**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG ĐIỆN- ĐIỆN TỬ**

Ảnh có chứa văn bản, áp phích, Phông chữ, Đồ họa

Mô tả được tạo tự động

**BÁO CÁO ĐỒ ÁN II**

**ĐỀ TÀI: THIẾT KẾ VÀ CHẾ TẠO MẠCH ĐO NHIỆT ĐỘ SỬ DỤNG CẢM BIẾN DS18B20**

Giảng viên hướng dẫn: Vũ Hồng Vinh

Mã lớp học: 748494

Nhóm: 6

Sinh viên thực hiện:

|  |  |
| --- | --- |
| Uông Nhật Phong: | 20214040 |
| Bùi Nhật Minh: | 20213996 |
| Trương Minh: | 20210577 |

Hà Nội, 1/2025

# LỜI NÓI ĐẦU

Việt Nam đang từng ngày vươn lên mạnh mẽ, khẳng định vị thế của mình trên thế giới. Trong thời đại cách mạng công nghiệp 4.0 - một bước ngoặt lớn thay đổi cục diện đất nước - chúng ta, thế hệ trẻ Việt Nam, có cơ hội tiếp cận và làm chủ những thành tựu vĩ đại của nhân loại, đặc biệt trong các lĩnh vực khoa học kỹ thuật nói chung và ngành Điện tử - Viễn thông nói riêng.

Tuy nhiên, nếu không không ngừng nỗ lực học hỏi và đổi mới bản thân, chúng ta rất dễ tụt lại phía sau trong cuộc đua toàn cầu hóa khốc liệt này. Thấu hiểu điều đó, Trường Đại học Bách khoa Hà Nội đã sớm đề ra những chương trình đào tạo bài bản, từ cơ bản đến chuyên sâu, nhằm trang bị cho sinh viên nền tảng kiến thức vững chắc. Đặc biệt, Viện Điện tử - Viễn thông luôn chú trọng tổ chức các dự án thực hành như Đồ án II, giúp sinh viên tích lũy kinh nghiệm thực tiễn, sẵn sàng đáp ứng yêu cầu khắt khe của thị trường lao động.

Với sự phát triển vượt bậc của khoa học kỹ thuật, ngành Điện tử - Viễn thông đã có những ứng dụng rộng rãi trong công nghiệp và đời sống. Trong lĩnh vực điều khiển, sự ra đời của vi mạch lập trình đã mang lại những giải pháp hiện đại, vượt trội so với các mạch điều khiển truyền thống sử dụng linh kiện rời. Các mạch này không chỉ nhỏ gọn, tiết kiệm chi phí, mà còn đảm bảo độ tin cậy cao và tiêu thụ năng lượng thấp. Nhờ đó, công nghệ điều khiển đã trở thành trái tim của nhiều thiết bị gia dụng hiện đại như máy giặt, đồng hồ điện tử, tivi..., góp phần nâng cao chất lượng cuộc sống của con người.

Từ cảm hứng đó, nhóm chúng em đã lựa chọn đề tài “Thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến DS18B20” cho Đồ án II. Đây không chỉ là cơ hội để áp dụng kiến thức đã học vào thực tế mà còn giúp chúng em hiểu rõ hơn về khả năng ứng dụng của công nghệ trong đời sống.

Trong quá trình thực hiện, chúng em đã nỗ lực hết mình để hoàn thiện đề tài một cách tốt nhất. Tuy nhiên, do giới hạn về kiến thức và kinh nghiệm, chắc chắn không thể tránh khỏi những thiếu sót. Rất mong nhận được những ý kiến đóng góp quý báu từ thầy để chúng em có thể cải thiện và hoàn thiện đề tài này.

Cuối cùng, chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy Vũ Hồng Vinh – người đã tận tình hướng dẫn, hỗ trợ và tạo điều kiện tốt nhất để nhóm chúng em hoàn thành đồ án.

**Mục lục**

[LỜI NÓI ĐẦU 2](#_Toc139927632)

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT CHUNG 4](#_Toc139927633)

[1.1 Giới thiệu chung 4](#_Toc139927634)

[1.2 Vi điều khiển AVR và Mạch Kit phát triển họ AVR 4](#_Toc139927635)

[1.2.1 Vi điều khiển AVR 4](#_Toc139927636)

[1.2.2 Kit phát triển họ AVR 7](#_Toc139927637)

[1.3 Ngôn ngữ lập trình và phần mềm 9](#_Toc139927638)

[CHƯƠNG 2. CHI TIẾT CẤU HÌNH CỦA KIT PHÁT TRIỂN 9](#_Toc139927639)

[2.1 Cấu trúc của mạch Kit 9](#_Toc139927640)

[2.2 Các thông số chính của Kit 13](#_Toc139927641)

[2.3 Mạch nạp mã nguồn 13](#_Toc139927642)

[2.4 Màn hình LCD 1602 và module DS18B20 14](#_Toc139927643)

[2.4.1 Màn hình LCD 1602 14](#_Toc139927644)

[2.4.2 Module DS18B20 15](#_Toc139927645)

[CHƯƠNG 3. THIẾT KẾ MẠCH ĐO NHIỆT ĐỘ 17](#_Toc139927646)

[3.1 Phần mềm biên dịch Atmel studio và nạp code Progisp 17](#_Toc139927647)

[3.1.1 Phần mềm biên dịch Atmel studio 17](#_Toc139927648)

[3.1.2 Phần mềm nạp code Progisp 1.72 19](#_Toc139927649)

[3.2 Mô phỏng và thiết kế 20](#_Toc139927650)

[3.2.1 Mô phỏng trên phần mềm proteus 8.13 20](#_Toc139927651)

[3.2.2 Thiết kế sơ đồ mạch Kit AVR trên phần mềm Altium 22](#_Toc139927652)

[3.3 Mạch thực tế 24](#_Toc139927653)

[3.4 Nhận xét 25](#_Toc139927654)

[KẾT LUẬN 25](#_Toc139927655)

[TÀI LIỆU THAM KHẢO 26](#_Toc139927656)

# CƠ SỞ LÝ THUYẾT CHUNG

## Giới thiệu chung

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, 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.

