ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG ĐIỆN-ĐIỆN TỬ

****

ĐỒ ÁN II

Đề tài : Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự LM35

Giảng viên hướng dẫn: TS. Dương Thanh Phương

Nhóm sinh viên:

Nguyễn Văn Tài	20214082
Nguyễn Tiến Trường	20214121
Phạm Văn Hoàn	20213929
Trần Hoàng Nam	20214021
Nghiêm Quang Minh	20214004
Đào Đức Hiệp	20213911
Phùng Xuân Quân	20214064
Nguyễn Đức Minh	20210578

Hà Nội, 01/2025

BẢNG PHÂN CÔNG NHIỆM VỤ

STT	Họ và tên	Nhiệm vụ
1	Nguyễn Văn Tài 20214082	Lập trình vi điều khiển
2	Nguyễn Tiến Trường 20214121	Thiết kế phần mềm
3	Phạm Văn Hoàn 20213929	Thiết kế lại PCB dựa trên kit
4	Trần Hoàng Nam 20214021	Mô phỏng mạch trên Proteus
5	Nghiêm Quang Minh 20214004	Làm báo cáo, tổng hợp thông tin
6	Đào Đức Hiệp 20213911	Tìm hiểu cấu hình cho vi điều khiển
7	Phùng Xuân Quân 20214064	Mô phỏng mạch trên Proteus
8	Nguyễn Đức Minh 20210578	Thiết kế lại PCB dựa trên kit

LỜI NÓI ĐẦU

Việt Nam ta ngày một phát triển và giàu mạnh. Trong Thời kì cách mạng 4.0, là bước ngoặt quan trọng thay đổi đất nước,để chúng ta - con người Việt có cơ hội nắm bắt nhiều thành tựu vĩ đại của thế giới, đặc biệt là về các lĩnh vực khoa học kĩ thuật nói chung và ngành Điện Tử - Viễn thông nói riêng.

Thế hệ trẻ chúng ta không tự mình phấn đấu học hỏi không ngừng thì chúng ta sẽ sớm lạc hậu và nhanh chóng thụt lùi. Nhìn ra được điều đó "Đại học Bách khoa Hà Nội" đã sớm chủ trương hình thức đào tạo sâu rộng, từ thấp đến cao. Để tăng chất lượng học tập của sinh viên nhà trường đã tổ chức cho sinh viên làm các Đồ Án II nhằm tạo nên tảng vững chắc cho sinh viên khi ra trường, đáp ứng nhu cầu tuyển dụng việc làm.

Với sự tiến bộ không ngừng của khoa học kỹ thuật, đặc biệt là ngành điện tử - viễn thông đã ứng dụng rất nhiều trong công nghiệp. Trong lĩnh vực điều khiển, từ khi công nghệ chế tạo loại vi mạch lập trình phát triển đã đem đến các kỹ thuật điều khiển hiện đại có nhiều ưu điểm so với việc sử dụng các mạch điều khiển được lắp ráp từ các linh

kiện rời như kích thước mạch nhỏ, gọn, giá thành rẻ, độ làm việc tin cậy và công suất tiêu thụ thấp ... Ngày nay lĩnh vực điều khiển đã được ứng dụng rộng rãi trong các thiết bị, sản phẩm phục vụ cho nhu cầu sinh hoạt hằng ngày của con người như máy giặt, đồng hồ điện tử, ti vi ... nhằm giúp cho đời sống ngày càng hiện đại và tiện lợi hơn. Đề tài ứng dụng vi điều khiển trong đời sống thực tế rất phong phú và đa dạng nhằm đáp ứng cho cuộc sống tiện nghi của con người. Vì vậy chúng em đã chọn đề tài "*Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự* " làm đề tài đồ án kỹ thuật II của chúng em.

Trong quá trình làm đồ án của mình, chúng em đã cố gắng để hoàn thiện đề tài một cách tốt nhất. Nhưng với kiến thức và sự hiểu biết còn hạn chế nên chác chắn sẽ không tránh khỏi những thiếu sót. Chúng em mong thầy đóng góp ý kiến để đề tài của chúng em được hoàn thiện hơn.

Em xin gửi lời cảm ơn sâu sắc đến thầy Dương Thanh Phương đã nhiệt tình hướng dẫn, giúp đỡ và tạo điều kiện tốt nhất để chúng em hoàn thành đồ án này.

LỜI CAM ĐOAN

Nhóm chúng em xin cam đoan toàn bộ nội dung được trình bày trong đồ án *Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự* là kết quả quá trình tìm hiểu và nghiên cứu của nhóm. Các dữ liệu được nêu trong đồ án là hoàn toàn trung thực, phản ánh đúng kết quả đo đạc thực tế. Mọi thông tin trích dẫn đều tuân thủ các quy định về sở hữu trí tuệ; các tài liệu tham khảo đều được liệt kê rõ ràng. Nhóm em xin chịu hoàn toàn trách nhiệm với những nội dung được viết trong đồ án này.

Hà Nội, 02 tháng 01 năm 2025

MỤC LỤC

DANH MỤC KÍ HIỆU VIẾT TẮT

Từ viết tắt	Tiếng Việt
VĐK	Vi điều khiển

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

TÓM TẮT ĐỒ ÁN

Đồ án thiết kế II tập trung vào trình bày về việc tìm hiểu họ vi điều khiển AVR, trong đồ án này vi điều khiển được sử dụng đó là Atmega 16. Qua đó tìm hiểu nguyên lí hoạt động của bộ kit, các ứng dụng của bộ kit. Trong báo cáo này chúng em sẽ trình bày ứng dụng của bộ kit trong việc đo nhiệt độ bằng cảm biến tương tự LM35 qua đề tài "*Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự*". Đồ án gồm có bốn chương:

Chương I. Cơ sở lý thuyết chung. Giới thiệu tổng quan về đề tài, về họ vi điều khiển AVR, các ngôn ngữ thực hiện để lập trình vi điều khiển.

Chương II. Chi tiết cấu hình của mạch kit. Trình bày chi tiết về cấu trúc mạch kit được sử dụng trong đồ án, các thông số của mạch kit, kết nối với các thiết bị ngoại vi như LCD, UART-USB.

Chương III. Thực hành lập trình cho vi điều khiển. Tiến hành nạp code và chạy thử vi điều khiển, kiểm tra hiệu năng và lỗi mạch.

Chương IV. Vận dụng vào kiến thức thực tế. Chương này sẽ đi vào tìm hiểu thiết kế và giải quyết các mục tiêu của đề tài là sử dụng cảm biến nhiệt độ để đo và in kết quả ra LCD theo những yêu cầu kĩ thuật cho trước.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT CHUNG

1.1 Giới thiệu chung

Với học phần Đồ án II tìm hiểu về vi điều khiển họ AVR, nội dung được thiết kế để nâng cao năng lực chuyên môn cho các sinh viên; giúp liên kết các khối kiến thức về điện tử tương tự, điện tử số, kỹ thuật vi xử lý, xử lý số tín hiệu, thông tin số, v.v. nhằm hoàn thiện khả năng vận dụng kiến thức vào thực tế.

Trong học phần này, nội dung công việc cần thực hiện bao gồm: làm quen công cụ thiết kế mạch điện, thực hành lập trình phần cứng và xây dựng một ứng dụng cơ bản với các vi mạch có thể lập trình được. Để đảm bảo tiến độ công việc, chúng em và các bạn sinh viên khác đã được viện hỗ trợ một số phương tiện cơ bản để triển khai công việc trên nền tảng xây dựng sẵn, có thể kế thừa.

Sau khi tiếp cận nhanh với vi điều khiển (VĐK) thông qua việc xây dựng một số ứng dụng trên VĐK họ AVR cụ thể trong bài báo cáo này chúng em dùng Atmega16A. chúng em sẽ tiến hành giải quyết một vấn đề cụ thể - thiết kế mạch đo nhiệt độ và hiển thị trên LCD sử dụng cảm biến tương tự và cụ thể đó là cảm biến LM35 một cảm biến nhiệt độ tương tự được ứng dụng phổ biến trong việc cảm biến nhiệt độ.

1.2 Mạch Kit cho VĐK họ AVR

AVR là một VĐK 8-bit khá mạnh và thông dụng tại thị trường Việt Nam. Với tốc độ xung nhịp tối đa lên tới 16 Mhz, bộ nhớ chương trình tối đa tới 256 kB, và rất nhiều chức năng ngoại vi tích hợp sẵn, VĐK họ AVR có thể đáp ứng tốt cho nhiều ứng dụng trong thực tế, từ đơn giản đến phức tạp.

Với Kit phát triển sử dụng trong học phần Đồ án II (hình 1.1) được Viện Điện tử - Viễn thông thiết kế riêng để đảm bảo tính hiệu quả trong quá trình đào tạo. Bộ kit có thể được thử nghiệm với các ứng dụng cơ bản sau:

- Điều khiển cổng ra số, với LED đơn và LED 7 thanh
- Đọc trạng thái logic đầu vào số, từ bán phím và giắc cắm mở rộng
- Đo điện áp tương tự với biến trở vi chỉnh và bộ ADC 10-bit

- Điều khiển màn hình tinh thể lỏng, với màn hình LCD dạng text
- Giao tiếp với máy tính qua chuẩn UART ↔ USB
- Thử nghiệm các ngắt ngoài, thử khả năng điều chế độ rộng xung.

- Nhiều ứng dụng điều khiển các chức năng tích hợp sẵn trong VĐK như: vận hành các bộ định thời (Timer) và bộ đếm (Counter), đọc ghi EEPROM, lập trình các ngắt chương trình, thiết lập Watchdog, v.v.

Hình 1.1 Mạch Kit phát triển và các phụ kiện

Ngoài ra, bằng việc kết nối giữa các mô-dun mở rộng, mạch Kit hoàn toàn có thể thực hiện các ứng dụng phức tạp hơn như:

- Đo tham số môi trường: nhiệt độ, độ ẩm, độ sáng, v.v.
- Điều khiển tải cơ bản: đèn báo, van điện tử, động cơ DC, động cơ bước, v.v.
- Điều khiển hiển thị cơ bản: LED ma trận, LCD ma trận, màn hình cảm ứng, v.v.
- Giao tiếp I2C và SPI: IC thời gian thực, IC EEPROM, cảm biến gia tốc, v.v.
- Ứng dụng tổng hợp: đo và duy trì sự ổn định các tham số mô trường; số hóa và xử lý tín hiệu âm thanh, điều khiển robot hoặc xe tự hành, v.v.

