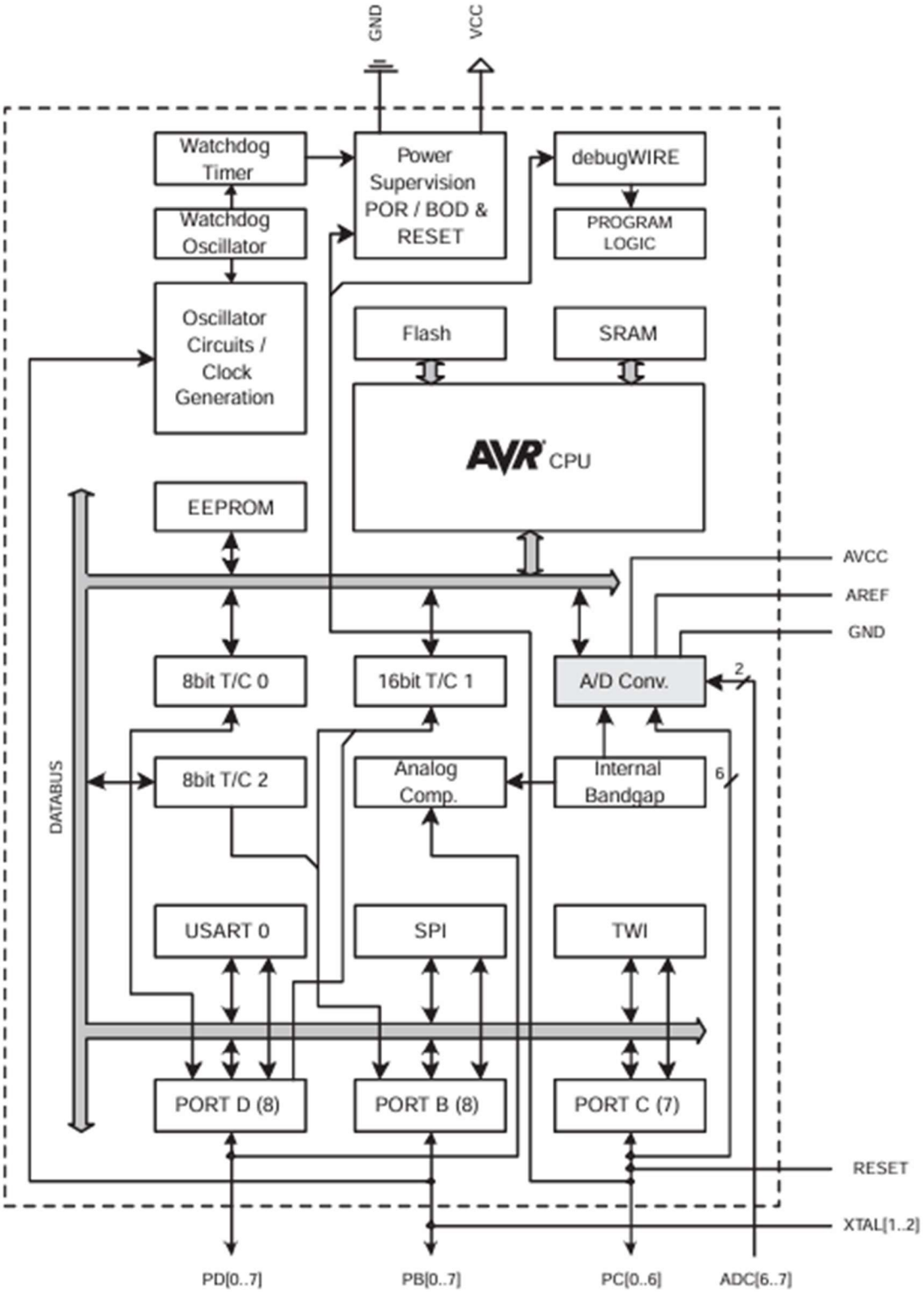
Ngắt là một cơ chế cho phép thiết bị ngoại vi báo cho CPU biết về tình trạng sẵn sàng trao đổi dữ liệu của mình. Ví dụ: Khi bộ truyền nhận UART nhận được một byte, nó sẽ báo cho CPU biết thông qua cờ RXC, hoặc khi nó đã truyền được một byte thì cờ TX được thiết lập… Khi có tín hiệu báo ngắt CPU sẽ tạm dừng công việc đang thực hiện lại và lưu vị trí đang thực hiện chương trình (con trỏ PC) vào ngăn xếp sau đó trỏ tới vector phục vụ ngắt và thực hiện chương trình phục vụ ngắt đó cho tới khi gặp lệnh RETI (Return from Interupt) thì CPU lại lấy PC từ ngăn xếp ra và tiếp tục thực hiện chương trình mà trước khi có ngắt nó đang thực hiện.

Trong trường hợp mà có nhiều ngắt yêu cầu cùng một lúc thì CPU sẽ lưu các cờ báo ngắt đó lại, và thực hiện lần lượt các ngắt theo mức ưu tiên. Trong khi đang thực hiện ngắt mà xuất hiện ngắt mới thì sẽ xảy ra hai trường hợp. Trường hợp ngắt này có ưu tiên cao hơn thì nó sẽ được phục vụ, và ngược lại, nếu có mức ưu tiên thấp hơn thì nó sẽ bị bỏ qua.

Bộ nhớ ngăn xếp là vùng bất kì trong SRAM từ địa chỉ 0x60 trở lên. Để truy nhập vào SRAM thông thường thì ta dùng con trỏ X, Y, Z và truy cập vào SRAM theo kiểu ngăn xếp thì ta dùng con trỏ SP. Con trỏ nagy là một thanh ghi 16 bit và được truy cập như hai thanh ghi 8 bit chung có địa chỉ: SPL: 0x3D/0x5D (IO/SRAM) và SPH:0x3E/0x5D.

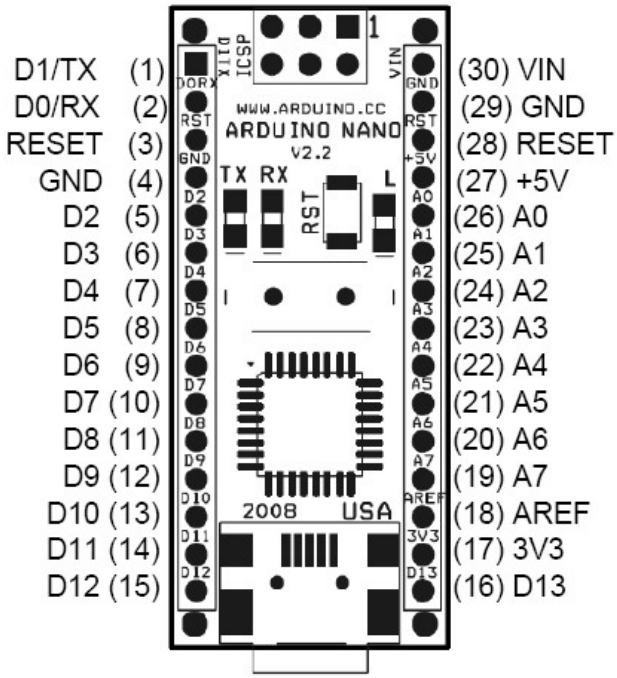
Khi chương trình phục vụ ngắt hoặc chương trình con thì con trỏ PC được lưu vào ngăn xếp trong khi con trỏ ngăn xếp bị giảm hai vị trí. Và con trỏ ngăn xếp sẽ giảm 1 khi thực hiện lệnh PUSH. Ngược lại thực hiện lệnh POP thì con trỏ ngăn xếp sẽ tăng 1 và khi thực hiện lệnh RET hoặc RETI thì con trỏ ngăn xếp sẽ tăng 2. Như vậy, con trỏ ngăn xếp cần được chương trình đặt trước giá trị khởi tạo ngăn xếp trước khi một chương trình con được gọi hoặc các ngắt được cho phép phục vụ. Và giá trị ngăn xếp ít nhất cũng phải lớn hơn 60H (0x60) vì 5FH trỏ lại là vùng các thanh ghi.