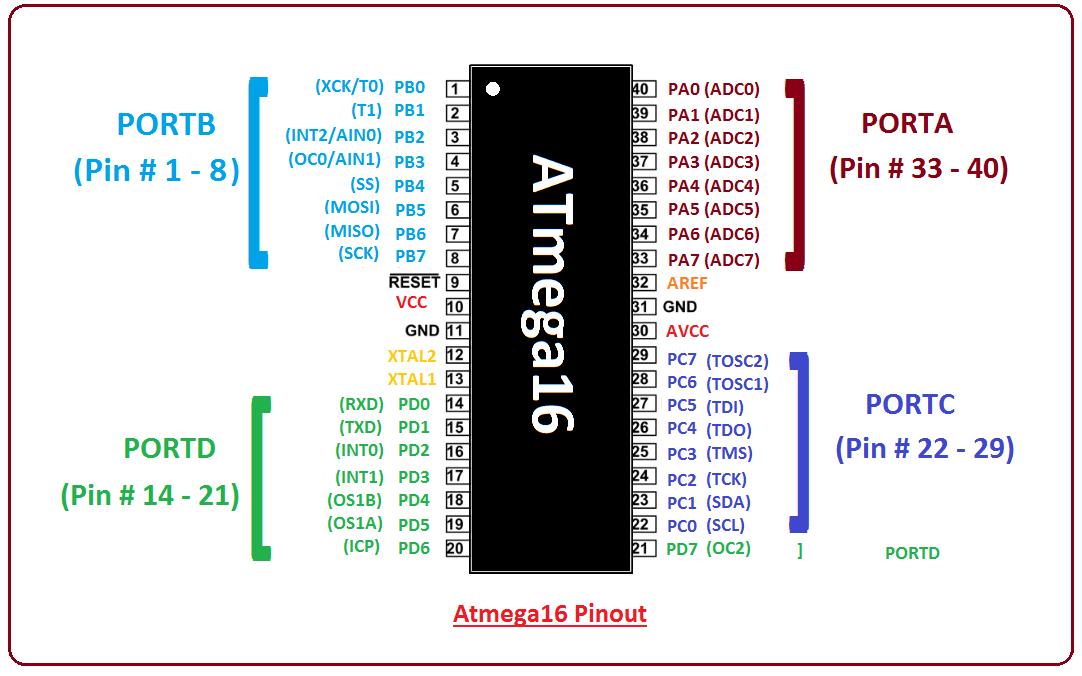
Sau khi tiếp cận nhanh với vi điều khiển (VĐK) thông qua việc xây dựng một số ứng dụng trên VĐK họ AVR cụ thể trong bài báo cáo này em dùng Atmega16L. Em sẽ tiến hành giải quyết một vấn đề cụ thể - thiết kế mạch đo nhiệt độ và hiển thị trên LCD sử dụng cảm biến cảm biến DS18B20 một cảm biến nhiệt độ số được ứng dụng phổ biến trong việc cảm biến nhiệt độ với độ chính xác cao.

## Vi điều khiển AVR và Mạch Kit phát triển họ AVR

### 1.2.1 Vi điều khiển AVR

AVR là một VĐK 8 bit khá mạnh và thông dụng tại thị trường Việt Nam được sản xuất bời hãng Atmel. 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. Atmega16 là một bộ vi điều khiển 8 bit dựa trên kiến trúc RISC bộ nhớ chương trình 16KB ISP flash có thể ghi xóa hàng nghìn lần, 512B EEPROM, một bộ nhớ RAM vô cùng lớn trong thế giới vi xử lý 8 bit (1KB SRAM).

Với 32 chân có thể sử dụng cho các kết nối vào hoặc ra i/O, 32 thanh ghi, 3 bộ timer/counter có thể lập trình, có các gắt nội và ngoại (2 lệnh trên một vector ngắt), giao thức truyền thông nối tiếp USART, SPI, I2C. Ngoài ra có thể sử dụng bộ biến đổi số tương tự 10 bít (ADC/DAC) mở rộng tới 8 kênh, khả năng lập trình được watchdog timer, hoạt động với 5 chế độ nguồn, có thể sử dụng tới 6 kênh điều chế độ rộng xung (PWM), hỗ trợ bootloader.



**Hình 1.1 Sơ đồ chân Atmega16**

* **Mô tả chân của Atmega16**

Atmega16 có 40 chân, mỗi chân được sử dụng để thực hiện một nhiệm vụ cụ thể, có tổng cộng 32 chân I / O và bốn cổng, mỗi cổng bao gồm 8 chân I / O.

PORTA = 8 chân (Chân 33-40)

PORTB = 8 chân (Chân 1-8)

PORTC = 8 chân (Chân 22-29)

PORTD = 8 chân (Chân 14-21)

* **Sau đây là các chức năng chính liên quan đến các chân.**
* PORTA: Các chân từ 33 đến 40 thuộc PORTA. Nó hoạt động giống như đầu vào analog cho bộ chuyển đổi A / D. Tuy nhiên, trong trường hợp không có bộ chuyển đổi A / D, PORTA được sử dụng làm cổng I / O hai chiều 8 bit. Nó đi kèm với điện trở kéo bên trong.

* PORTB: Các chân từ 1 đến 8 thuộc về PORTB. Đây là các chân hai chiều I / O. Cổng này cũng bao gồm các điện trở kéo lên bên trong.

* PORTC: PORTC là cổng I / O hai chiều bao gồm 8 chân. Chân từ 22 đến 29 thuộc về cổng này, tương tự như các cổng khác, nó đi kèm với điện trở kéo bên trong.

* PORTD: Chân từ 14 đến 21 thuộc về cổng này. Đây là cổng hai chiều trong đó mỗi chân có thể được sử dụng làm chân đầu vào hoặc đầu ra. Tuy nhiên, có các tính năng bổ sung liên quan đến cổng này như ngắt, giao tiếp nối tiếp, bộ hẹn giờ và PWM.

* Reset: Chân 9 là chân reset mức thấp đang hoạt động. Xung mức thấp dài hơn độ dài xung tối thiểu sẽ tạo ra reset. Các xung ngắn không có khả năng tạo ra reset.

* VCC: Chân 10 là chân cấp nguồn cho bộ điều khiển này. Nguồn điện của cần phải có 5 V để đặt bộ điều khiển này trong điều kiện đang chạy.

* GND: Chân 11 là chân nối đất.

* AREF: Chân 32 là chân tham chiếu tương tự chủ yếu được sử dụng cho bộ chuyển đổi A / D .

* AVCC: Chân 30 là AVCC là chân điện áp cung cấp cho PORTA và ADC. Nó được kết nối với VCC thông qua bộ lọc thông thấp khi có ADC. Tuy nhiên, trong trường hợp không có ADC, AVCC được kết nối bên ngoài với VCC.

* Chân 12 & 13: Một bộ dao động tinh thể được kết nối với các chân này. Atmega16 hoạt động ở tần số bên trong 1MHZ; bộ dao động được thêm vào để tạo ra xung clock và tần số cao.