1.3 Ngôn ngữ lập trình và phần mềm

AVR nói chung chung cũng như Atmega16 nói riêng hỗ trợ 2 ngôn ngữ lập trình thông dụng là Assembly và C. Việc lập trình bằng Assembly giúp chương trình nhỏ gọn nhưng khá phức tạp do gần với ngôn ngữ máy. Lập trình bằng C tuy cho chương trình có dung lượng lớn hơn so với khi lập trình bằng Assembly, nhưng đổi lại dễ dàng hơn trong việc code và debug.

Để lập trình cho AVR, có khá nhiều trình biên dịch, ví dụ như AVR studio, WinAVR, codevisionAVR…

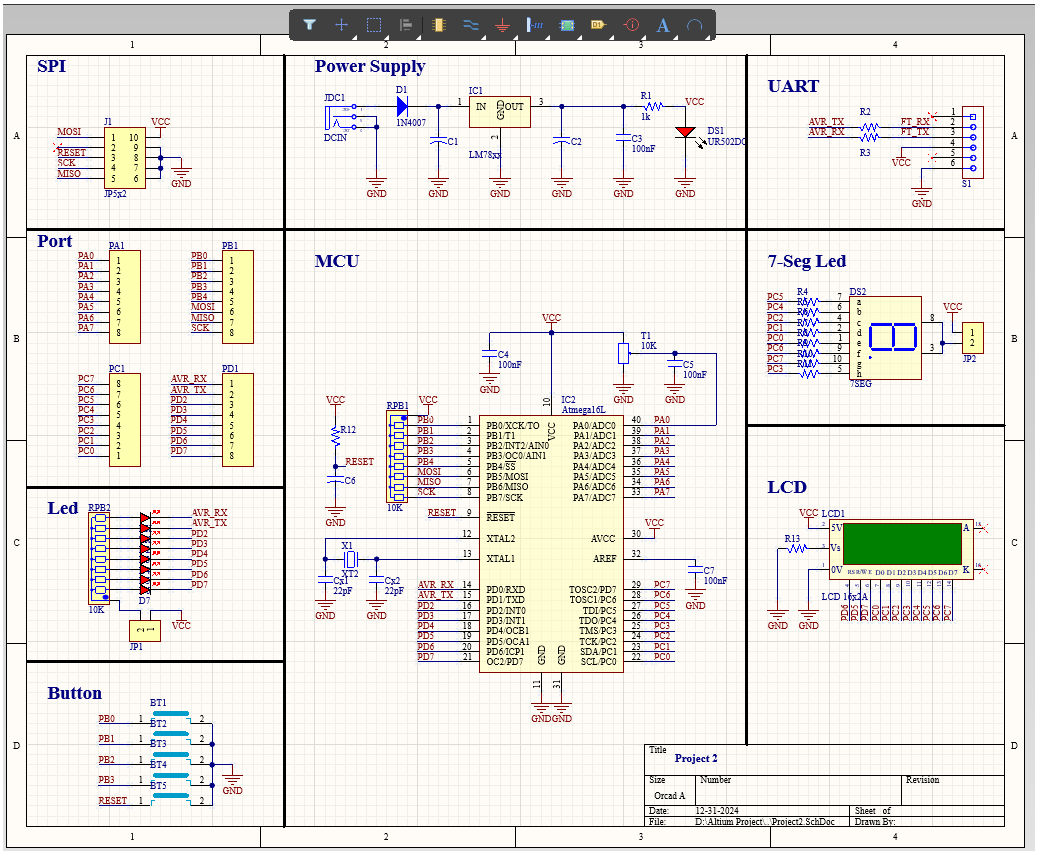
Trong nội dung về học phần Đồ án II, chúng em sẽ sử dụng các ngôn ngữ C/C++ và các công cụ hỗ trợ sau:

- IDE là Microchip Studio 7.0 để tiến hành soạn thảo và biên dịch code sang mã máy.
- Công cụ để quản lý, lưu trữ, trao đổi mã nguồn giữa các thành viên trong nhóm là Github.
- Phần mềm nạp mã máy là PROGISP (phiên bản 1.72).
- Phần mềm giao tiếp giữa máy tính và VĐK là Terminal trong extension của Visual Studio Code.
- Ngoài ra, trong phần vận dụng chúng em có sử dụng thêm phần mềm Proteus 8.11 để mô phỏng mạch
- Phần mềm Altium Designer 24 để thiết kế mạch in PCB.

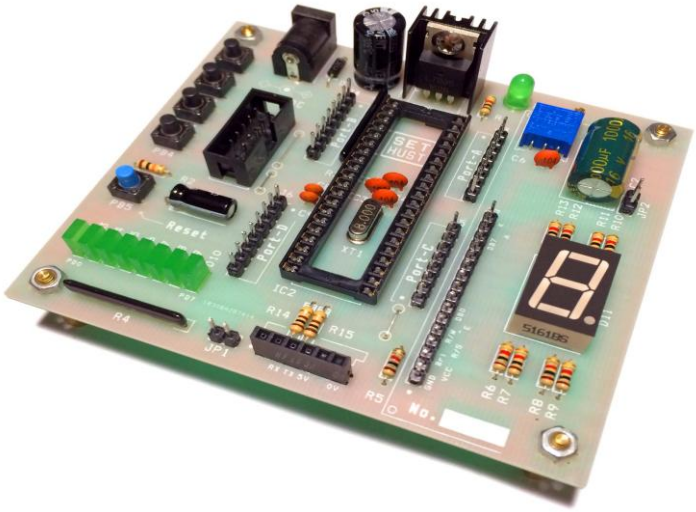
CHƯƠNG 2. CHI TIẾT CẤU HÌNH CỦA MẠCH KIT

2.1 Cấu trúc của mạch Kit

Mạch Kit được cung cấp để đáp ứng các ứng dụng cơ bản có cấu trúc và chức năng của từng linh kiện quan trọng được nêu trong bảng dưới đây. Đầu tiên ta sẽ tìm hiểu về sơ đồ nguyên lý của bộ kit cùng với đó là mạch nguyên lý được vẽ bởi Altium một trong những công cụ vẽ mạch đi dây cũng như để xuất bản thiết kế mạch mạnh và phổ biến nhất hiện nay. Thông qua môn học đồ án II này chúng em có thể thành thạo hơn việc sử dung altium để vẽ và thiết kế mạch điện tử. Tự mình tạo ra một bản mạch mà mình mong muốn.



Hình 2.1 Sơ đồ nguyên lý của bộ kit



Hình 2.2 Bộ kit hoàn thiện

Bảng 2.1 Linh kiện quan trọng của mạch Kit và chức năng tương ứng

STT	Tên linh kiện	Chức năng
1	Giắc cắm nguồn	Nhận nguồn điện 9-12 VDC cấp cho mạch Kit
2	IC ổn áp 7805	Hạ 9-12 VDC xuống 5 VDC và giữ ổn định mức điện áp này để cấp cho toàn mạch
3	LED báo nguồn	Báo nguồn (sáng: có nguồn 5 VDC, tắt: mất nguồn)
4	VĐK Atmega16	Điều khiển hoạt động của mạch theo mã nguồn đã được nạp
5	Thạch anh	Quyết định tần số xung nhịp cấp cho VĐK
6	Nút ấn Reset	Khởi động lại VĐK
7	Giắc ISP	Kết nối mạch nạp để nạp mã nguồn cho VĐK
8	Nhóm 4 phím ấn	Nhận lệnh điều khiển từ người sử dụng
9	Giắc cắm 8 chân	Nối tới 8 chân vào/ra (Port-A) của VĐK
10	Giắc cắm 8 chân	Nối tới 8 chân vào/ra (Port-B) của VĐK
11	Giắc cắm 8 chân	Nối tới 8 chân vào/ra (Port-C) của VĐK
12	Giắc cắm 8 chân	Nối tới 8 chân vào/ra (Port-D) của VĐK

13	Dãy LED đơn	Báo trạng thái logic của 8 chân ở Port-D của VĐK
14	Jumper dãy LED đơn	Cho phép hoặc vô hiệu hóa dãy LED đơn
15	LED 7 thanh	Hiển thị số 0-9 và một vài kí tự do người dùng định nghĩa
16	Jumper LED 7 thanh	Cho phép hoặc vô hiệu hóa dãy LED 7 thanh
17	Giắc cắm LCD	Kết nối tới màn hình LCD1602
18	Biến trở vi chỉnh	Điều chỉnh trơn và liên tục, từ 0 đến 5 VDC, từ mức điện áp tại đầu vào ADC0 của bộ ADC (chân PA0)
19	Giắc UART-USB	Kết nối module chuyển đổi UART-USB (còn gọi là COM-USB)

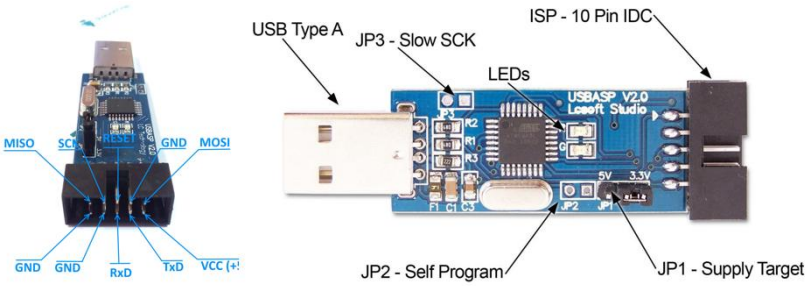
2.2 Các thông số chính của Kit

Các thông số kĩ thuật của Kit:

- Điện áp nguồn:
  - Tiêu chuẩn: 9-12 VDC
  - Giới hạn: 7-18 VDC
- Dòng điện tiêu thụ:
  - Khi không có module mở rộng, toàn bộ LED chỉ thị I/O tắt: 18mA
  - Khi có LCD và module USB, LED chỉ thị I/O bị vô hiệu hóa: 22mA
  - Khi có LCD và module USB, toàn bộ LED chỉ thị I/O sáng: 80mA
- Mạch có khả năng tự bảo vệ khi bị lắp ngược cực tính nguồn
- Mức logic các cổng I/O: TTL (5V)
- Điện áp tương tự vào các chân ADC: 0 – 5V
- Loại VĐK được hỗ trợ: Atmega16, Atmega32 và tương đương
- Cổng I/O mở rộng: 4 giắc cắm (loại 8 chân) ứng với 4 port
- Hỗ trợ màn hình LCD: dạng text, giao tiếp 8 bit hay 4 bit
- Hỗ trợ module USB: UART-USB hay COM-USB (mức 5 VDC)
- Xung nhịp tích hợp sẵn: thạch anh 8 Mhz

2.3 Mạch nạp mã nguồn

Mạch nạp mã nguồn cho VĐK trong Kit là loại mạch nạp ISP thông dụng. Trong bộ Kit này sử dụng mạch nạp ISP chuẩn 10 chân như hình dưới.