1.1.2 Sơ đồ khối



Hình 5 Sơ đồ khối Arduino Nano

1.1.3 Sơ đồ chân và ý nghĩa các chân



Hình 6 Sơ đồ chân của Arduino Nano

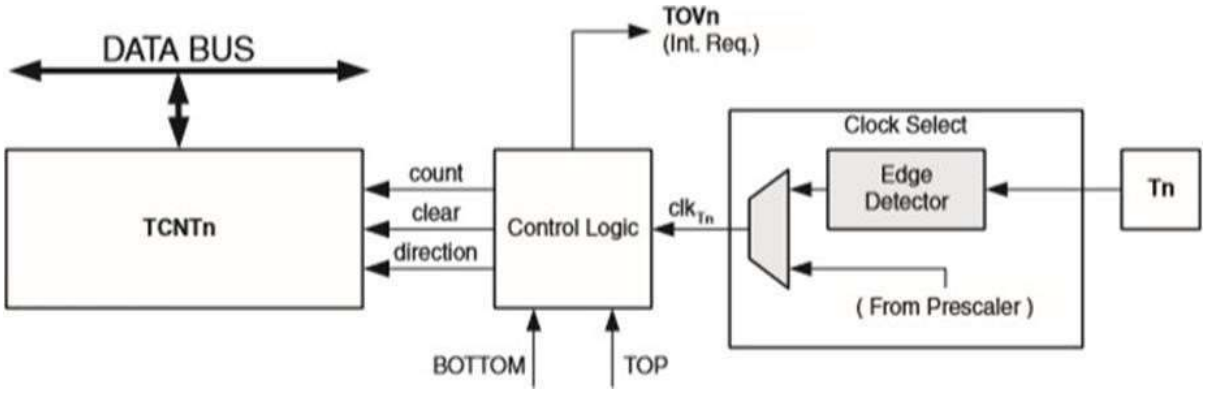
- **Chân VCC**: Là chân cấp điện áp nguồn cho vi điều khiển. Điện áp nguồn tiêu chuẩn là +5V (±0.5V). Có thể cấp trực tiếp từ nguồn ngoài hoặc qua cổng USB.
- **Chân GND**: Là chân nối đất (Mass). Arduino Nano có 2 chân GND. Khi thiết kế cần sử dụng mạch ổn áp (ví dụ IC 7805) nếu cấp nguồn ngoài cho chân RAW.
- **Port A (PA)**: Trên Arduino Nano, Port A tương ứng với các chân **A0 đến A7** (từ PC0 đến PC5, thêm ADC6/ADC7 nội bộ). Có chức năng đầu vào analog để chuyển đổi ADC.
- **Port B (PB)**: Gồm các chân từ **D8 đến D13** (PB0–PB5). Ngoài chức năng xuất/nhập, còn có thể dùng cho SPI (D11, D12, D13) và xuất PWM.
- **Port C (PC)**: Bao gồm các chân **A0 đến A5**, dùng cho chức năng analog hoặc digital. Một số chân còn có thể dùng làm chân I2C (A4 - SDA, A5 - SCL).
- **Port D (PD)**: Bao gồm các chân từ **D0 đến D7**, dùng cho chức năng xuất/nhập. Các chân D0 và D1 có chức năng giao tiếp UART (Serial).
- **Chân RESET (RST)**: Là chân reset ở gần đầu cắm USB. Khi kéo chân này xuống mức thấp trong ít nhất 2 chu kỳ clock, hệ thống sẽ thiết lập lại toàn bộ về trạng thái ban đầu.
- **Chân XTAL1 và XTAL2**: Không lộ ra ngoài trên board Arduino Nano, nhưng được nối nội bộ với thạch anh 16 MHz và các tụ để tạo xung clock ổn định.
- **Chân AVCC**: Là nguồn cấp riêng cho bộ ADC. Trên Arduino Nano, chân này được nối chung với VCC. Nếu dùng ADC, nên nối qua cuộn lọc hoặc tụ để giảm nhiễu.
- **Chân AREF**: Là chân chuẩn tham chiếu analog cho bộ ADC. Mặc định là 5V nếu không nối gì. Có thể nối điện áp ngoài (ví dụ 3.3V) để làm mức tham chiếu ADC chính xác hơn.

1.1.4 Bộ đinh thời

Bộ định thời (Timer/Counter) là một mô-đun quan trọng trong vi điều khiển Arduino Nano, dùng để tạo độ trễ chính xác, phát PWM, tạo tần số, hoặc đếm các sự kiện ngoài. Trên Arduino Nano (sử dụng vi điều khiển ATmega328P), Timer0 là một bộ đếm 8-bit, có các đặc điểm chính như sau:

Đặc điểm của Timer0:

- Bộ đếm một kênh: chỉ có một bộ đếm chính TCNT0.
- Tự động xóa khi so sánh (CTC mode): bộ đếm sẽ trở về 0 khi TCNT0 = OCR0.
- Phát xung PWM: dùng để điều khiển độ sáng LED hoặc tốc độ động cơ.
- Tạo tần số ổn định: nhờ vào bộ so sánh và chia tần.
- Bộ đếm sự kiện ngoài: có thể đếm số lần xuất hiện của tín hiệu ngoài trên chân T0 (PD4).
- Bộ chia tần số (Prescaler): 10-bit cho phép chia xung clock nội để phù hợp với ứng dụng.
- Ngắt tràn và ngắt so sánh: điều khiển bằng TIFR (cờ ngắt) và TIMSK (cho phép ngắt)

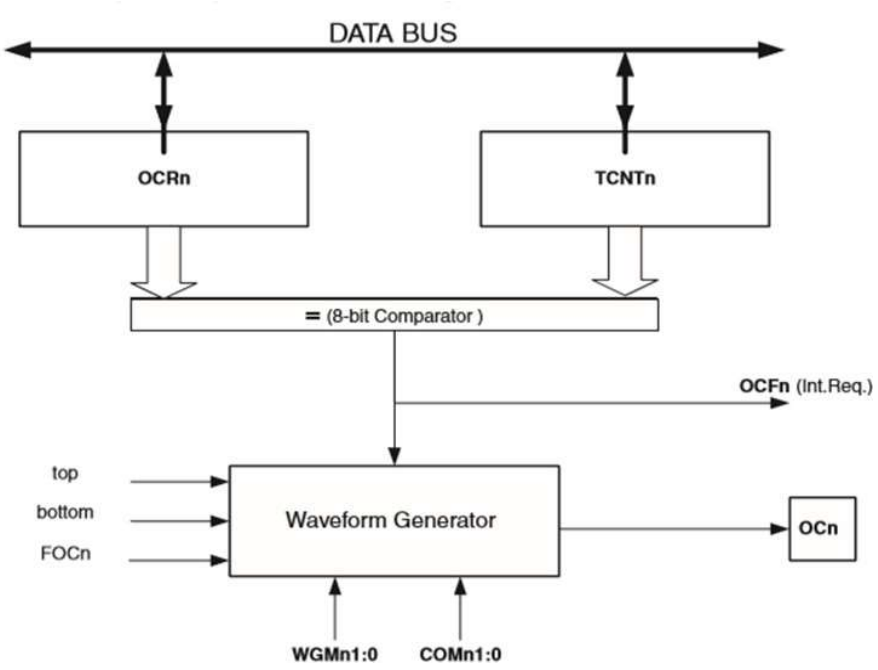


Hình 7 Sơ đồ đơn vị đếm

Trong đó:

- Count: tăng hay giảm TCN0:1
- Direction: lựa chọn giữa đếm lên và đếm xuống
- Clear: xóa thanh ghi TCNT0 (đưa các bit về 0)

Bộ so sánh 8-bit liên tục so sánh giá trị TCNT0 với các giá trị thanh ghi so sánh ngõra (OCR0). Khi giá trị TCNT0 bằng với OCR0, bộ so sánh sẽ tạo ra một báo hiệu. Báohiệu này sẽ đặt giá trị cờ so sánh ngõ ra (OCF0) lên 1 vào chu kì clock tiếp theo. Nếuđược kích hoạt (OCIE0 = 1), cờ OCF0 sẽ tạo ra một ngắt so sánh ngõ ra và sẽ tự động được xóa khi ngắt được thực thi. Cờ OCF0 cũng có thể xóa được bằng phần mềm.