* **Các ứng dụng**

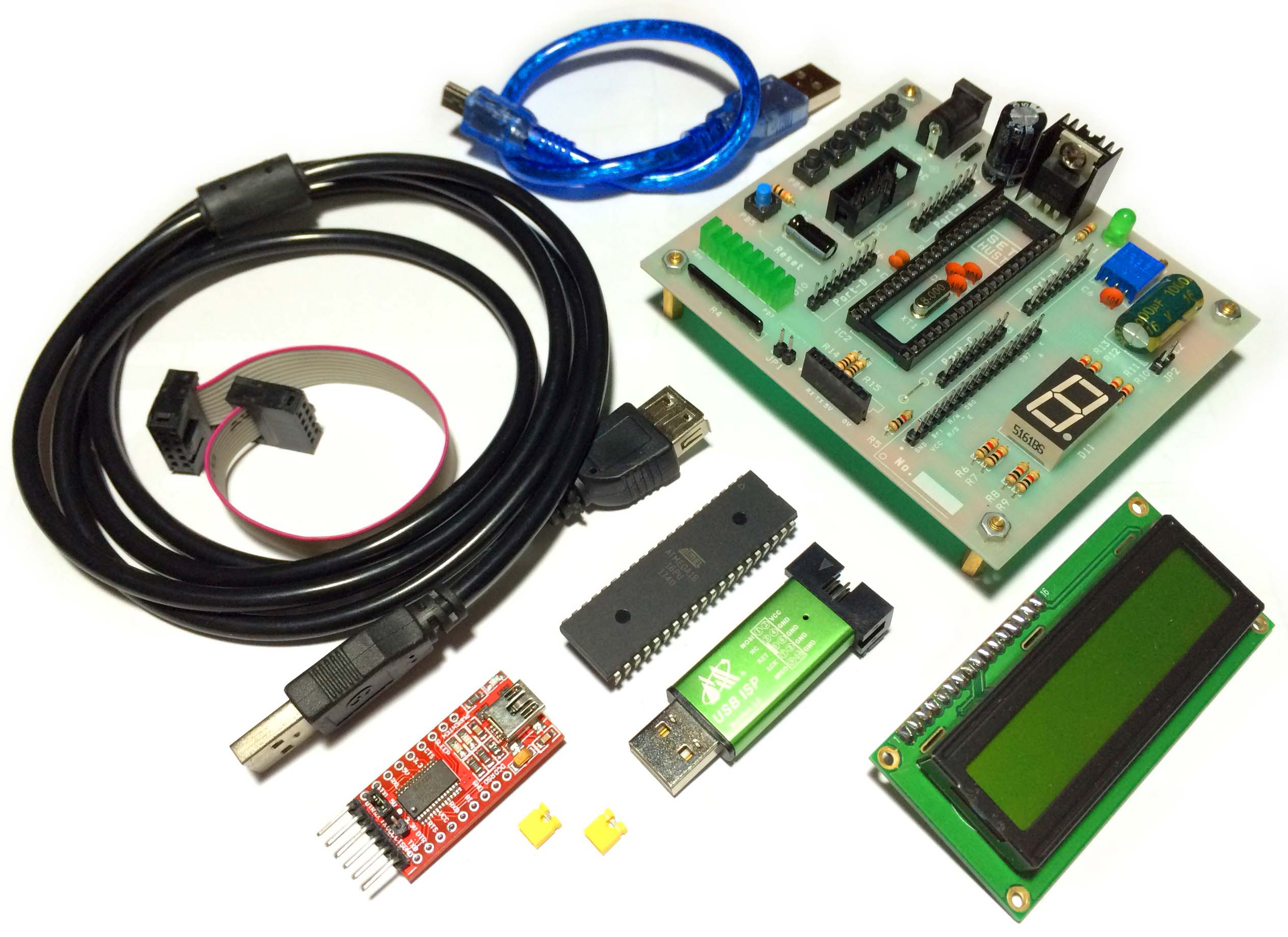
Bộ điều khiển AVR đi kèm với một loạt các ứng dụng cần tự động hóa. Sau đây là các ứng dụng chính của Atmega16.

* Thiết bị y tế
* Tự động hóa nhà
* Những hệ thống nhúng
* Project Arduino
* Được sử dụng trong ô tô và tự động hóa công nghiệp
* Thiết bị gia dụng và hệ thống an ninh
* Thiết bị kiểm soát nhiệt độ và áp suất

### Kit phát triển họ AVR

Với Kit phát triển sử dụng trong học phần Đồ án II 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 hính 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 . Mạch Kit phát triển và các phụ kiện**

**Hình 1.2 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.

## 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, AtmelStudio, WinAVR, codevisionAVR…

Trong nội dung về học phần Đồ án II, em sẽ sử dụng ngôn ngữ lập trình C để lập trình cho VĐK. Môi trường để soạn thảo và biên dịch là phần mềm AtmelStudio 6.2. Phần mềm nạp mã máy là PROGISP (phiên bản 1.72). Ngoài ra, trong phần vận dụng em có sử dụng thêm phần mềm Proteus 8.13 để mô phỏng mạch và phần mềm Altium Designer 19 để thiết kế mạch.