Hình 2.3 Mạch nạp ISP chuẩn 10 chân sử dụng cho bộ Kit

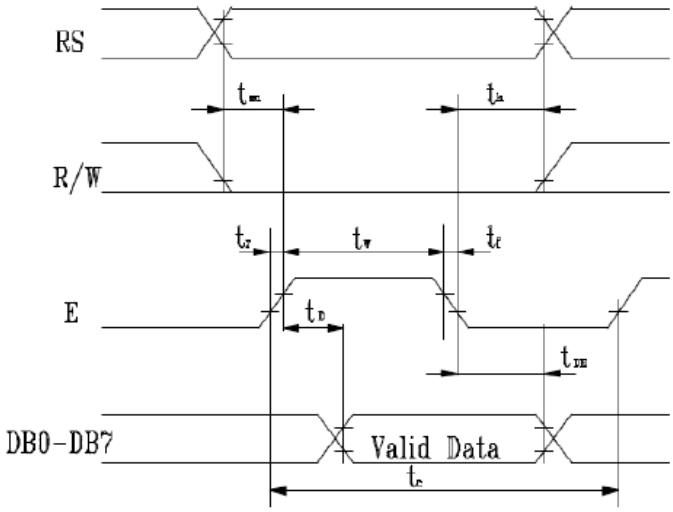
2.4 Màn hình LCD

Mạch Kit được thiết kế để tương thích với màn hình LCD text (LCD 1602). LCD được thiết kế thông thường là 16 chân.

Hình 2.4 Màn hình LCD 1602 sử dụng trong bộ Kit

Các chân và chức năng cụ thể tưng chân như sau:

- Các chân VSS, VDD, VEE lần lượt là các chân: GND, VCC, điều chỉnh độ tương phản. Độ tương phản lớn nhất khi VEE = VSS
- Các chân RS, R/W, E: là các chân chức năng điều khiển trạng thái của LCD (cho phép ghi/ đọc lệnh/ dữ liệu lên LCD). Hoạt động của các chân được tuân theo sơ đồ Timming (Timming diagram)
- Các chân D0-D7: Nhận lênh/ dữ liệu từ VĐK để hiển thị lên màn hình
- Chân A, K: Để bật/ tắt đèn nền của LCD. Thứ tự kết nối là (VCC, GND)



Hình 2.5 Timing diagram của LCD

2.5 Arduino Uno giao tiếp UART với máy tính

Thông thường ta có thể sử dụng một số Module giao tiếp UART với máy tính (ví dụ như FT232RL,…). Tuy nhiên, để thuận tiện với những phần cứng sẵn có, chúng em sẽ dùng board và lập trình thêm trên board Arduino Uno (Sử dụng chip ATmega328p) để đọc dữ liệu UART gửi từ vi điều khiển ATmega16a. Thông qua IC CH340G trên board Uno, dữ liệu UART sẽ được hiển thị trên máy tính để thuận tiện trong quá trình debug.

Hình 2.6 Module Arduino Uno sử dụng chip ATmega328p

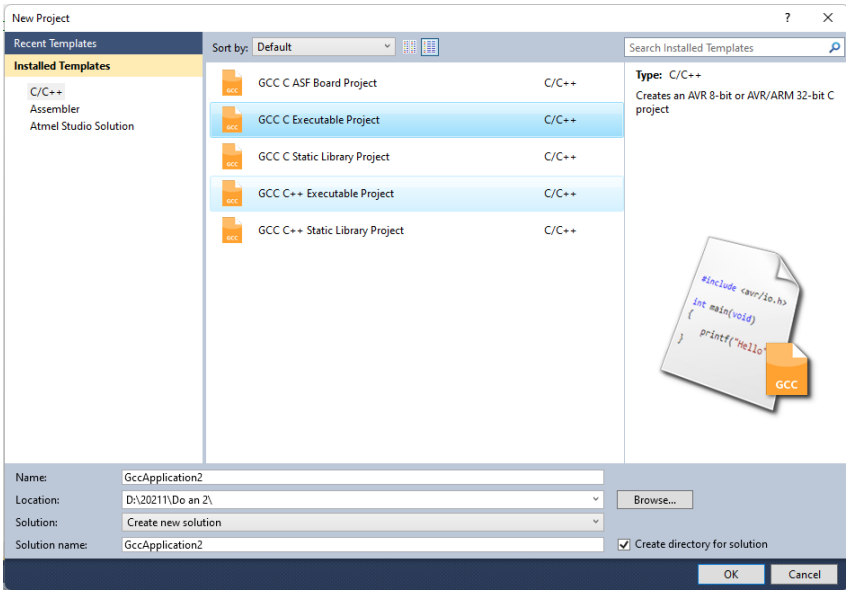
CHƯƠNG 3. THỰC HÀNH LẬP TRÌNH CHO VĐK

3.1 Tạo Project mới với Microchip Studio và nạp thử mã máy cho VĐK

Khi bắt đầu lập trình với VĐK họ AVR có rất nhiều phần mềm hỗ trợ tốt cho dòng VĐK này, tuy nhiên, lí do mà Microchip Studio được sử dụng là mang lại sự hiểu biết về VĐK đang lập trình – vì với Microchip Studio ta phải khai báo cụ thể các chân, v.v. còn một số IDE khác hỗ trợ điều này khi tạo một project mới.

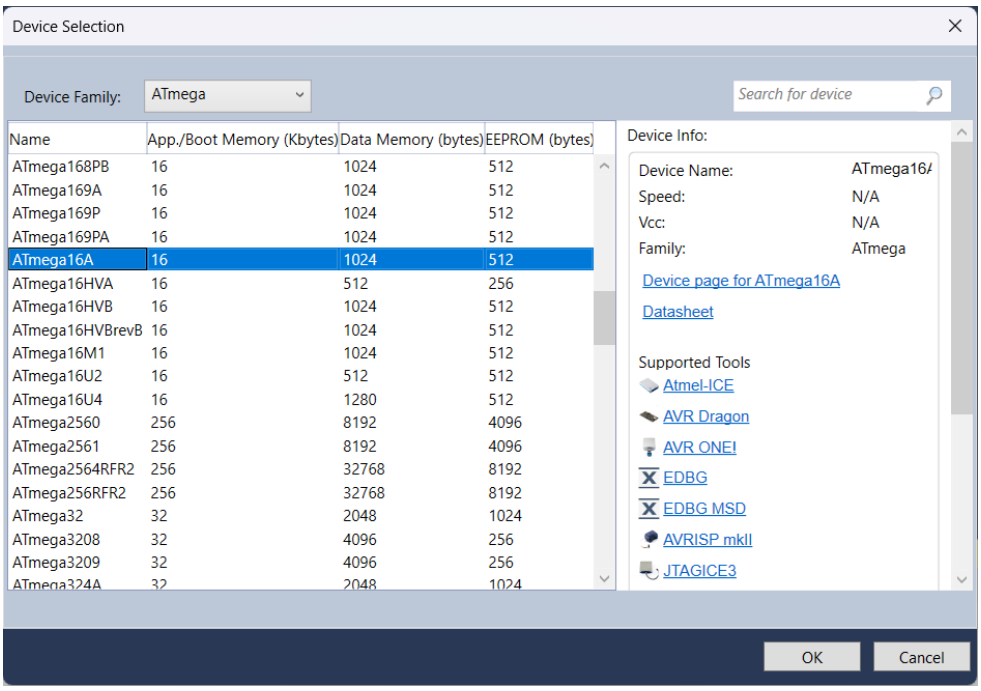
Cụ thể, để tạo một Project mới và nạp thử mã máy cho VĐK ta cần thông qua các bước sau:

Bước 1: Trong Microchip Studio 7.0, vào *File* > *New* > *Project***.** Sau khi thay đổi tên và đường dẫn lưu project thì ta bấm OK.

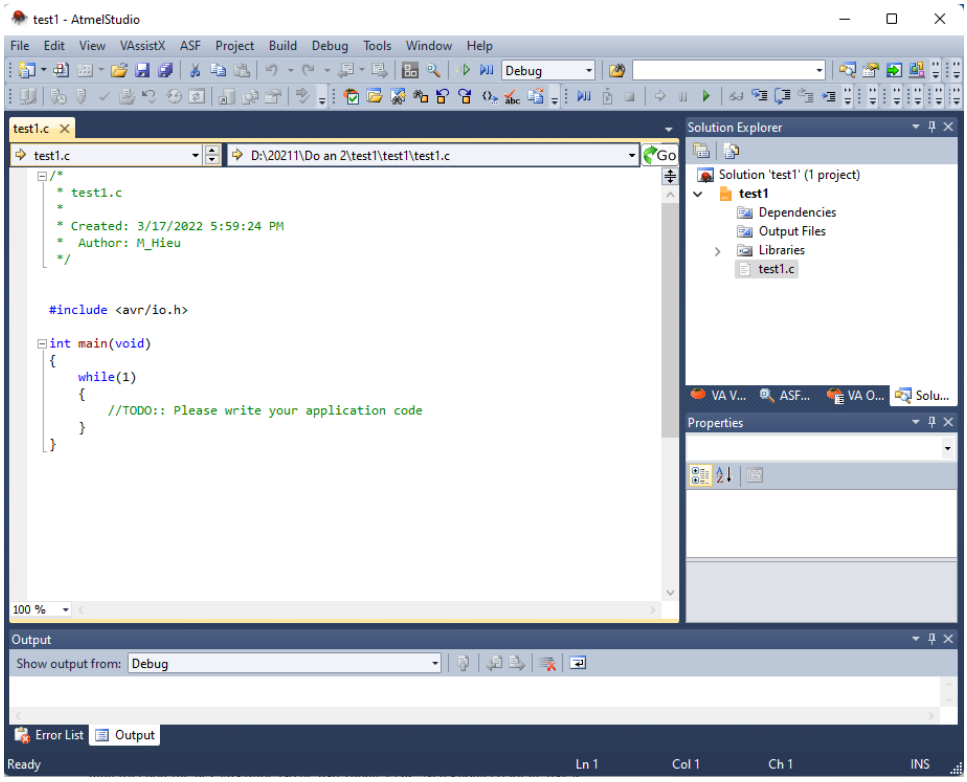


Hình 3.1 Giao diện tạo Project mới trong Microchip Studio 7.0

Bước 2: Sau khi tạo Project, giao diện chọn loại vi điều khiển sẽ hiện ra, ở đây ta sẽ chọn loại vi điều khiển là Atmega 16a.