Hình 8 . Sơ đồ đơn vị so sánh ngõ ra

- Thanh ghi điều khiển bộ định thời/ bộ đếm - TCCR0 (Timer/Counter Control Register)

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 9 Thanh ghi điều khiển bộ định thời/bộ đếm – TCCR0

- Bit 7 (FOC0): so sánh ngõ ra bắt buộc: bit này chỉ tích cực khi bit WGM00 chỉ định chế độ làm việc không có PWM. Khi đặt bit này lên 1, một báo hiệu so sánh bắt buộc xuất hiện đơn vị dạng tạo sóng
- Bit 6 (WGM00) – Bit 3 (WGM01): chế độ tạo dạng sóng: các bit này điều khiển hoạt động của chân OC0. Nếu một hoặc cả hai bit COM01, COM00 được đặt lên 1, ngõ ra OC0 sẽ hoạt động

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Bảng 1 Chế độ đầu ra so sánh, non-PWM

- Bit 2 (CS02), bit 1 (CS01), bit 0 (CS00) chọn xung đồng hồ: ba bit này dùng để lựa chọn nguồn xung cho bộ định thời/bộ đếm.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Bảng 2 Bảng lựa chọn nguồn xung cho bộ định thời

- Thanh ghi bộ định thời/bộ đếm TCNT0

  Cho phép truy cập trực tiếp vào bộ đếm 8 bit:

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 10. Thanh ghi TCNT0

- Thanh ghi so sánh ngõ ra – OCR0

  Thanh ghi chứa một giá trị 8bit và liên tục được so sánh với giá trị của bộ đếm.

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 11 . Thanh ghi so sánh ngõ ra – OCR0

- Thanh ghi mặt nạ ngắt

- Bit 1 (OCIE0): cho phép ngắt báo hiệu so sánh
- Bit 0 (TOIE0): cho phép ngắt tràn bộ đếm

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 12 Thanh ghi mặt nạ ngắt

- Thanh ghi cờ ngắt bộ định thời
  - Bit 1 (OCF0): cờ so sánh ngõ ra 0
  - Bit 0 (TOV0): cờ tràn bộ đếm

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hình 13 Thanh ghi cờ ngắt bộ định thời

Bit TOV0 được đặt lên 1 khi bộ đếm bị tràn và được xóa bởi phần cứng khi vector ngắt tương ứng được thực hiện. Bit này cũng có thể được xóa bằng phần mềm.

1.2 Bộ ADC trên AVR

ADC (Analog to Digital Converter) chuyển đổi tín hiệu tương tự sang số. Trên ATMega16 tích hợp 8 kênh ADC với độ rộng mỗi kênh là 10 bit, tức là điện áp sẽ được chuyển thành 1024 mức level dạng bậc thang.

ADC được ứng dụng rất nhiều như đo nhiệt độ, đọc giá trị điện áp, tính dòng diện, đọc phím nhấn theo điện trở, …

1.2.1 Thiết kế nguồn cho bộ ADC

Chân AVCC là chân cấp nguồn cho bộ ADC của ATmega16 (chân này không chung với VCC của chip).

Chân AREF là chân chọn điện áp tham chiếu ngoài của bộ ADC (AREF Max=5V), thực hiện so sánh tham chiếu để cho ra các mức logic tương ứng tỉ lệ %.

Ví dụ, AREF = 5V, ADC Input = 2.5V = 50% AREF thì giá trị ADC là 24/2=512.

Thiết kế nguồn như trên là lý tưởng cho các bộ ADC của AVR, nhưng trong thực tế nhằm giảm độ cồng kềnh và đã có các mạch nguồn ổn định thì chỉ cần một con tụ lọc 104.

1.2.2 Công thức chuyển đổi giá trị điện áp

ADC 10 bit: $\boldsymbol{Vin} = (\boldsymbol{Vref} * \boldsymbol{ADC})/\boldsymbol{1024}$

ADC 8 bit: $\boldsymbol{Vin} = (\boldsymbol{Vref} * \boldsymbol{ADC})/\boldsymbol{256}$

Trong đó: $\boldsymbol{Vref}$ là điện áp tham chiếu (đơn vị V)
ADC là giá trị sau chuyển đổi.

1.2.3 Các thanh ghi tham gia điều khiển ADC

- Thanh ghi ADMUX (ADC Multiplexer Selection Register)

  Là thanh ghi chọn điện áp tham chiếu và chọn kênh ADC

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Hình 14 . Thanh ghi ADMUX

Trong đó:

- Bit 7 và Bit 6 (REFS1 và REFS0) là 2 bit chọn điện áp tham chiếu để so sánh, ta có bảng sau:

REFS1	REFS0	Vref to ADC
0	0	AREF pin
0	1	AVCC pin i.e. Vcc 5 V
1	0	Reserved
1	1	Internal 2

Bảng 3 Chọn điện áp tham chiếu

- Bit 5 (ADLAR) là bit sắp xếp hai thanh ghi ADCH và ADCL, do bộ chuyển đổi ADC là 10 bit nên phải có 2 thanh ghi 8 bit để lưu giá trị chuyển đổi
- Bit 4:0 – MUX4:0 Bit chọn kênh ADC và có giá trị thập phân từ 0-7 ứng với kênh ADC0- ADC7. Ví dụ: Chọn kênh ADC 2 thì đặt 00010 trong MUX4:0.

- **Thanh ghi ADCSRA**
  - Bit 7 ADEN: cho phép bộ ADC hoạt động, nếu không set bit này lên 1 thì bộ ADC không hoạt động.
  - Bit 6 ADSC: là bit cho phép bộ ADC chuyển đổi, khi quá trình chuyển đổi hoàn tất bit này sẽ tự động được xóa về 0 và muốn bộ ADC chuyển đổi tiếp thì phải set lại bit này.
  - Bit 5 ADATE: là bit cho phép chuyển đổi liên tục, set bit này thì bộ ADC sẽ chuyển đổi liên tục.
  - Bit 4 ADIF là bit cờ ngắt, khi quá trình chuyển đổi hoàn tất thì bit này tự động được set lên 1, ta sẽ kiểm tra bị này để đảm bảo quá trình đọc ADC không bị lỗi.
  - Bit 3 ADIE là bit cho phép ngắt bộ ADC. Khi chuyển đổi hoàn tất, sẽ có một ngắt xảy ra nếu cho phép ngắt ADC.