# CHƯƠNG 2. CHI TIẾT CẤU HÌNH CỦA KIT PHÁT TRIỂN

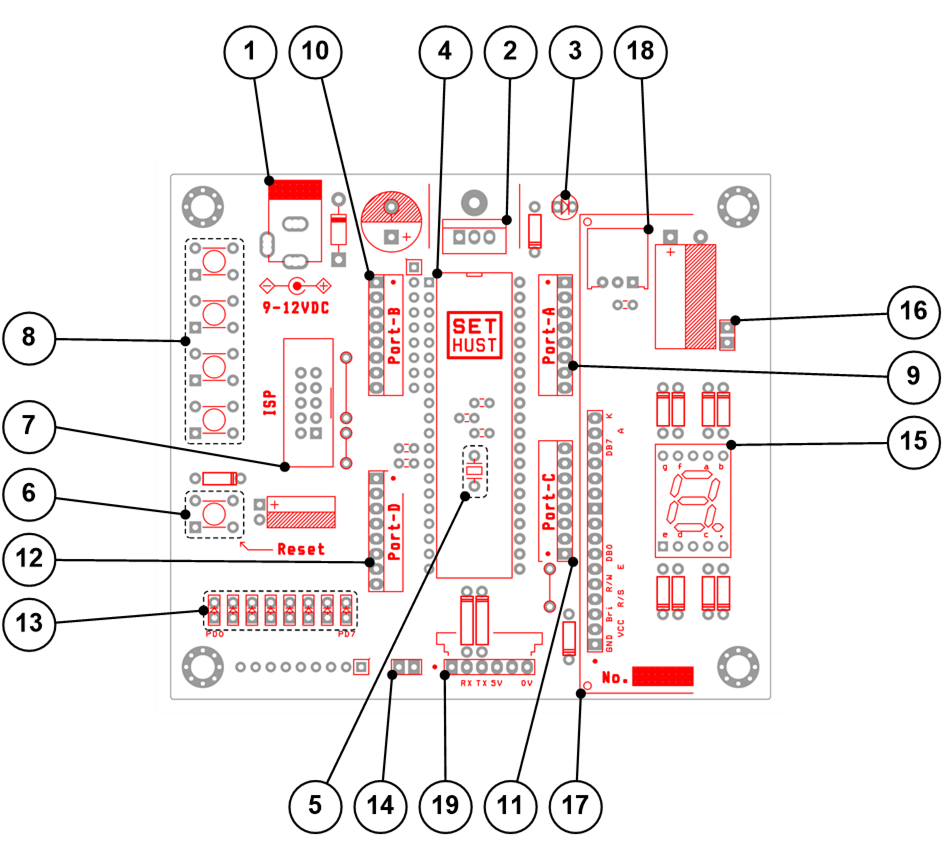
## Cấu trúc của mạch Kit

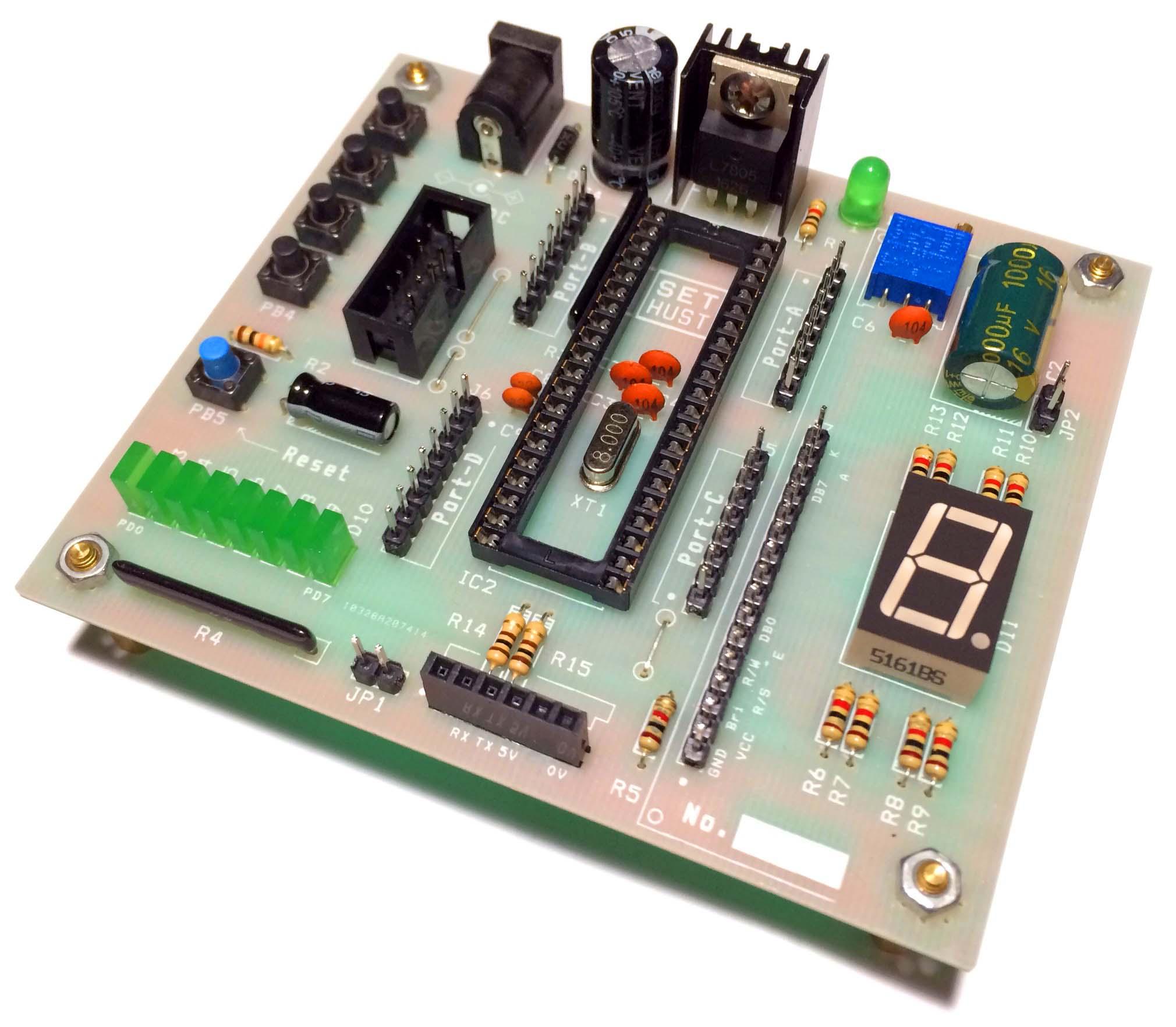
Mạch Kit được cung cấp để đáp ứng các ứng dụng cơ bản có cấu trúc và chức năng của từng linh kiện quan trọng được nêu trong bảng dưới đây. Đầu tiên ta sẽ tìm hiểu về sơ đồ nguyên lý của bộ kit cùng với đó là mạch nguyên lý được vẽ bởi Altium một trong những công cụ vẽ mạch đi dây cũng như để in mạch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