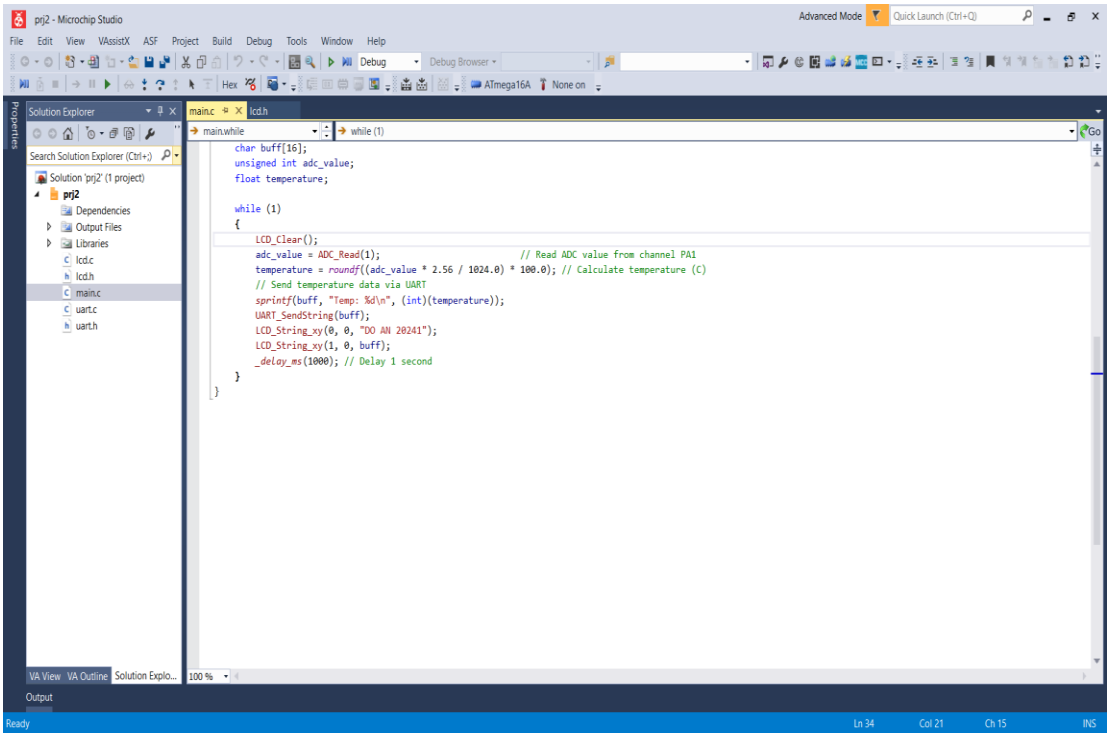


Hình 3.2 Giao diện chọn loại vi điều khiển



Hình 3.3 Giao diện làm việc chính sau khi tạo project

Bước 3: Cuối cùng thì chọn view → Solution Explorer (Ctrl + Alt + L) Microchip Studio 7.0 sẽ hiện ra cấu trúc của project ta đang thực hiện.



Hình 3.4 Cấu trúc của project hiện tại

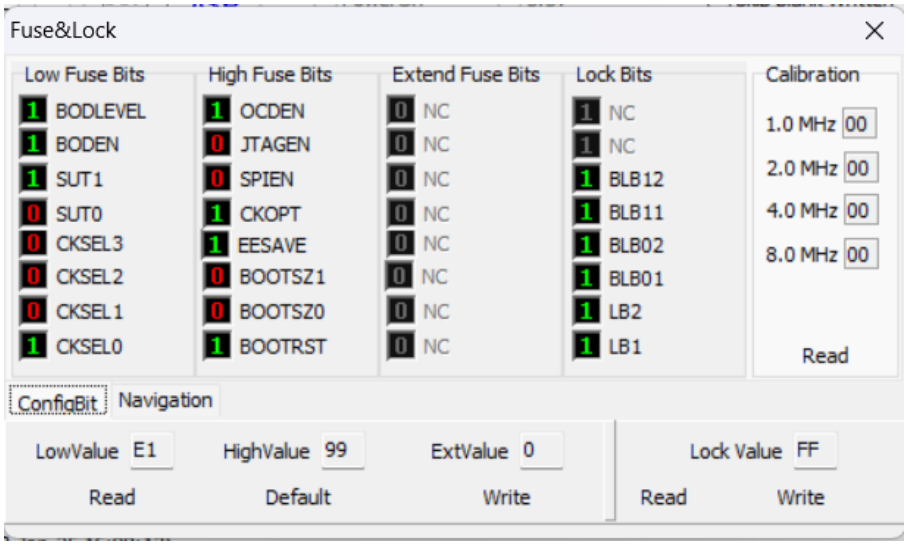
Bước 4 : Copy đoạn mã nguồn vào file *main.c* trong project vừa tạo:

```c
#define F_CPU 8000000UL // CPU Frequency (8 MHz)
#include <avr/io.h>
#include <util/delay.h> // Lib for use delay function

int main(void)
{
    DDRD |= 0xFF;
    uint8_t iter = 0xFF;
    PORTD = iter;
    // DDRC |= 0xFF;

    for (;;)
    {
        iter <<= 1;
        PORTD = iter;
        _delay_ms(50);
        if (!iter)
        {
            iter = 0xFF;
        }
    }

    return 0;
}
```

Đoạn code trên có nghĩa đang xét chế độ cho Port D với DDRx là mode Input hay Output của mỗi chân vi điều khiển. 1 tức là chân đó là chân xuất tín hiệu ra và ngược lại.

Bước 5: Dịch đoạn mã nguồn trên sang mã máy bằng cách chọn *Build→Build Solution (F7)*. Nếu không có lỗi gì, file mã máy *"tên project".hex* sẽ được tạo trong thư mục *Debug* của Project.

Bước 6: Sử dụng phần mềm PROGISP để chỉnh cấu hình Fuse bit cho VĐK như hình rồi chọn file hex vừa được tạo để nạp xuống VĐK. Nếu không có lỗi gì, dãy 8 LED đơn trên Kit sẽ sáng lần lượt từ LED D3 → D10 và lặp lại như vậy vô hạn



Hình 3.5 Thiết lập Fuse bit trong Proisp

Để có thể nạp file hex vào cho vi điều khiển ta cần cắm chân nạp của usbisp vào đê có ghi isp trên bản mạch của mình.

Lưu ý: Fuse bit là những thiết lập ban đầu cho vi điều khiển bao gồm nhiều thiết lập như: chọn nguồn xung clock, chọn nguồn giao tiếp…. Hạn chế thay đổi Fuse&Lock bit vì thiết lập sai có thể đưa VĐK vào chế độ đặc biệt – không thể khôi phục lại.

- Để nạp Fuse Bit, dùng phím tắt Ctrl + U.
- Để xóa dữ liệu trên chip (bước khuyến nghị trước khi nạp chương trình mới), dùng phím tắt Ctrl + E.
- Để nạp firmware vào chip, dùng phím tắt Ctrl + F.

3.2 Ví dụ lập trình điều khiển cổng ra số

Mục tiêu

- Với LED đơn: khi bật nguồn, toàn bộ LED tắt. Sau mỗi 0.5 s, có thêm một LED sáng (từ trái sang phải) để tạo thành dải sáng có độ dài tăng dần. Sau khi dải sáng đạt độ dài cực đại, toàn bộ LED tắt và quy trình được lặp lại từ đầu.

- Với LED 7 thanh: khi bật nguồn, LED 7 thanh hiện số 0. Sau mỗi 0.5 s, số đếm trên LED 7 thanh tăng hêm 1 đơn vị và dấu chấm trên LED đảo trạng thái (nhấp nháy). Sau khi tăng đến 9, số đếm quay lại giá trị 0 và quy trình được lặp lại từ đầu.

Thực hiện

Bước 1: Tạo mới Project AVR_Kit_Test như đã hướng dẫn ở trên. Trong project này ta tạo tiếp 2 file là *AVR_Kit_Test.c* (chưa hàm *main()*) và *thu_vien_rieng.h*

Bước 2: Trong file *thu_vien_rieng.h* ta tạo 4 chương trình con:

- INIT(): là chương trình con dùng để khởi tạo trạng thái cho các chân I/O của VĐK.
- PORT(): là chương trình còn dùng để bật/tắt các LED thong qua việc điều khiển các PORT của VĐK.
- LED7_OUT(num): là chương trình con dùng để điều khiển LED 7 thanh hiển thị số theo biến num ($0 \leq num \leq 9$).
- DELAY_MS(mili_count): là chương trình con dùng để tạo ra các khoảng thời gian trễ, tính bằng mili giây, giữa các lần bật/tắt LED để dễ dàng qua sát hiệu ứng (*Lưu ý: project phải chọn Optimization là -O0 thì hàm này mới hoạt động đúng!*).

Trong file *AVR_Kit_Test.c* ta cần khai báo các thư viện sẽ sử dụng (file *.h) – gồm thư viện chuẩn của AVR (*avr/io.h*) và *thu_vien_rieng.h*, định nghĩa các hằng số, khai báo biến toàn cục và gọi hàm *main()*. Trong hàm *main()* ta sẽ gọi hai chương trình con INIT() và PORT() từ file *thu_vien_rieng.h*.

Bước 3: Tiến hành biên dịch sang mã mãy (Build - F7) và nạp mã máy xuống IC VĐK như đã hướng dẫn ở trên.