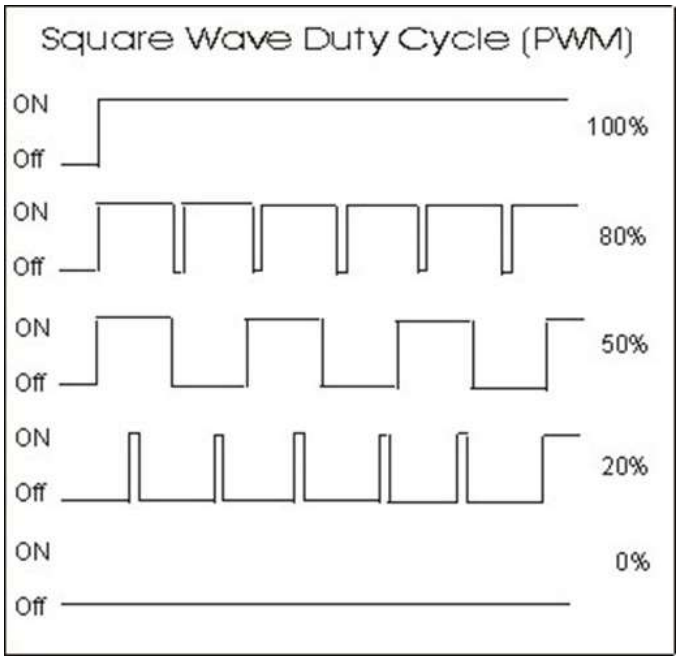
- ADPS2:0 là 3bit chọn xung nhịp cho bộ ADC, tốc độ chuyển đổi phụ thuộc vào 3bit này. Chế độ chuyển đổi 10bit sẽ chậm hơn 8 bit.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Bảng 4 Chọn xung nhịp cho bộ ADC

1.2.4 Giới thiệu về PWM

Điều chế PWM (Pulse-Width Modulation) là thay đổi duty cycle (khoảng thời gian tín hiệu ở mức cao) trong một chu kỳ (Time period) cố định, qua đó làm thay đổi điện áp trung bình cấp ra tải (thay đổi vận tốc động cơ DC, …)



Hình 15 . Dạng sóng vuông thể hiện các mức chia duty cycle

Trong đó: T là chu kì của xung, T1 là thời gian xung ở mức cao.

Hệ số điều chỉnh D (%) **D = T1/T**.

- Điện áp lớn nhất cấp cho tải: **Udmax**
- Điện áp trung bình cấp cho tải khi có PWM: **Ud = Udmax.D.** Bộ Timer1 của Atmega8 cung cấp cho chúng ta 2 kênh tạo xung điều rộng PWM:
    + Kênh A: tín hiệu ra ở OC1A (PORTB.1)
    + Kênh B: tín hiệu ra ở OC1B (PORTB.2)

Trong đồ án này, sử dụng chế độ Fast PWM, mode 14 (kênh A)

- Thiết lập dạng tín hiệu PWM ra trên thanh ghi TCCR1A
- Khi hoạt động, ban đầu chân OC1A ở mức cao (tùy vào dạng tín hiệu PWM ra bạn chọn ở trên), TCNT1 tăng giá trị từ 0 cho đến khi bằng giá trị trên thanh ghi OCR1A thì chân OC1A được xóa về 0. TCNT1 vẫn tiếp tục tăng đến khi bằng giá trị trong thanh ghi ICR1 thì TCNT1 reset về 0 và chân OC1A được kéo lên mức cao.

  Thiết lập:

    + Giá trị ICR1: là chu kỳ xung.
    + Giá trị OCR1A: là thời gian xung ở mức cao.

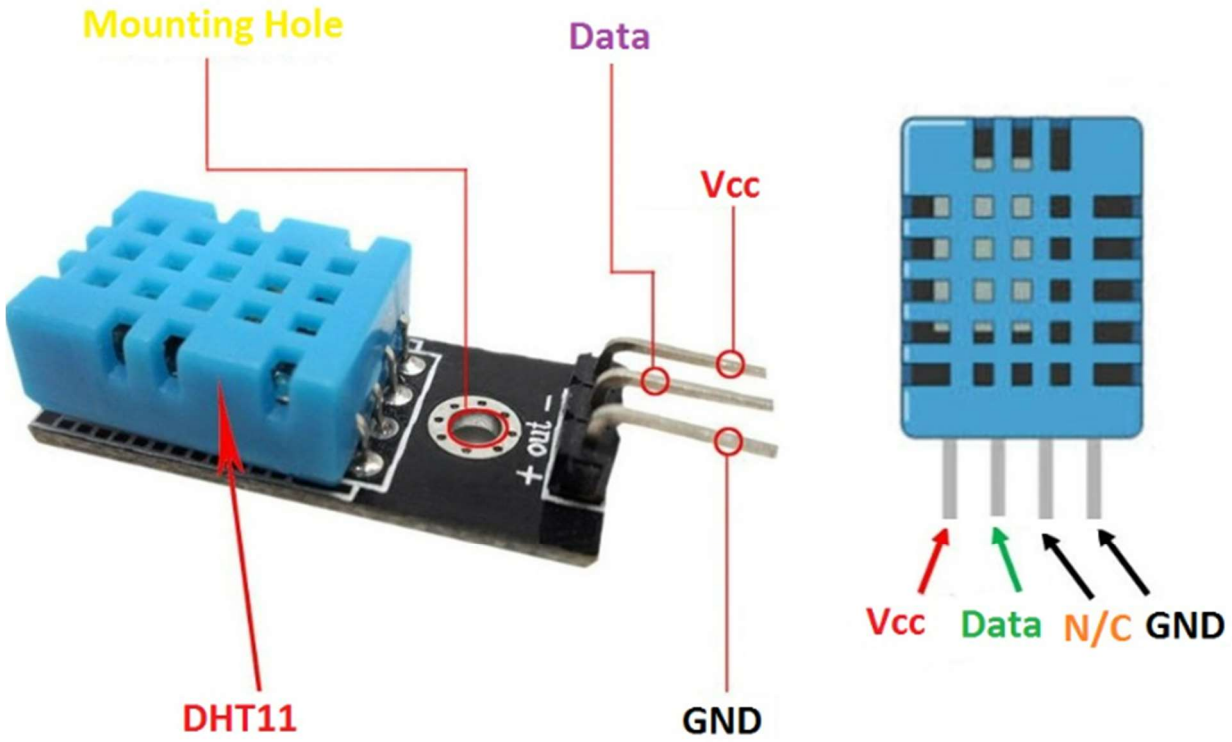
1.3 Các linh kiện cần thiết và thông số

Nhóm đã sử dụng những linh kiện sau để hoàn thành mạch:

Thiết bị	Chân	Chức năng / Mô tả
LCD I2C 16x2	A4 (SDA) A5 (SCL)	Giao tiếp I2C, hiển thị thông số nhiệt độ / độ ẩm / độ ẩm đất
Cảm biến DHT11	A2	Đo nhiệt độ và độ ẩm không khí
Cảm biến độ ẩm đất	A3	Đọc tín hiệu analog, xác định độ ẩm của đất
Relay điều khiển bơm	D6	Bật / tắt bơm điện
Nguồn cấp	5V / GND	Cấp điện cho toàn bộ hệ thống

Cảm biến nhiệt độ, độ ẩm môi trường DHT11

Với ưu điểm như hoạt động khá chính xác với sai số ít, kích thước nhỏ và giá thành thấp, IC cảm biến nhiệt độ LM35 là một trong những cảm biến tương tự được sử dụng rất nhiều trong các ứng dụng đo nhiệt độ thời gian thực.