Ảnh có chứa văn bản, phần mềm, ảnh chụp màn hình, Phần mềm đa phương tiện

Mô tả được tạo tự động

**Hình 2.1 Sơ đồ nguyên lý của bộ kit trên Altium**



****

**Hình .2 Cấu trúc mạch kit được cung cấp**

**Hình .3 Sơ đồ mạch in, quan sát trên phần mềm thiết kế**

**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) |

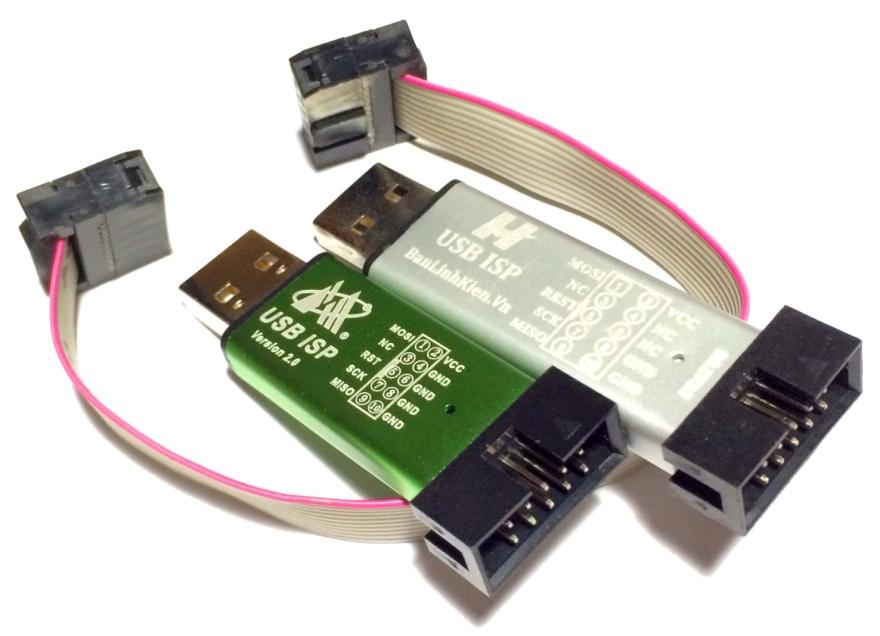
## Các thông số chính của Kit

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

## Mạch nạp mã nguồn

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



Hình 2.3 Ảnh thực tế của USB-ISP



Hình 2.4 Sơ đồ chân của USB-ISP

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

## Màn hình LCD 1602 và module DS18B20

### Màn hình LCD 1602

* Thiết bị hiển thị [**LCD 1602**](http://www.suachualaptop24h.com/Linh-kien-laptop.html) (Liquid Crystal Display) được sử dụng trong rất nhiều các ứng dụng của VĐK. LCD 1602 có rất nhiều ưu điểm so với các dạng hiển thị khác như: khả năng hiển thị kí tự đa dạng (chữ, số, kí tự đồ họa); dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tiêu tốn rất ít tài nguyên hệ thống, giá thành rẻ,…
* Thông số kĩ thuật  của sản phẩm LCD 1602:

- Điện áp MAX : 7V

- Điện áp MIN : - 0,3V

- Hoạt động ổn định : 2.7-5.5V

- Điện áp ra mức cao : > 2.4

- Điện áp ra mức thấp : <0.4V

- Dòng điện cấp nguồn : 350uA - 600uA

- Nhiệt độ hoạt động : - 30 - 75 độ C

* Chức năng của từng chân LCD 1602:

- Chân số 1 - VSS : chân nối đất cho LCD được nối với GND của mạch điều khiển

- Chân số 2 - VDD : chân cấp nguồn cho LCD, được nối với VCC=5V của mạch điều khiển

- Chân số 3 - VE : điều chỉnh độ tương phản của LCD

- Chân số 4 - RS : chân chọn thanh ghi, được nối với logic "0" hoặc logic "1":

  + Logic “0”: Bus DB0 - DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ “ghi” -   write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ “đọc” - read)

   + Logic “1”: Bus DB0 - DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD

- Chân số 5 - R/W : chân chọn chế độ đọc/ghi (Read/Write), được nối với logic “0” để ghi hoặc nối với logic “1” đọc

- Chân số 6 - E : chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân này như sau:

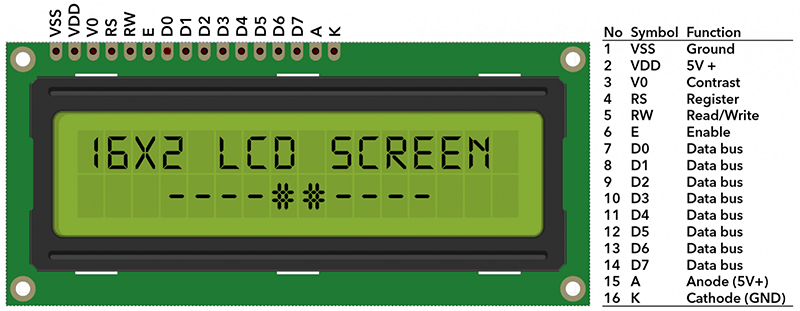
   + Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào thanh ghi bên trong khi phát hiện một xung (high-to-low transition) của tín hiệu chân E

   + Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện cạnh lên (low-to-high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp

- Chân số 7 đến 14 - D0 đến D7: 8 đường của bus dữ liệu dùng để trao đổi thông tin với MPU. Có 2 chế độ sử dụng 8 đường bus này là: Chế độ 8 bit (dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7) và Chế độ 4 bit (dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7)

- Chân số 15 - A : nguồn dương cho đèn nền

- Chân số 16 - K : nguồn âm cho đèn nền



Hình 2.5 Màn hình LCD 1602

### 2.4.2 Module DS18B20

Module DS18B20 là loại cảm biến loại digital với độ phân giải cao (12 bit), vì vậỵ cảm biến có thể đo nhiệt độ chính xác, với sai số rất nhỏ và sử dụng được ở nhiều môi trường, điều kiện khác nhau và có thể sử dụng các loại chip thông dụng để nhận và xử lý dữ liệu như 8051, AVR, PIC, Arduino . . .

**Thô​ng số​ kỹ​ thuật​:**

- Kích thước: 28mm x 12mm x10 mm (L x W x H)

- Chip chính: 18B20 cảm biến nhiệt độ

- Điện áp làm việc: DC 5V

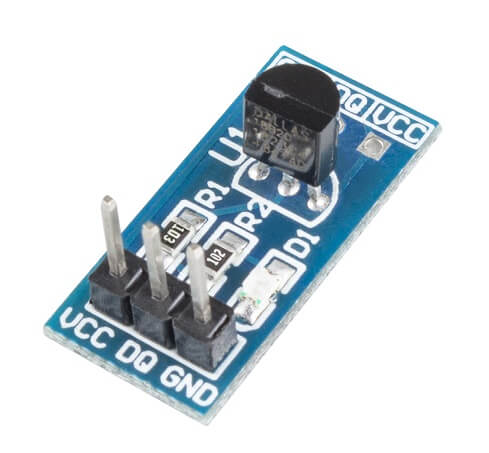
- Phạm vi đo nhiệt độ (độ C): -55 ~ 125o

- Sai số cho phép: 0.5

- Chân VCC nối với dương nguồn

- Chân DQ là chân dữ liệu

- Chân GND nối đất



**Hình 2.6 Module DS18B20**

**Hình . Màn hình LCD 1602 dụng trong bộ Kit**

**Hình . Module UART-USB sử dụng chip FT232RL**

# CHƯƠNG 3. THIẾT KẾ MẠCH ĐO NHIỆT ĐỘ

## Phần mềm biên dịch Atmel studio và nạp code Progisp

### 3.1.1 Phần mềm biên dịch Atmel studio

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 Atmel Studio được sử dụng là mang lại sự hiểu biết về VĐK đang lập trình – vì với Atmel 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 Atmel Studio 6, vào*File > New > Project* **.** Sau khi thay đổi tên và đường dẫn lưu project thì ta bấm OK .