Chi tiết các hàm

Hàm INIT(), để sử dụng các chân I/O của VĐK ta cần khai báo trạng thái output của các chân (1 tương ứng với trạng thái output, 0 tương ứng với trạng thái input). Sau khi đã khai báo trạng thái ta cần khai báo mức logic của các chân đó.

```
void INIT()
{
    // Khoi tao trang thai output cho cac chan noi toi cac LED don
    DDRD |= 0xFF;
    // Khoi tao trang thai logic 1 cho cac chan noi toi cac LED don
    PORTD |= 0xFF;
    // Khoi tao trang thai output cho cac chan noi toi cac LED 7 thanh
    DDRC |= 0xFF;
    // Khoi tao trang thai logic 1 cho cac chan noi toi cac LED 7 thanh
    PORTC |= 0xFF;
}
```

Hàm PORT() đây là hàm không có tham số và không trả về giá trị, hàm điều khiển trạng thái logic của các chân trong các Port của VĐK với mục đích cho các LED sáng tắt theo yêu cầu ở trên.

```
void PORT()
{
    /* Khai bao cac bien dung trong ham */
    // Bien led_shift dung de dieu khien cac LED don
    // Gia tri dau la 255 = 0xFF 0b11111111 -> cac LED deu tat
    unsigned char led_shift = 255;
    // Bien dem cho LED 7 thanh, gia tri dau la 0
    unsigned char led_7_count = 0;
    // Vong for giup cac LED sang/tat theo quy luat lap di lap lai
    for (;;)
    {
        /* Doan ma dieu khien cac LED don */
        // Cac LED tat/sang theo 8bit cua bien led_shift
        PORTD = led_shift;
        // Thay doi bien led_shift
        if (led_shift != 0)             // Neuu led_shift khac 0
            led_shift = led_shift << 1; // Dich trai 1bit
        else
            led_shift = 255; // Tro lai gia tri255
        /* Doan ma dieu khien LED 7 thanh */
        // Xuat gia tri dem ra LED 7 thanh
        LED7_OUT((led_7_count));
        // Dao trang thai PC3 de nhap nhay dau cham tren LED 7 thanh
        PORTC ^= (1 << PC3);
        // Tang dan gia tri dem
        led_7_count = led_7_count + 1;
        // Khi vuot qua 9, gia tri dem duoc reset ve 0
        if (led_7_count > 9)
```

```
            led_7_count = 0;
        // Ham tre khoang 0.5s = 500 ms
        DELAY_MS(500);
    }
}
```

Hàm LED7_OUT() có tham số là num và không trả về giá trị, hàm điều khiển LED 7 thanh hiển thị num bằng cách sáng tắt các thanh LED một cách phù hợp.

```
void LED7_OUT(unsigned char num)
{
    // Khai bao bien temp luu trang thai cua PORTC
    unsigned char temp = PORTC;
    // Cac chan vi dieu khien  ung voi cac thanh LED
    //  a - PC5
    //  b - PC4
    //  c - PC2
    //  d - PC1
    //  e - PC0
    //  f - PC6
    //  g - PC7
    // dot - PC3
    // Tat cac doan LED hien dang sang truoc khi sang cac doan LED moi
    temp &= 0B00001000;
    // Gan muc logic cho 8 bit cua bien temp ung voi gia tri cua bien num
    switch (num)
    {
    case 0:
        temp |= 0B10000000;
        break;
    case 1:
        temp |= 0B11100011;
        break;
    case 2:
        temp |= 0B01000100;
        break;
    case 3:
        temp |= 0B01000001;
        break;
    case 4:
        temp |= 0B00100011;
        break;
    case 5:
        temp |= 0B00010001;
        break;
    case 6:
        temp |= 0B00010000;
```

```
        break;
    case 7:
        temp |= 0B11000011;
        break;
    case 8:
        temp |= 0B00000000;
        break;
    case 9:
        temp |= 0B00000001;
        break;
    }
    // Xuat gia tri logic moi ra PORTC de lam sang LED 7 thanh
    PORTC = temp;
}
```

Hàm DELAY_MS(), hàm này nhận mili_count làm tham số và không trả về giá trị, được sử dụng để tạo ra khoảng thời gian trễ tính bằng mili giây. Việc trễ được thực hiện bằng các vòng lặp rỗng, tuy không thực hiện gì nhưng vẫn làm CPU tiêu tốn một khoảng thời gian nhất định cho việc khởi tạo và kết thúc.

```
void DELAY_MS(unsigned int mili_count)
{
    // Khai bao hai bien chay cho 2 vong for
    unsigned int i, j;
    // Xung nhip cua he thong cang cao, so vong lap cang tang
    mili_count = mili_count * FRE;
    // Cac vong for gay tre
    for (i = 0; i < mili_count; ++i)
        for (j = 0; j < 53; ++j)
            ;
}
```

3.3 Ví dụ lập trình đọc trạng thái logic đầu vào số

Mục tiêu

- Với LED đơn: khi bật nguồn, toàn bộ LED tắt. Nếu phím PB1 được ấn, chỉ hai LED ngoài cùng bên trái sáng. Nếu phim PB2 được ấn, chỉ hai LED tiếp theo sáng,… Nếu phim PB4 được ấn, chỉ hai LED ngoài cùng bên phải sáng.
- Với LED 7 thanh: khi bật nguồn, LED 7 thanh hiện số 0. Nếu phim PBx được ấn, LED 7 thanh hiện giá trị của x.

Thực hiện

Bước 1: Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* sau đó chỉnh sửa lại project này.

Bước 2: Trong file *thu_vien_rieng.h* thêm hai chương trình con:

- PB_2_LED(): chương trình con dùng để điều khiển LED theo phím ấn.
- PB_CHECK(): chương trình con dùng để nhận diện phím đang được ấn.

Trong file *AVR_Kit_Test.c* ta khởi tạo một biến toàn cục *unsigned char push_button = 0*, xóa hoặc comment hàm PORT() và gọi hàm PB_2_LED() sau INIT().

Bước 3: Tiến hành biên dịch sang mã mãy và nạp mã máy xuống IC VĐK.

Chi tiết các hàm

Hàm PB_2_LED() là hàm không có tham số và không trả về giá trị, được sử dụng để điều khiển LED theo phím ấn theo quy tắc nhất định đã được mô tả ở trên.

```
void PB_2_LED()
{
    // Vong for giup viec quet phim an duoc lap di lap lai
    for (;;)
    {
        // Goi ham quet phim, luu ket qua phim an vao bien push_button
        push_button = PB_CHECK();
        // Hien so thu tu phim an ra LED 7 thanh
        LED7_OUT(push_button);
        // Dieu khien hang LED don
        switch (push_button)
        {
            // Neu push_button = 1, sang 2 LED ngoai cung ben trai
        case 1:
            PORTD = 0b11111100;
            break;
            // Tuong tu voi, cac gia tri con lai...
        case 2:
            PORTD = 0b11110011;
            break;
        case 3:
            PORTD = 0b11001111;
            break;
        case 4:
            PORTD = 0b00111111;
            break;
            // push_button = 0, tat tat ca cac LED default:
            PORTD = 0xFF;
        }
    }
}
```

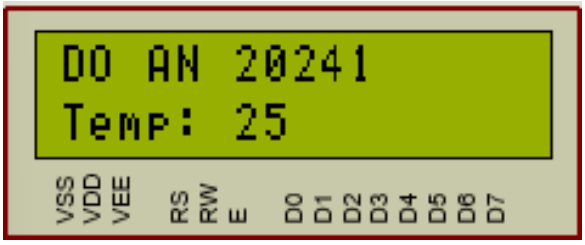
Hàm PB_CHECK() là hàm không có tham số và có trả về giá trị, được sử dụng để nhận diễn các phím ấn. Giá trị trả về của hàm là thứ tự các phím ấn. Khi được ấn, phím sẽ kết nối chân tương ứng của VĐK với GND (mức logic 0). Khi nhả phím, chân tương ứng của VĐK sẽ được treo lên mức 1 nhờ các điện trở kéo có sẵn. Hàm trả về 0 khi không có phím nào được ấn (khi không có phím nào được ấn thì trạng thái LED sẽ reset), trong trường hợp hàm trả về push_button thì trạng thái của các LED sẽ được giữ cho đến khi phím khác được ấn. (push_button là biến toàn cục lưu giá trị trả về của hàm PB_CHECK()).

```
unsigned char PB_CHECK()
{
    // Kiem tra trang thai logic cua 4 chan PB0-3. Neu khac 1111
    if ((PINB & 0x0F) != 0x0F)
    {
        // Kiem tra PB0, neu la muc logic 0, ham ket thuc va tra ve 1
        if (!(PINB & (1 << PB0)))
            return 1;
        // Kiem tra PB1, neu la muc logic 0, ham ket thuc va tra ve 2
        if (!(PINB & (1 << PB1)))
            return 2;
        // Kiem tra PB2, neu la muc logic 0, ham ket thuc va tra ve 3
        if (!(PINB & (1 << PB2)))
            return 3;
        // Kiem tra PB3, neu la muc logic 0, ham ket thuc va tra ve 4
        if (!(PINB & (1 << PB3)))
            return 4;
    }
    // Neu khong co phim nao duoc an, ham ket thuc va tra ve 0
    return 0;
}
```

3.4 Ví dụ lập trình đo điện áp tương tự và hiển thị kết quả lên LCD

Mục tiêu

- Bộ ADC: liên tục số hóa mức điện áp tại đầu vào ADC0, 10 bit kết quả được lưu trong thanh ghi ADC.
- Màn hình LCD: hiển thị dòng text và liên tục cập nhật giá trị ADC theo mẫu:



Hình 3.6 Hiển thị lên màn hình LCD

Thực hiện

Bước 1: Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* sau đó chỉnh sửa lại project này.

Bước 2: Thêm 2 file chứa thư viện ADC và LCD từ thư mục *Library* đã được cung cấp sẵn vào project bằng cách vào *File* > *New* > *Source…* (hoặc ấn tổ hợp phím Ctrl + N) tạo ra 2 file mới sau đó copy lại nội dung 2 file này vào 2 file mới được tạo trên.

Bước 3: Chỉnh sửa và bổ sung các lệnh cần thiết vào thư viện *thu_vien_rieng.h* để khởi tạo ADC, LCD, điều khiển hoạt động. Chỉnh sửa, cập nhật file *AVR_Kit_Test.c*.

Bước 4: Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VĐK.