Hình 16 . Hình ảnh cảm biến DHT11

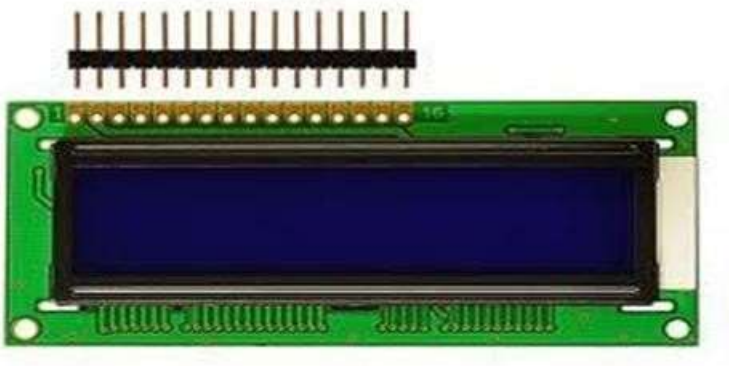
Số chân	Tên chân	Mô tả
1	Vcc	Nguồn 3.3-5V
2	OUT	Đầu ra thông qua dữ liệu nối tiếp
3	GND	Nối đất

Bảng 5 Chức năng chi tiết các chân của cảm biến DHT11

Thông số kỹ thuật DHT11

- Điện áp hoạt động: 3.3V đến 5V
- Dòng tiêu thụ trung bình: khoảng 60 µA
- Khoảng đo nhiệt độ: 0 °C đến 50 °C
- Độ chính xác đo nhiệt độ: ±2 °C
- Độ phân giải nhiệt độ: 1 °C
- Khoảng đo độ ẩm: 20% đến 90% RH (độ ẩm tương đối)
- Độ chính xác đo độ ẩm: ±5% RH
- Chu kỳ lấy mẫu: ít nhất 1 giây/lần (tối đa 1 Hz)
- Giao tiếp: một dây (single-wire) kỹ thuật số
- Kích thước: 15.5mm x 12mm x 5.5mm

Màn hình LCD(1602)

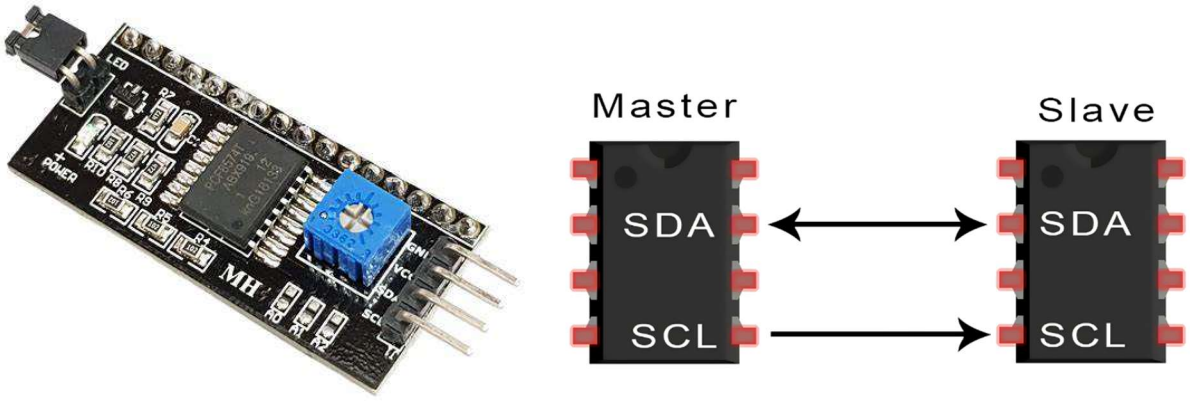


Hình 17 LCD1602

Thông số kỹ thuật:

- LCD 16x2 được sử dụng để hiển thị trạng thái hoặc các thông số
- LCD 16x2 có 16 chân trong đó 8 chân dữ liệu (D0 - D7) và 3 chân điều khiển (RS, RW, EN)
- 5 chân còn lại dùng để cấp nguồn và đèn nền cho LCD 16x2
- Các chân điều khiển giúp ta dễ dàng cấu hình LCD ở chế độ lệnh hoặc chế độ dữ liệu
- Chúng còn giúp ta cấu hình ở chế độ đọc hoặc ghi
- LCD 16x2 có thể sử dụng ở chế độ 4-bit hoặc 8-bit tùy theo ứng dụng ta đang làm.

Mạch mở rộng I2C



Hình 18 Mạch chuyển đổi I2C cho LCD

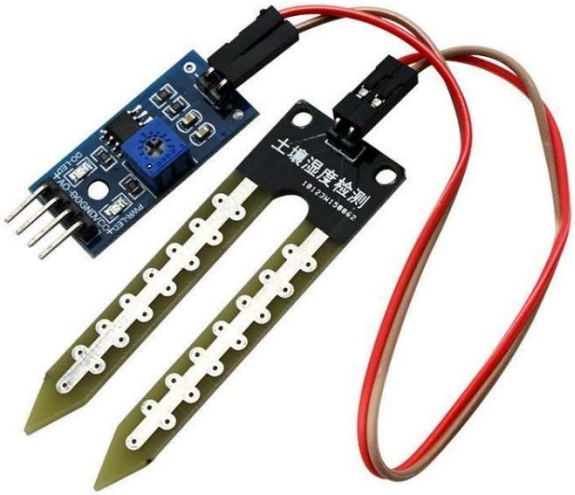
Thông số kỹ thuật

- GND: Nối đất.
- Chuẩn giao tiếp: I2C (2 dây: SDA, SCL).
- Dòng tải 33 - 5A.
- Tương thích với các vi điều khiển hỗ trợ I2C như: ATmega8/16/32, STM32, Arduino Uno/Nano, ESP32, v.v

Chức năng chính:

- Giao tiếp I2C tốc độ cao để truyền/nhận dữ liệu với vi điều khiển.
- Nạp code hoặc truyền dữ liệu qua I2C nếu được vi điều khiển hỗ trợ bootloader tương ứng.
- Cấp nguồn trực tiếp cho mạch vi điều khiển hoặc cảm biến I2C.
- Đọc dữ liệu từ các thiết bị I2C như EEPROM, RTC (DS1307/DS3231), cảm biến nhiệt độ/độ ẩm, v.v.
- Ghi cấu hình hoặc ghi dữ liệu xuống thiết bị I2C hỗ trợ ghi (như EEPROM).
- Bảo vệ giao tiếp bằng điện trở pull-up tích hợp sẵn nếu có.

Cảm biến đất TM *(Soil Moisture Sensor)*



Hình 21. Cảm biến độ ẩm đất

Thông số kỹ thuật cảm biến độ ẩm đất

- Điện áp hoạt động: 3.3V – 5V
- Dòng tiêu thụ: < 20 mA
- Tín hiệu xuất ra:
    - Analog: điện áp tỷ lệ với độ ẩm đất
    - Digital: mức HIGH/LOW tùy theo ngưỡng so sánh
- Cấu tạo: 2 thanh kim loại (loại điện trở) hoặc đầu dò không tiếp xúc (loại điện dung)
- Kích thước module: khoảng 60mm x 20mm (tùy loại)

Chức năng chính

- Đo độ ẩm của đất trồng, xác định mức khô/ẩm để tự động tưới cây.
- Xuất tín hiệu analog giúp vi điều khiển đọc giá trị chính xác hơn để điều khiển bơm nước.
- Xuất tín hiệu digital (qua mạch so sánh) để đưa ra cảnh báo hoặc kích relay.
- Tương thích với nhiều loại vi điều khiển như: Arduino Uno/Nano, ESP8266, ESP32, STM32, v.v.
- Có thể dùng kết hợp LCD/I2C để hiển thị độ ẩm đất theo thời gian thực.