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, máy tính

Mô tả được tạo tự động

**Hình 3.1 Giao diện tạo Project mới trong Atmel Studio 6**

**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 16.

Ảnh có chứa văn bản, phần mềm, ảnh chụp màn hình, máy tính

Mô tả được tạo tự động

**Hình 3.2 Giao diện chọn loại vi điều khiển**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, máy tính

Mô tả được tạo tự động

**Hình 3.3 Giao diện làm việc chính sau khi tạo project**

**Bước 3:** Cuối cùng thì chương trình sẽ hiện ra file chưa hàm main của project từ đó ta có thể code chương trình theo ý muốn.

### 3.1.2 Phần mềm nạp code Progisp 1.72

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

**Hình 3.4 Giao diện chính của Progisp 1.72**

Để nạp được file hex ta cần chỉnh cấu hình Fuse bit cho VĐK nhưhình rồi chọn file hex vừa được tạo để nạp xuống VĐK.

Table

Description automatically generated

**Hình . Thiết lập Fuse bit trong PROGISP**

**Hình 3.5 Cấu hình Fuse&Lock cho Progisp**

## Mô phỏng và thiết kế

### Mô phỏng trên phần mềm proteus 8.13

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.

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, biểu đồ

Mô tả được tạo tự động

**Hình 4.1 Mô phỏng mạch đo nhiệt độ trực tiếp từ DS18B20 trên Proteus**

Tiến hành biên dịch sang mã máy và mô phỏng bằng cách, click chuột vàoVĐK 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*.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Mô tả được tạo tự động

**Hình 4.2 Thiết lập chọn file mã máy cho VĐK để mô phỏng trong Proteus**

**Hình . Kết quả mô phỏng trên Proteus**

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

Ảnh có chứa văn bản, phần mềm, ảnh chụp màn hình, Phần mềm đa phương tiện

Mô tả được tạo tự động

**Hình 4.3 Sơ đồ nguyên lí trên Altium 19**

Ảnh có chứa văn bản, ảnh chụp màn hình, Phần mềm đa phương tiện, phần mềm

Mô tả được tạo tự động

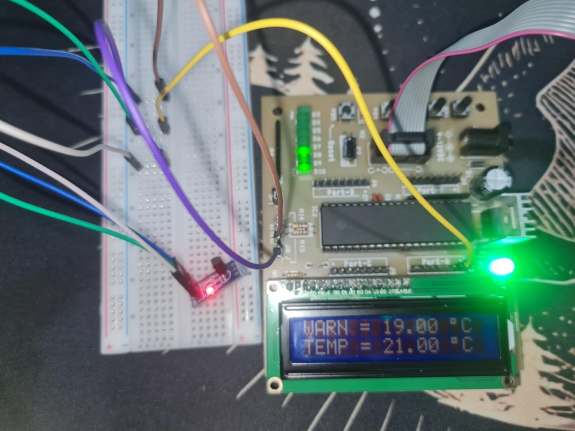
**Hình 4.4 Mạch PCB trên Altium**

A picture containing text, electronics

Description automatically generated

**Hình 4.5 Mô hình 3D trên Altium**

## Mạch thực tế



**Hình 4.6 Mạch thực tế**

## Nhận xét

Độ chính xác cao, mạch đơn giản hơn do không cần phải sửdụng khuếch đại mà dùng luôn tính năng của VĐK. So với mạch đo sử dụng cảm biến nhiệt độ tương tự như LM35 mạch sử dụng module DS18B20 có độ chính xác cao hơn, sai số nhỏ hơn 0.5°C.

# 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 DS18B20*“, nhóm 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. 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 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, nhóm 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 chúng em xin chân thành cảm ơn thầy Vũ Hồng Vinh 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. Thầy Vũ Hồng Vinh, “Tài liệu hướng dẫn đồ án 2”.
2. [Datasheet DS18B20](https://pdf1.alldatasheet.com/datasheet-pdf/view/58557/DALLAS/DS18B20.html)
3. [ATMEGA16 Datasheet(PDF) - ATMEL Corporation (alldatasheet.com)](https://www.alldatasheet.com/datasheet-pdf/pdf/78532/ATMEL/ATMEGA16.html)
4. [16x2 LCD Display Module Pinout, Features, Description & Datasheet (components101.com)](https://components101.com/displays/16x2-lcd-pinout-datasheet)

**PHỤ LỤC – CODE CHƯƠNG TRÌNH**

**File Do nhiet do.c**

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include "lcd.h"

#include "ds18b20.h"

#include <stdio.h>

#include <stdlib.h>

#define BTN\_INC\_DDR DDRB

#define BTN\_INC\_PIN PINB

#define BTN\_INC (1 << 0) //PB0

#define BTN\_DEC\_DDR DDRB

#define BTN\_DEC\_PIN PINB

#define BTN\_DEC (1 << 1) // PB1

#define ALARM\_LED\_DDR DDRD

#define ALARM\_LED\_PORT PORTD

#define ALARM\_LED (1 << 0); //PD0

float temperature; /\* Gia tri nhiet do doc \*/

float temperatureSet = 25; /\* Gia tri nhiet do dat \*/

unsigned long ms = 0; /\* Bien dem thoi gian (1ms) \*/

unsigned long prevMs = 0; /\* Bien thoi diem truoc do \*/

char strBuffer[64]; /\* Bo dem du lieu hien thi \*/

int main(void)

{

    /\* Cau hinh vao ra \*/

    BTN\_INC\_DDR &= ~BTN\_INC; /\* PB0 la dau vao \*/

    BTN\_DEC\_DDR &= ~BTN\_DEC; /\* PB1 la dau vao \*/

    ALARM\_LED\_DDR |= ALARM\_LED; /\* PB4 la dau ra \*/

    ALARM\_LED\_PORT &= ~ALARM\_LED; /\* PB4 = 0 de bat den \*/

    /\* Khoi tao lcd \*/

    LCD\_init();