Chi tiết các hàm

Hàm ADC_2_LCD(), hàm này là hàm không có giá trị trả về và không có tham số, có tác dụng thiết lập các bit khởi tạo cho LCD1602 và sau đó thực hiện việc hiển thị, cập nhập liên tiếp giá trị ADC đọc được từ chân ADC0 (nối tới biến trở).

```
void ADC_2_LCD()
{
    // Khoi tao man hinh LCD
    DDRD |= (1 << PD5);
    PORTD &= ~(1 << PD5);
    PORTC |= 0x0F;
    LCD4_INIT(0, 0);
    // Hien thi cac dong text tinh
    LCD4_CUR_GOTO(1, 0);
    LCD4_OUT_STR("Test ADC & LCD");
    LCD4_CUR_GOTO(2, 0);
    LCD4_OUT_STR("ADC0: 0000/1024");
    // Vong lap do va cap nhat gia tri ADC
    for (;;)
    {
        ADC_STA_CONVERT();
        LCD4_CUR_GOTO(2, 6);
        LCD4_OUT_DEC(ADC, 4);
        DELAY_MS(200);
    }
}
```

Ngoài ra, ta cũng cần phải thêm vào hàm INIT() những câu lệnh khởi tạo cho ADC. Trong đó ADC_AVCC() là câu lệnh để thiết lập điện áp tham chiếu cho ADC là 5 V, ADC_IN(0) là câu lệnh để đọc giá trị ADC từ chân ADC0. Ngoài ADC_AVCC() ta còn có thể dùng ADC_2V56() chuyển điện áp tham chiều từ 5 V xuống 2.56 V.

```
// Khoi tao tran thai input tha noi cho 8 dau vao ADC DDRA = 0x00;
PORTA = 0x00;
// Goi cac ham khoi tao tham so cho bo ADC
ADC_PRES(128);
ADC_AVCC();
ADC_IN(0);
```

3.5 Ví dụ lập trình giao tiếp với máy tính qua chuẩn UART-USB

Mục tiêu

- Bộ UART: hoạt động ở chế độ truyền 8 bit, tốc độ 9600 bps, liên tục gửi đoạn text sau lên máy tính:

  [Họ và tên], [MSSV], [Lớp – Khóa]
  Vien Dien tu – Vien thong, Truong DHBK HN
- Màn hình LCD: hiển thị và dịch dần (sau mỗi 1s) hai dòng chữ trên sang trái để có thể quan sát được oàn bộ đoạn text.

Thực hiện

Bước 1: Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* sau đó chỉnh sửa lại project này.

Bước 2: Thêm 2 file chứa thư viện UART và LCD từ thư *Library* đã được cung cấp sẵn vào project bằng cách vào *File* > *New* > *Source…* (hoặc ấn tổ hợp phím Ctrl + N) tạo ra 2 file mới sau đó copy lại nội dung 2 file này vào 2 file mới được tạo trên.

Bước 3: Chỉnh sửa và bổ sung các lệnh cần thiết vào thư viện *thu_vien_rieng.h* để khởi tạo UART rồi điều khiển hoạt động. Chỉnh sửa, cập nhật file *AVR_Kit_Test.c*.

Bước 4: Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VĐK.

Chi tiết các hàm

Hàm UART() là hàm không có tham số, không có giá trị trả về. Hàm này sẽ thiết lập, khởi tạo UART và LCD, sau đó truyền dữ liệu liên tục thông qua UART.

```
void UART()
{
    // Khoi tao UART voi cac thong so ve braud rate, uart size, parity bit,
stop bit
    UART_INIT(51, 8, 0, 1);
    // Khoi tao LCD
    DDRD |= (1 << PD5);
    PORTD &= ~(1 << PD5);
    PORTC |= 0x0F;
```

```
    LCD4_INIT(0, 0);
    // Hien thi dong text tinh len LCD
    LCD4_CUR_GOTO(1, 0);
    LCD4_OUT_STR("Le Cong Tuan, 20172901, DTVT05 - K62");
    LCD4_CUR_GOTO(2, 0);
    LCD4_OUT_STR("Vien DTVT, Truong DHBK HN");
    // Ham tao tre
    DELAY_MS(1000);
    // Vong for vo han, gui du lieu cho may tinh thong qua UART
    for (;;)
    {
        UART_TRAN_STR("Le Cong Tuan, 20172901, DTVT05 - K62");
        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);
        UART_TRAN_STR("Vien DTVT, Truong DHBK HN");
        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);
        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);
        DELAY_MS(1000);
        // Ham dich toan bo du lieu cua LCD sang trai 1 don vi
        LCD4_DIS_SHIFT(1, 1);
    }
}
```

Hàm LCD4_DIS_SHIFT() là hàm có 2 tham số và không có giá trị trả về, được sử dụng để mở tính năng dịch dữ liệu của LCD thông qua lệnh LCD4_OUT_CMD.

```
void LCD4_DIS_SHIFT(unsigned char lcd4_direct, unsigned char lcd4_step)
{
    unsigned char i;
    if (lcd4_direct == 0)
        // Kiem tra lcd4_direct, bang 0 thi dich phai, 1 thi dich trai
        // Vong for de dich du lieu theo step
        for (i = 0; i < lcd4_step; i++)
            LCD4_OUT_CMD(0x1C);
    else
        // Vong for de dich du lieu theo step
        for (i = 0; i < lcd4_step; i++)
            LCD4_OUT_CMD(0x18);
}
```

CHƯƠNG 4. VẬN DỤNG CÁC KIẾN THỨC VÀO THỰC TẾ

4.1 Mục tiêu

Sau quá trình làm quen công cụ thiết kế mạch điện và thực hành lập trình phần cứng với mã nguồn mẫu, chúng em sẽ vận dụng các kiến thức đã học để thiết kế và hoàn thiện một yêu cầu được đặt ra, bao gồm:

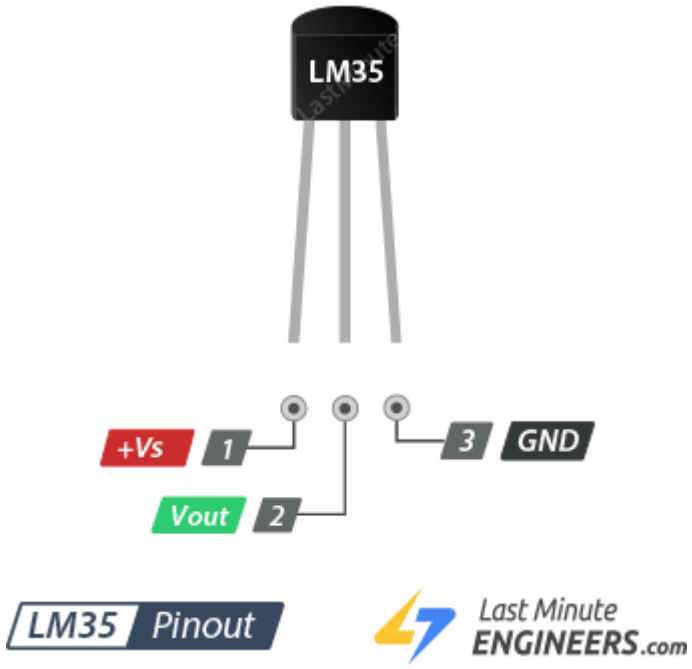
- Thiết kế mạch đo nhiệt độ sử dụng cảm biến tương tự
- Thiết kế mạch điều khiển động cơ DC công suất nhỏ

Trong học phần này, chúng em sẽ chọn đề tài 1 "***Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự***" làm đề tài trong phần vận dụng kiến thức vào thực tế.

Mục tiêu: Chế tạo một mạch điện có khả năng đo nhiệt độ trong dải 0-100 ℃ sử dụng các cảm biến tương tự (LM35). Kết quả đo được hiển thị bằng màn hình LCD với độ phân giải tối thiểu là 0.25 ℃.

4.2 Cảm biến nhiệt độ LM35

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 4.1 Cảm biến nhiệt độ LM35

Một số thông số kĩ thuật của IC cảm biến này:

- Điện áp hoạt động: 4 – 20 V
- Công suất tiêu thụ: 60 µA
- Khoảng đo nhiệt độ: -55 °C đến 150 °C
- Nhiệt độ thay đổi tuyến tính: 10 mV/°C
- Độ chính xác thực tế: ±1/4 °C ở nhiệt độ phòng và trung bình ±3/4 °C cho toàn bộ dải từ -55 °C đến 150 °C

4.3 Phương hướng giải quyết

Như đã tìm hiểu ở trên, ADC của VĐK Atmega16 có độ phân giải 10-bit, tương đương với 1024 mức điện áp từ 0 – 5 V. Vậy mỗi mức tương đương với:

$$\frac{5}{1024} = 0{,}00488$$

Theo mục tiêu đã đề ra, độ phân giải hiển thị phải tối thiểu là 0.25 °C, tương ứng với 2.5 mV. Do đó, ADC của VĐK phải đọc được sự thay đổi là 2.5 mV của IC LM35. Ngoài ra, ta chỉ cần do nhiệt độ từ 0 – 100 °C nên ta không cần quan tâm đến điện áp âm của LM35. Với phương hướng như vậy, ta có 2 giải pháp:

- Hạ mức điện áp tham chiếu của VĐK xuống 2.56 V → Mỗi mức của ADC tương ứng với: $\frac{2.56}{1024} = 2.5mV$
- Sử dụng mạch khuếch đại, có hệ số khuếch đại K = $\frac{4.88}{2.5} = 1.95$

Giải pháp: Hạ mức điện áp tham chiếu của VĐK xuống 2.56V

Với giải pháp này khi ta giam mức điện áp tham chiếu cho bộ ADC của VĐK xuống 2.56 V thì lúc này mỗi mức của ADC tương ứng với chính xác là 2.5 mV. Vậy ta sẽ không cần phải khuếch đại mà thay vào đó ta có thể sử dụng luôn mức điện áp này. Để đổi mức điện áp này sang nhiệt độ ta chỉ cần lấy mức ADC đọc được và nhân với 0.25.