Relay và máy bơm nhỏ 6V

Hình 22. Relay Module

Thông số kỹ thuật Relay

- Điện áp điều khiển: 5V DC (cũng có loại 3.3V hoặc 12V)
- Dòng điều khiển: 15 – 20 mA

- Tín hiệu điều khiển: TTL (mức HIGH/LOW từ vi điều khiển)
- Số kênh: 1 kênh (1 relay)
- Loại relay: SPDT (Single Pole Double Throw – 1 thường đóng, 1 thường hở)
- Tải tối đa (tiếp điểm relay):
    - AC: 250V – 10A
    - DC: 30V – 10A
- Cách ly quang (optocoupler): Có (trong các loại relay cách ly)
- Đèn LED hiển thị: Có (báo trạng thái relay ON/OFF)
- Kích thước: ~ 50mm x 26mm x 18mm (tùy loại)

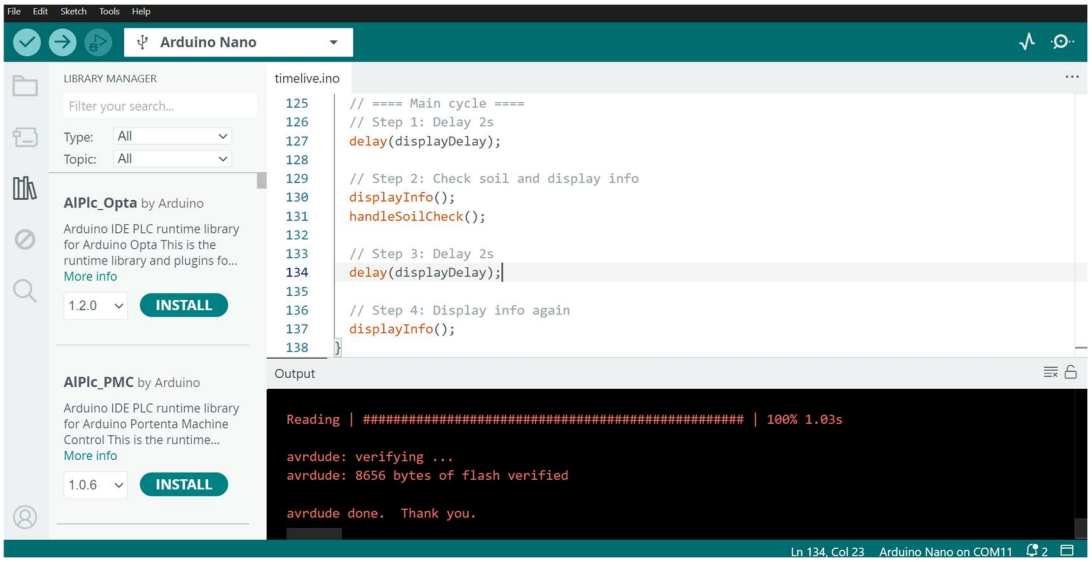
Chức năng chính

- Đóng/ngắt thiết bị điện xoay chiều hoặc một chiều như đèn, quạt, máy bơm...
- Tách biệt mạch điều khiển logic (Arduino, ESP32) với tải điện áp cao (qua tiếp điểm relay).
- Điều khiển tự động qua vi điều khiển, dễ tích hợp với cảm biến (đất, nhiệt độ, chuyển động...)
- Bảo vệ mạch điều khiển khỏi dòng điện hoặc áp cao .

1.4 Các phần mềm sử dụng

1.4.1 Arduino IDE

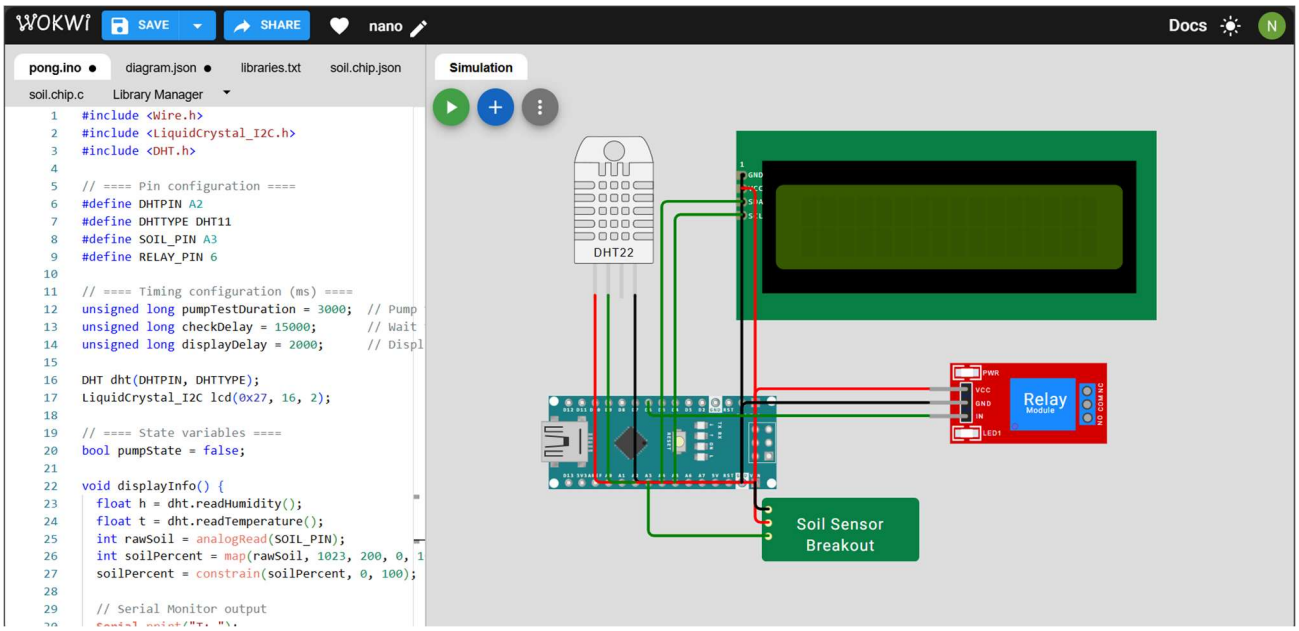
Arduino một trình biên dịch code C, môi trường phát triển tích hợp và tạo chương trình tự động được thiết kế cho Arduino nói chung.



Hình 23 Giao diện của phần mềm Arduino IDE

1.4.2 Phần mềm Wokwi kiểm tra và chạy thử mô phỏng trước khi lắp mạch

Cho phép chạy thử dưới dạng minh họa dễ hiểu vào vi điều khiển. Chương trình sau khi chạy có giao diện như sau:

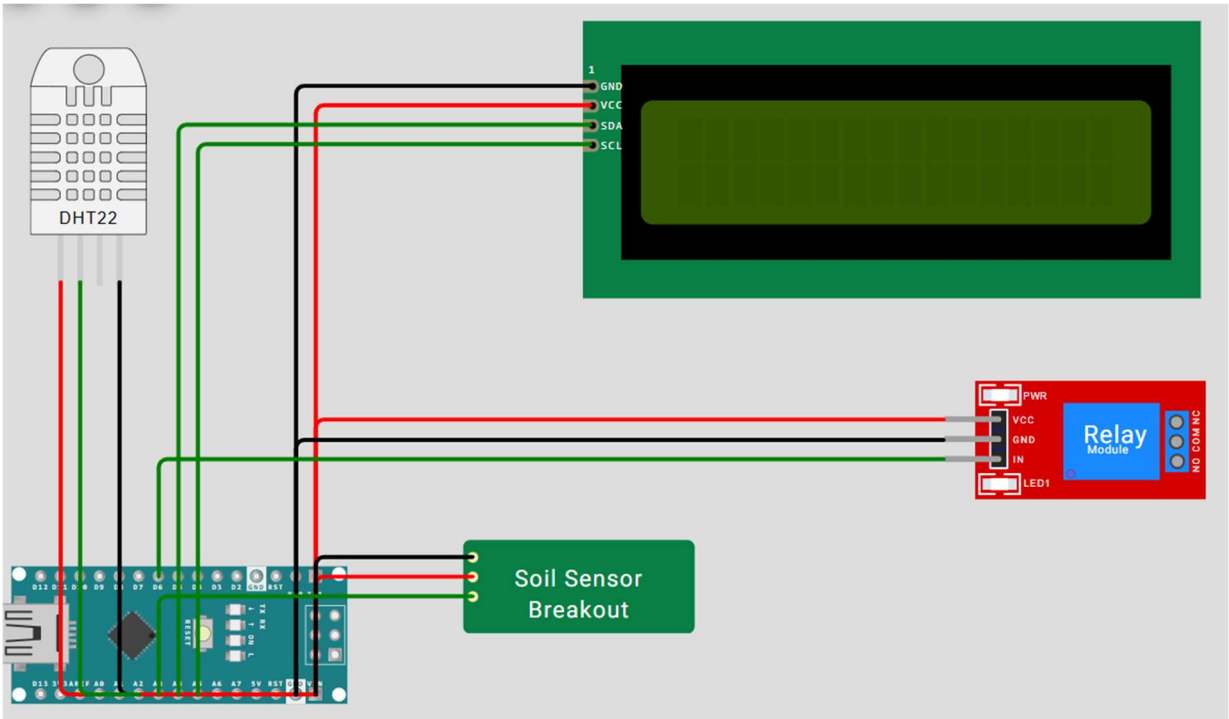


Hình 24 Giao diện web chạy thử mạch

Sau đó ta cần chạy thử, nếu thành công có thể mua và lắp mạch thật rồi nạp code.

II. MẠCH TƯỚI CÂY TỰ ĐỘNG

2.1 Sơ đồ nguyên lý

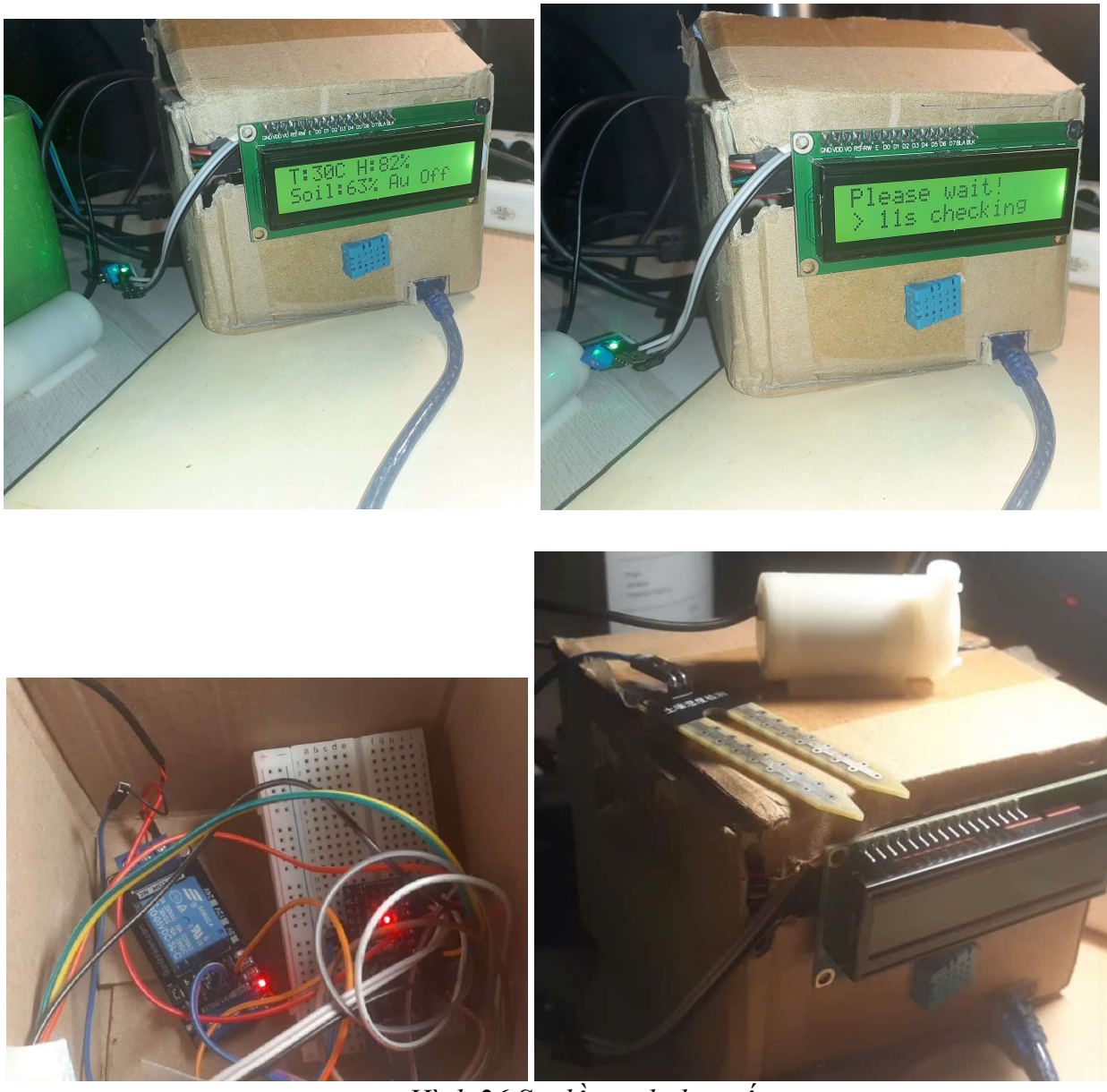


Hình 25 Sơ đồ nguyên lý của mạch

II. GHÉP NỐI LINH KIỆN VÀ KẾT QUẢ

3.1 Lắp mạch và nạp chương trình

Sau khi lắp xong mạch Kit trắng, kết nối chân VCC và GND của module cảm biến DHT11 với lỗ cắm 5V và 0V của mạch Kit để cấp nguồn cho cảm biến. Tiếp đó kết nối chân DATA của module DHT11 chân A2, Relay vào chân D6, Soil Sensor vào A3 của vi điều khiển Arduino Nano để đưa đầu vào là tín hiệu tương tự của cảm biến tới vi điều khiển để chuyển đổi thành tín hiệu số hiển thị lên LCD.



Hình 26 Sơ đồ mạch thực tế

3.2 Chương trình phần mềm

File main.ino:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

// ==== Pin configuration ====
#define DHTPIN A2
#define DHTTYPE DHT11
#define SOIL_PIN A3
#define RELAY_PIN 6

// ==== Timing configuration (ms) ====
```

```
unsigned long pumpTestDuration = 3000;  // Pump test duration (ms)
unsigned long checkDelay = 15000;       // Wait time after pump test (ms)
unsigned long displayDelay = 2000;      // Display info delay (ms)

DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

// ==== State variables ====
bool pumpState = false;

void displayInfo() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  int rawSoil = analogRead(SOIL_PIN);
  int soilPercent = map(rawSoil, 1023, 200, 0, 100);
  soilPercent = constrain(soilPercent, 0, 100);

  // Serial Monitor output
  Serial.print("T: ");
  if (isnan(t)) {
    Serial.print("Error");
  } else {
    Serial.print(t);
  }
  Serial.print("C, H: ");
  if (isnan(h)) {
    Serial.print("Error");
  } else {
    Serial.print(h);
  }
  Serial.print("%, Soil: "); Serial.print(soilPercent);
  Serial.print("%, Mode: AUTO");
  Serial.print(", Pump: "); Serial.println(pumpState ? "ON" : "OFF");

  // LCD Display
  lcd.clear();
  lcd.setCursor(0, 0);
  if (isnan(t) || isnan(h)) {
    lcd.print("DHT11 Error!");
  } else {
    lcd.print("T:");
    lcd.print((int)t);
    lcd.print("C H:");
    lcd.print((int)h);
    lcd.print("%  ");
  }
  lcd.setCursor(0, 1);
  lcd.print("Soil:");
  lcd.print(soilPercent);
  lcd.print("% ");
```