    /\* Khoi tao timer 1ms <=> Fupdate = 1000Hz \*/

    /\* Freq / Fupdate = autoreload \* prescaler \*/

    /\* 8000000Hz / 1000Hz = (256 - TCNT0) \* prescaler \*/

    /\* chon prescaler = 64 => TCNT0 = 131 \*/

    TCCR0 = (1 << CS00) | (1 << CS01);

    TCNT0 = 131;

    TIMSK = (1 << TOIE0); /\* Cho phep ngat TIMER0 \*/

    /\* Cho phep ngat toan cuc \*/

    sei();

    while (1)

    {

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Nut DEC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Neu nut giam duoc nhan \*/

        if (!((BTN\_DEC\_PIN & BTN\_DEC) == BTN\_DEC))

        {

            /\* Chong doi phim \*/

            \_delay\_ms(20);

            /\* Neu chac chan duoc nhan \*/

            if (!((BTN\_DEC\_PIN & BTN\_DEC) == BTN\_DEC))

            {

                /\* Giam gia tri nhiet do xuong 1 \*/

                if (temperatureSet > 0)

                {

                    temperatureSet=temperatureSet-1;

                }

            }

            /\* Cho cho den khi nut nhan duoc nha \*/

            while (!((BTN\_DEC\_PIN & BTN\_DEC) == BTN\_DEC));

        }

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Nut INC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Neu nut tang duoc nhan \*/

        if (!((BTN\_INC\_PIN & BTN\_INC) == BTN\_INC))

        {

            /\* Chong doi phim \*/

            \_delay\_ms(20);

            /\* Neu chac chan duoc nhan \*/

            if (!((BTN\_INC\_PIN & BTN\_INC) == BTN\_INC))

            {

                /\* Giam gia tri nhiet do xuong 1 \*/

                if (temperatureSet < 99)

                {

                    temperatureSet=temperatureSet+1;

                }

            }

            /\* Cho cho den khi nut nhan duoc nha \*/

            while (!((BTN\_INC\_PIN & BTN\_INC) == BTN\_INC));

        }

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Hien thi nhiet do dat \*\*\*\*\*/

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Chuyen float sang kieu chuoi \*/

        dtostrf(temperatureSet, -6, 2, strBuffer);

        /\* Hien thi ra LCD \*/

        LCD\_setPos(0, 0);

        LCD\_string("WARN = ");

        LCD\_setPos(7, 0);

        LCD\_string(strBuffer);

        LCD\_setPos(13, 0);

        LCD\_data(0xDF);

        LCD\_char('C');

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Hien thi nhiet do doc \*\*\*\*\*/

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* Dinh thoi 500ms doc nhiet do \*/

        if (ms - prevMs >= 500)

        {

            /\* Sao luu thoi diem truoc do \*/

            prevMs = ms;

            /\* Doc gia tri nhiet do \*/

            temperature = DS18B20\_getTemp();

            /\* Kiem tra canh bao \*/

            if (temperature > temperatureSet)

            {

                /\* Dao trang thai led \*/

                ALARM\_LED\_PORT ^= ALARM\_LED;

            }

            else

            {

                /\* Tat led \*/

                ALARM\_LED\_PORT &= ~ALARM\_LED;

            }

        }

        /\* Chuyen float sang kieu chuoi \*/

        dtostrf(temperature, -6, 2, strBuffer);

        /\* Hien thi ra LCD \*/

        LCD\_setPos(0, 1);

        LCD\_string("WARN = ");

        LCD\_setPos(7, 1);

        LCD\_string(strBuffer);

        LCD\_setPos(13, 1);

        LCD\_data(0xDF);

        LCD\_char('C');

    }

}

/\* Trinh phuc vu ngat timer 0 \*/

ISR (TIMER0\_OVF\_vect)

{

    /\* Nap lai gia tri \*/

    TCNT0 = 131;

    /\* Tang gia tri dem \*/

    ms++;

}

**\*/ File ds18B20.h**  
#ifndef DS18B20\_H\_

#define DS18B20\_H\_

#include <avr/io.h>

#define DS18B20\_DDR       DDRA

#define DS18B20\_PORT      PORT

#define DS18B20\_PIN       PINA

#define DS18B20\_INPUT     1

#define DS18B20\_CMD\_CONVERTTEMP     0x44

#define DS18B20\_CMD\_RSCRATCHPAD     0xBE

#define DS18B20\_CMD\_WSCRATCHPAD     0x4E

#define DS18B20\_CMD\_CPYSCRATCHPAD   0x48

#define DS18B20\_CMD\_RECEEPROM       0xB8

#define DS18B20\_CMD\_RPWRSUPPLY      0xB4

#define DS18B20\_CMD\_SEARCHROM       0xF0

#define DS18B20\_CMD\_READROM         0x33

#define DS18B20\_CMD\_MATCHROM        0x55

#define DS18B20\_CMD\_SKIPROM         0xCC

#define DS18B20\_CMD\_ALARMSEARCH     0xEC

uint8\_t DS18B20\_reset();

void DS18B20\_writeBit(uint8\_t bit);

uint8\_t DS18B20\_readBit(void);

void DS18B20\_writeByte(uint8\_t byte);

uint8\_t DS18B20\_readByte(void);

float DS18B20\_getTemp();

#endif

**\*/File ds18B20.c**

#define F\_CPU 8000000UL

#include <util/delay.h>

#include "ds18B20.h"

uint8\_t DS18B20\_reset() {

    uint8\_t i;

    DS18B20\_PORT &= ~(1 << DS18B20\_INPUT);

    DS18B20\_DDR |= (1 << DS18B20\_INPUT);

    \_delay\_us(480);

    DS18B20\_DDR &= ~(1 << DS18B20\_INPUT);

    \_delay\_us(60);

    i = (DS18B20\_PIN & (1 << DS18B20\_INPUT));

    \_delay\_us(420);

    return i;

}

void DS18B20\_writeBit(uint8\_t bit) {

    DS18B20\_PORT &= ~(1 << DS18B20\_INPUT);

    DS18B20\_DDR |= (1 << DS18B20\_INPUT);

    \_delay\_us(1);

    if (bit) DS18B20\_DDR &= ~(1 << DS18B20\_INPUT);

    \_delay\_us(60);

    DS18B20\_DDR &= ~(1 << DS18B20\_INPUT);

}

uint8\_t DS18B20\_readBit(void) {

    uint8\_t bit = 0;

    DS18B20\_PORT &= ~ (1 << DS18B20\_INPUT);

    DS18B20\_DDR |= (1 << DS18B20\_INPUT);

    \_delay\_us(1);

    DS18B20\_DDR &= ~(1 << DS18B20\_INPUT);

    \_delay\_us(14);

    if (DS18B20\_PIN & (1 << DS18B20\_INPUT))

    bit = 1;