4.4 Các chương trình được xây dựng

4.4.1 Hàm đọc dữ liệu từ LM35

```
// Initialize ADC
void ADC_Init()
{
    ADMUX = (1 << REFS1) | (1 << REFS0);     // Select Vref = 2.56V internal
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1); // Enable ADC,
Prescaler = 64
}
```

```
// Read ADC value
unsigned int ADC_Read(unsigned char channel)
{
    ADMUX = (ADMUX & 0xF8) | (channel & 0x07); // Select ADC channel
    ADCSRA |= (1 << ADSC);                     // Start conversion
    while (ADCSRA & (1 << ADSC))
        ; // Wait for conversion to complete
    return ADC;
}
```

Ở đây ta dùng tín hiệu vào từ cổng PA1 để lấy tín hiệu Analog từ LM35 và đưa qua bộ ADC của ATmega16a để chuyển thành các mức điện áp Digital

4.4.2 Chương trình gửi dữ liệu điều khiển đến LCD

```
#include "lcd.h"

void LCD_Command(unsigned char cmnd) {
    LCD_PORT = cmnd;             // Send command to LCD port
    LCD_CONTROL_PORT &= ~(1 << RS); // RS = 0, command mode
    LCD_CONTROL_PORT &= ~(1 << RW); // RW = 0, write mode
    LCD_CONTROL_PORT |= (1 << EN);  // Enable pulse
    _delay_us(1);
    LCD_CONTROL_PORT &= ~(1 << EN);
    _delay_ms(2);                // Wait for command execution
}

void LCD_Char(unsigned char data) {
    LCD_PORT = data;             // Send data to LCD port
    LCD_CONTROL_PORT |= (1 << RS);  // RS = 1, data mode
    LCD_CONTROL_PORT &= ~(1 << RW); // RW = 0, write mode
    LCD_CONTROL_PORT |= (1 << EN);  // Enable pulse
    _delay_us(1);
    LCD_CONTROL_PORT &= ~(1 << EN);
    _delay_ms(2);                // Wait for data execution
}

void LCD_Init() {
    LCD_DDR = 0xFF;              // Set LCD data port as output
    LCD_CONTROL_DDR |= (1 << RS) | (1 << RW) | (1 << EN); // Set control
pins as output
    _delay_ms(20);               // Wait for LCD to power up

    // Initialize LCD in 8-bit mode
    LCD_Command(0x38);           // Function set: 8-bit mode, 2 lines, 5x8
font
    LCD_Command(0x0C);           // Display ON, cursor OFF
    LCD_Command(0x01);           // Clear display
    _delay_ms(2);
```

```
    LCD_Command(0x06);          // Entry mode set: Increment cursor, no
shift
}

void LCD_String(char *str) {
    while (*str) {
        LCD_Char(*str++);       // Send each character of string to LCD
    }
}

void LCD_Clear() {
    LCD_Command(0x01);          // Clear display
    _delay_ms(2);
}

void LCD_String_xy(char row, char pos, char *str) {
    if (row == 0) {
        LCD_Command((pos & 0x0F) | 0x80); // Command for row 0
        } else if (row == 1) {
        LCD_Command((pos & 0x0F) | 0xC0); // Command for row 1
    }
    LCD_String(str);             // Send string to specified position
}

void LCD_Set_Cursor(char row, char col){
    if (row == 0){
        col |= 0x80;
    }
    else if (row == 1){
        col |= 0xC0;
    }
    LCD_Command(col);
}
```

4.4.3 Chương trình gửi dữ liệu qua UART

```
#include "uart.h"
void UART_Init(unsigned int baud)
{
    unsigned int ubrr = F_CPU / 16 / baud - 1;
    UBRRH = (unsigned char)(ubrr >> 8);  // Upper byte of UBRR
    UBRRL = (unsigned char)ubrr;         // Lower byte of UBRR
    UCSRB = (1 << RXEN) | (1 << TXEN);   // Enable TX and RX
    UCSRC = (1 << URSEL) | (3 << UCSZ0); // Configure 8-bit data
}

// Transmit a single character via UART
void UART_Transmit(unsigned char data)
{
    while (!(UCSRA & (1 << UDRE)))
```

```
        ;       // Wait until buffer is empty
    UDR = data; // Send the data
}

// Send a string via UART
void UART_SendString(const char *str)
{
    while (*str)
    {
        UART_Transmit(*str++);
    }
}
```

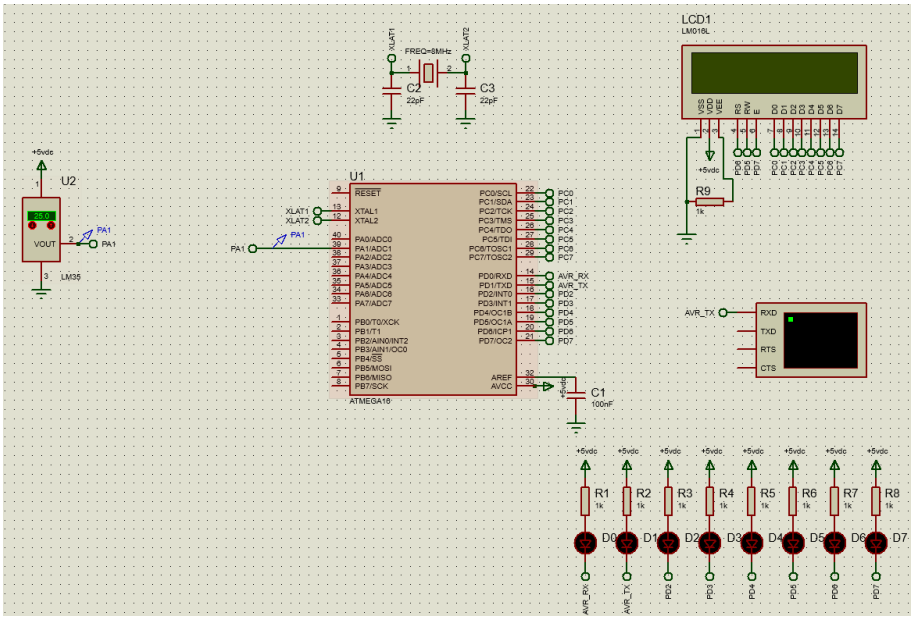
4.4.4 Hàm main

```
#include "uart.h"
#include "lcd.h"
#include "adc.h"

int main(void)
{
    UART_Init(9600); // Initialize UART with baud rate 9600
    ADC_Init();      // Initialize ADC
    LCD_Init();     // Init LCD

    char buff[16];
    unsigned int adc_value;
    float temperature;

    while (1)
    {
        LCD_Clear();
        adc_value = ADC_Read(1);                            // Read ADC value
from channel PA1
        temperature = roundf((adc_value * 2.56 / 1024.0) * 100.0); //
Calculate temperature (C)
        // Send temperature data via UART
        sprintf(buff, "Temp: %d\n", (int)(temperature));
        UART_SendString(buff);
        LCD_String_xy(0, 0, "DO AN 20241");
        LCD_String_xy(1, 0, buff);
        _delay_ms(1000); // Delay 1 second
    }
}
```

4.5 Mô phỏng và thiết kế

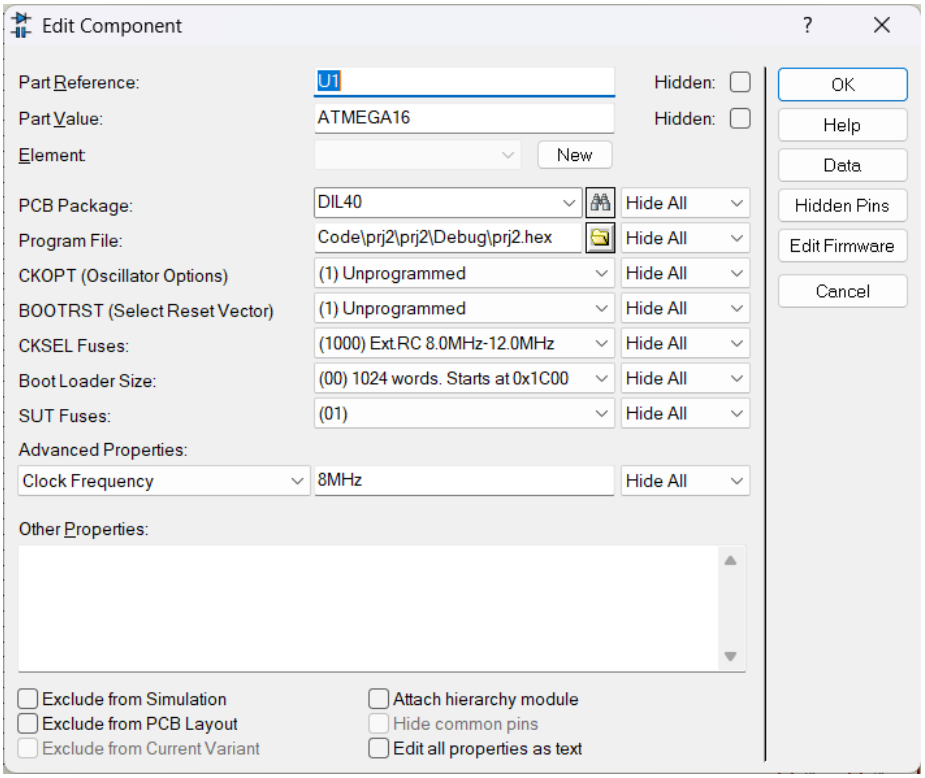
4.5.1 Mô phỏng trên phần mềm proteus 8

Sau khi đã thay đổi code, ta tiến hành mô phỏng trên phần mềm Proteus. Kết quả thu được đạt được yêu cầu đề ra và có độ chính xác cao.

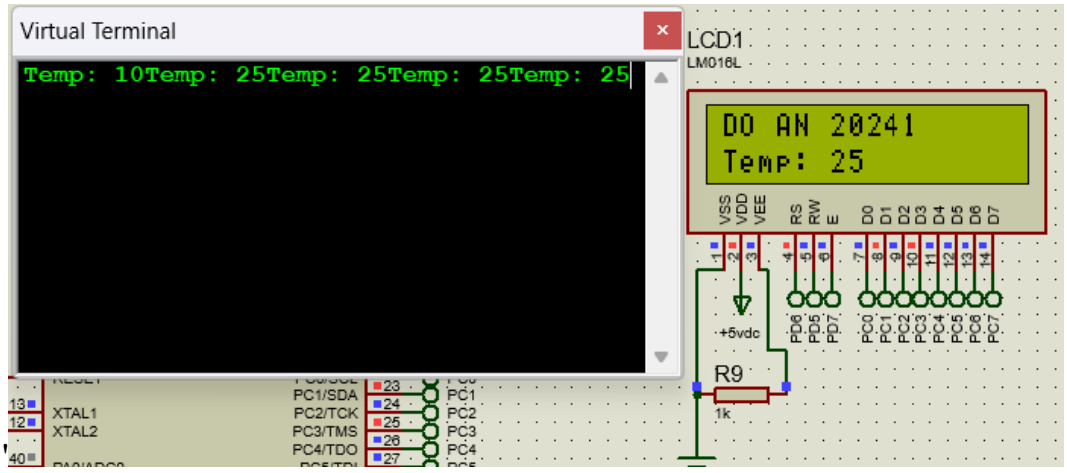


Hình 4.2 Mô phỏng mạch đo nhiệt độ trực tiếp từ LM35 trên Proteus

Lưu ý: Tiến hành biên dịch sang mã máy và mô phỏng bằng cách, click chuột vào VĐK trong Proteus, trong bảng hiện ra tại *Program file* chọn đường dẫn đến file .hex vừa được dịch. Sau đó bấm OK và *Debug* > *Run Simulation*.