```
  lcd.print("Au ");
  lcd.print(pumpState ? "On" : "Off");
}

void handleSoilCheck() {
  int raw = analogRead(SOIL_PIN);
  int soilPercent = map(raw, 1023, 200, 0, 100);
  soilPercent = constrain(soilPercent, 0, 100);

  if (soilPercent < 25) {
    Serial.println("[AUTO] Soil dry. Starting pump...");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Pump running 3s...");
    digitalWrite(RELAY_PIN, HIGH);
    pumpState = true;
    delay(pumpTestDuration);
    digitalWrite(RELAY_PIN, LOW);
    pumpState = false;

    Serial.println("[AUTO] Waiting for check...");

    // Display countdown during wait
    for (int i = checkDelay / 1000; i > 0; i--) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Please wait!  ");
      lcd.setCursor(0, 1);
      lcd.print("> ");
      lcd.print(i);
      lcd.print("s checking   ");
      delay(1000);
    }

    // Check soil again after wait
    raw = analogRead(SOIL_PIN);
    soilPercent = map(raw, 1023, 200, 0, 100);
    soilPercent = constrain(soilPercent, 0, 100);
    Serial.print("[AUTO] Soil after pump: ");
    Serial.print(soilPercent);
    Serial.println("%");
  }
}

void setup() {
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  dht.begin();
  lcd.init();
```

```
  lcd.backlight();
  Serial.begin(9600);

  // Startup display
  lcd.setCursor(1, 0);
  lcd.print("Irrigation ");
  lcd.setCursor(1, 1);
  lcd.print("System");
  delay(3000);
  lcd.clear();
}

void loop() {
  // ==== Main cycle ====
  // Step 1: Delay 2s
  delay(displayDelay);

  // Step 2: Check soil and display info
  displayInfo();
  handleSoilCheck();

  // Step 3: Delay 2s
  delay(displayDelay);

  // Step 4: Display info again
  displayInfo();
}
```

File LCD.h:

```
#ifndef LCD_H_
#define LCD_H_

#define F_CPU 8000000UL                 /* Define CPU Frequency e.g. here its 8MHz
*/
#include <avr/io.h>                     /* Include AVR std. library file */
#include <util/delay.h>                 /* Include Delay header file */

#define LCD_Data_Dir DDRC               /* Define LCD data port direction */
#define LCD_Command_Dir DDRD            /* Define LCD command port direction
register */
#define LCD_Data_Port PORTC             /* Define LCD data port */
#define LCD_Command_Port PORTD          /* Define LCD data port */
#define EN PD7                          /* Define Enable signal pin */
#define RW PD5                          /* Define Read/Write signal pin */
#define RS PD6                          /* Define Register Select (data
reg./command reg.) signal pin */
```

```
void LCD_Command (char);                /* LCD command write function */
void LCD_Char (char);                   /* LCD data write function */
void LCD_Init (void);                   /* LCD Initialize function */
void LCD_String (char*);                /* Send string to LCD function */
void LCD_String_xy (char,char,char*);   /* Send row, position and string to LCD
function */
void LCD_Clear(void);                       /* Clear LCD*/

void LCD_Command (char cmd)                         /* LCD command write function
*/
{
    LCD_Data_Port = cmd;                            /* Write command data to LCD
data port */
    LCD_Command_Port &= ~((1<<RS)|(1<<RW));         /* Make RS LOW (command reg.),
RW LOW (Write) */
    LCD_Command_Port |= (1<<EN);                    /* High to Low transition on EN
(Enable) */
    _delay_us(1);
    LCD_Command_Port &= ~(1<<EN);
    _delay_ms(3);                                   /* Wait little bit */
}

void LCD_Char (char char_data)                      /* LCD data write function */
{
    LCD_Data_Port = char_data;                      /* Write data to LCD data port
*/
    LCD_Command_Port &= ~(1<<RW);                   /* Make RW LOW (Write) */
    LCD_Command_Port |= (1<<EN)|(1<<RS);            /* Make RS HIGH (data reg.) and
High to Low transition on EN (Enable) */
    _delay_us(1);
    LCD_Command_Port &= ~(1<<EN);
    _delay_ms(1);                                   /* Wait little bit */
}

void LCD_Init (void)                                /* LCD Initialize function */
{
    LCD_Command_Dir |= (1<<RS)|(1<<RW)|(1<<EN);     /* Make LCD command port
direction as o/p */
    LCD_Data_Dir = 0xFF;                            /* Make LCD data port direction
as o/p */

    _delay_ms(20);                                  /* LCD power up time to get
things ready, it should always >15ms */
    LCD_Command(0x38);                              /* Initialize 16X2 LCD in 8bit
mode */
    LCD_Command(0x0C);                              /* Display ON, Cursor OFF
command */
    LCD_Command(0x06);                              /* Auto Increment cursor */
    LCD_Command(0x01);                              /* Clear LCD command */
```

```
    LCD_Command(0x80);                              /* 8 is for first line and 0 is
for 0th position */
}

void LCD_String (char *str)                         /* Send string to LCD function
*/
{
    int i;
    for(i=0;str[i]!=0;i++)                          /* Send each char of string
till the NULL */
    {
        LCD_Char(str[i]);                           /* Call LCD data write */
    }
}

void LCD_String_xy (char row, char pos, char *str)  /* Send string to LCD function
*/
{
    if (row == 1)
        LCD_Command((pos & 0x0F)|0x80);             /* Command of first row and
required position<16 */
    else if (row == 2)
        LCD_Command((pos & 0x0F)|0xC0);             /* Command of Second row and
required position<16 */
    LCD_String(str);                                /* Call LCD string function */

}

void LCD_Clear(void)
{
    LCD_Command(0x01);                              /* clear display */
    LCD_Command(0x80);
}

#endif /* LCD_H_ */
```

Sau khi debug, phần mềm Arduino IDE sẽ nạp vào vi xử lý và chạy mạch như trên.

3.3 Kết quả

Hình 27 Mạch sau khi chạy

IV. KẾT LUẬN

4.1 Kết quả

Mạch hoàn thành đúng thời hạn và chạy đúng như mô phỏng

4.2 Nhận xét

Trong quá trình thực hiện em đã có cơ hội tìm hiểu và vận dụng được các kiến thức đã học và tìm hiểu vào ứng dụng sản phẩm thực tế. Từ đó mà nắm chắc hơn kiến thức về vi điều khiển Arduino và Arduino Nano, cũng như nắm chắc hơn về cách xây dựng các mạch điện tử thông qua các phần mềm mô phỏng như Proteus, Arduino IDE.

Qua đồ án này, em nhận thấy đề tài không chỉ hữu ích trong thực tiễn mà qua đó giúp tiếp thu nhiều hơn các kiến, thức cơ bản của ngành điện tử, cũng như giúp các sinh viên nâng cao khả năng thuyết trình, tìm kiếm, nghiên cứu các tài liệu. Đây là cơ sở giúp em có nền tảng vững chắc hơn trong quá trình học tập và nghiên cứu sau này.

Một lần nữa em xin chân thành cảm ơn thầy Tào Văn Cường đã tận tình hướng dẫn và tạo mọi điều kiện thuận lợi nhất giúp đỡ chúng em trong quá trình hoàn thành đồ án này!

TÀI LIỆU THAM KHẢO

[1] https://arduinokit.vn/su-dung-cam-bien-do-am-dat-soil-moisture-sensor-voi-arduino/

[2] https://www.alldatasheet.vn/datasheet-pdf/view/1424860/ETC/ARDUINO-NANO.h