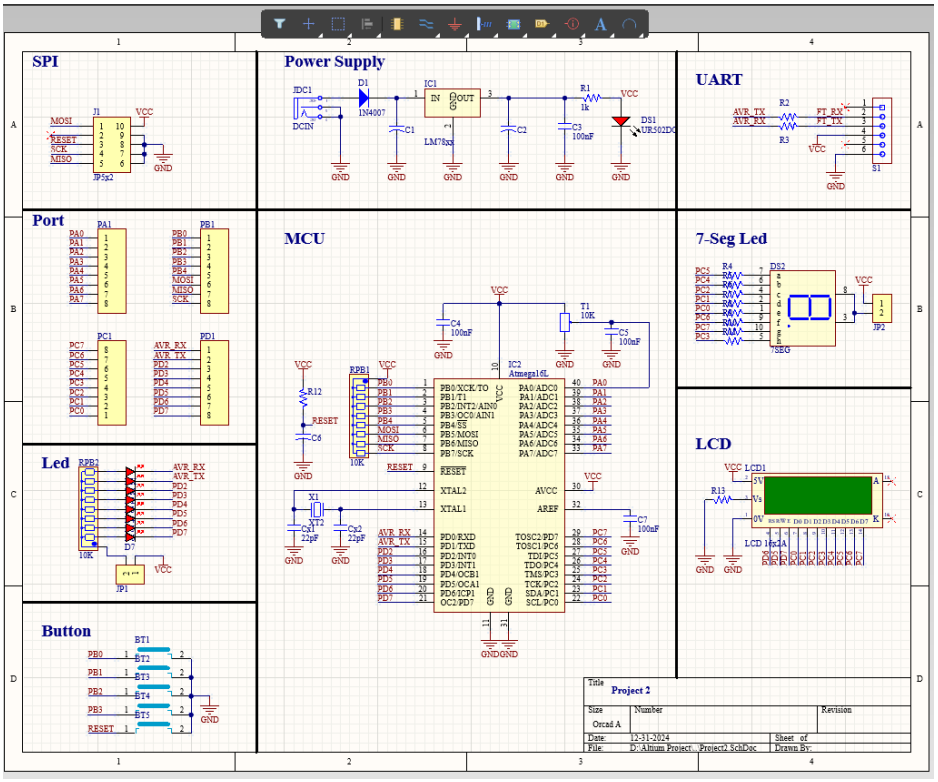


Hình 4.3 Thiết lập Fuse bit, xung clock và file hex trong Proteus

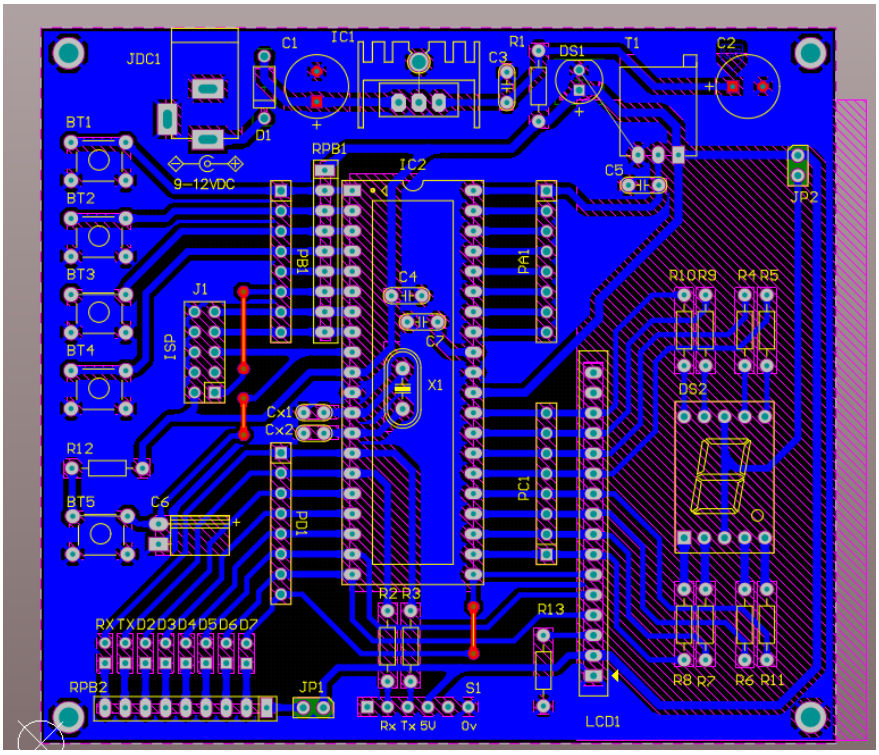


Hình 4.4 Kết quả mô phỏng trên Proteus

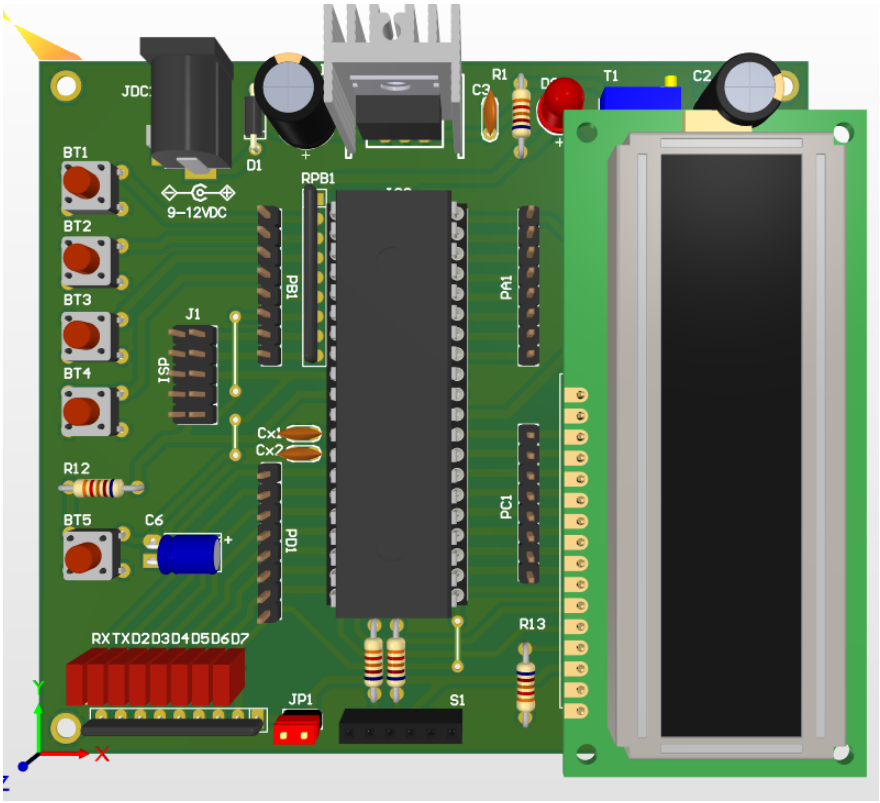
4.5.2 Thiết kế sơ đồ mạch Kit AVR trên phần mềm Altium



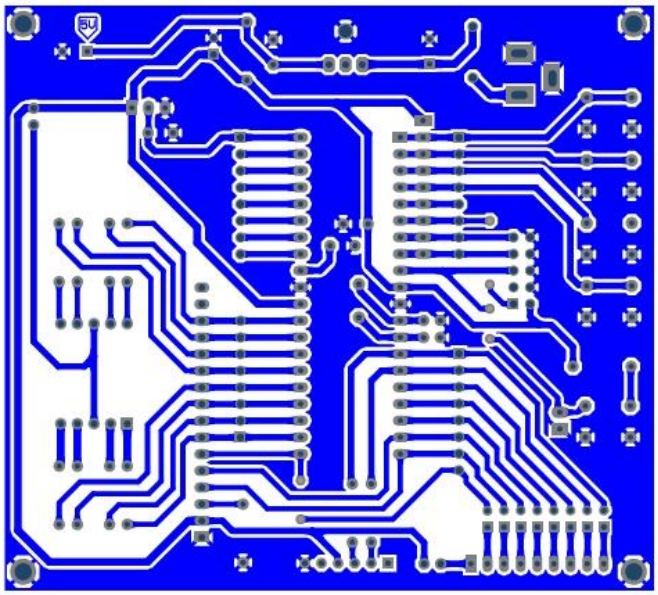
Hình 4.5 Sơ đồ nguyên lý trên Altium 24



Hình 4.6 Mạch PCB trên Altium



Hình 4.7 Mô hình 3D trên Altium



Hình 4.8 Mạch in PCB

4.6 Nhận xét

Độ chính xác rất cao, mạch đơn giản hơn do không cần phải sử dụng khuếch đại mà dùng luôn tính năng của VĐK.

KẾT LUẬN

Qua quá trình tìm hiểu, nghiên cứu và thực hiện đề tài "*Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự* ", chúng em nhận thấy các kết quả khảo sát so sánh thu được thông qua phần mềm mô phỏng và thực nghiệm là hoàn toàn phù hợp với các lý thuyết đã được tính toán trước đó. Trong quá trình thực hiện chúng 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 họ AVR cụ thể là Atmega 16, cũng như nắm chắc hơn về cách xây dựng các mạch điện tử thông qua các phần mềm mô phỏng như Proteus, hay các phần mềm thiết kế mạch in như Altium.

Qua đồ án này, chúng 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 chúng em có nền tảng vững chắc hơn trong quá trình học tập và nghiên cứu sau này.

Một lần nữa chúng em xin chân thành cảm ơn thầy Dương Thanh Phương 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] Takashi, “AVR GCC Tutorial (WinAVR)”

[2] LM35 Datasheet(PDF) - Tiger Electronic Co.,Ltd (alldatasheet.com)

[3] Atmel-8154-8-bit-ATmega16A_Datasheet

[4] Atmel-8154S-8-bit-AVR-ATmega16A_Datasheet Summary

[5] 16x2 LCD Display Module Pinout, Features, Description & Datasheet (components101.com)