    \_delay\_us(45);

    return bit;

}

void DS18B20\_writeByte(uint8\_t byte) {

    uint8\_t i = 8;

    while (i--) {

        DS18B20\_writeBit(byte & 0x01);

        byte >>= 1;

    }

}

uint8\_t DS18B20\_readByte(void) {

    uint8\_t i = 8, n = 0;

    while (i--) {

        n >>= 1;

        n |= (DS18B20\_readBit() << 7);

    }

    return n;

}

float DS18B20\_getTemp() {

    uint8\_t temperature\_l;

    uint8\_t temperature\_h;

    uint16\_t temperature\_d;

    float temperature\_c = 0;

    DS18B20\_reset();

    DS18B20\_writeByte(DS18B20\_CMD\_SKIPROM);

    DS18B20\_writeByte(DS18B20\_CMD\_CONVERTTEMP);

    while (!DS18B20\_readBit());

    DS18B20\_reset();

    DS18B20\_writeByte(DS18B20\_CMD\_SKIPROM);

    DS18B20\_writeByte(DS18B20\_CMD\_RSCRATCHPAD);

    temperature\_l = DS18B20\_readByte();

    temperature\_h = DS18B20\_readByte();

    temperature\_d = (temperature\_h << 8) | temperature\_l;

    if ((temperature\_d & 0xF800) == 0xF800) {

        temperature\_c = (0xFFFF - temperature\_d + 1)  \* 0.05 \* (-1.0);

        } else {

        temperature\_c = (temperature\_d - (temperature\_d % 5)) \* 0.05;

    }

    return temperature\_c;

}

**\*/File lcd.h**

#ifndef LCD\_H\_

#define LCD\_H\_

#include <avr/io.h>

#define LCD\_RS\_DDR        DDRD

#define LCD\_RS\_PORT       PORTD

#define LCD\_RS            6

#define LCD\_RW\_DDR        DDRD

#define LCD\_RW\_PORT       PORTD

#define LCD\_RW            5

#define LCD\_EN\_DDR        DDRD

#define LCD\_EN\_PORT       PORTD

#define LCD\_EN            7

#define LCD\_DDR           DDRC

#define LCD\_PORT          PORTC

#define LCD\_D4            4

#define LCD\_D5            5

#define LCD\_D6            6

#define LCD\_D7            7

void LCD\_pulseEnable();

void LCD\_write4Bits(uint8\_t nb);

void LCD\_command(uint8\_t cmd);

void LCD\_data(uint8\_t dt);

void LCD\_char(char chr);

void LCD\_string(char \* str);

void LCD\_setPos(uint8\_t x, uint8\_t y);

void LCD\_clear();

void LCD\_init();

#endif

**\*/File lcd.c**

#define F\_CPU 8000000UL

#include <util/delay.h>

#include <string.h>

#include "lcd.h"

void LCD\_pulseEnable() {

    LCD\_EN\_PORT &= ~(1 << LCD\_EN);

    \_delay\_us(1);

    LCD\_EN\_PORT |= (1 << LCD\_EN);

    \_delay\_us(1);

    LCD\_EN\_PORT &= ~(1 << LCD\_EN);

    \_delay\_ms(1);

}

void LCD\_write4Bits(uint8\_t nb) {

    if (((nb >> 0) & 0x01) == 1) { LCD\_PORT |= (1 << LCD\_D4); } else { LCD\_PORT &= ~(1 << LCD\_D4); }

    if (((nb >> 1) & 0x01) == 1) { LCD\_PORT |= (1 << LCD\_D5); } else { LCD\_PORT &= ~(1 << LCD\_D5); }

    if (((nb >> 2) & 0x01) == 1) { LCD\_PORT |= (1 << LCD\_D6); } else { LCD\_PORT &= ~(1 << LCD\_D6); }

    if (((nb >> 3) & 0x01) == 1) { LCD\_PORT |= (1 << LCD\_D7); } else { LCD\_PORT &= ~(1 << LCD\_D7); }

    LCD\_pulseEnable();

}

void LCD\_command(uint8\_t cmd) {

    LCD\_RS\_PORT &= ~(1 << LCD\_RS);

    LCD\_RW\_PORT &= ~(1 << LCD\_RW);

    LCD\_write4Bits(cmd >> 4);

    LCD\_write4Bits(cmd);

}

void LCD\_data(uint8\_t dt) {

    LCD\_RS\_PORT |= (1 << LCD\_RS);

    LCD\_RW\_PORT &= ~(1 << LCD\_RW);

    LCD\_write4Bits(dt >> 4);

    LCD\_write4Bits(dt);

}

void LCD\_char(char chr) {

    LCD\_data((uint8\_t) chr);

}

void LCD\_string(char \* str) {

    uint8\_t i = 0;

    while (str[i] != 0) {

        LCD\_char(str[i]);

        i++;

    }

}

void LCD\_setPos(uint8\_t x, uint8\_t y) {

    uint8\_t firstChar[] = {0x80, 0xC0, 0x94, 0xD4};

    LCD\_command(firstChar[y] + x);

}

void LCD\_clear() {

    LCD\_command(0x01);

    \_delay\_ms(2);

}

void LCD\_init() {

    LCD\_RS\_DDR |= (1 << LCD\_RS);

    LCD\_RW\_DDR |= (1 << LCD\_RW);

    LCD\_EN\_DDR |= (1 << LCD\_EN);

    LCD\_DDR |= (1 << LCD\_D4);

    LCD\_DDR |= (1 << LCD\_D5);

    LCD\_DDR |= (1 << LCD\_D6);

    LCD\_DDR |= (1 << LCD\_D7);

    \_delay\_ms(50);

    LCD\_RS\_PORT &= ~(1 << LCD\_RS);

    LCD\_EN\_PORT &= ~(1 << LCD\_EN);

    LCD\_RW\_PORT &= ~(1 << LCD\_RW);

    LCD\_write4Bits(0x03);

    \_delay\_ms(5);

    LCD\_write4Bits(0x03);

    \_delay\_ms(5);

    LCD\_write4Bits(0x03);

    \_delay\_ms(1);

    LCD\_write4Bits(0x02);

    LCD\_command(0x28);

    LCD\_command(0x0C);

    LCD\_clear();

    LCD\_command(0x06);

}

void lcdCenter(uint8\_t y, char \* s) {

    uint8\_t len = strlen(s);

    uint8\_t x = (16 - len) / 2;

    for (uint8\_t i = 0; i <= x; i++) LCD\_char(' ');

    LCD\_setPos(x, y);

    LCD\_string(s);

    for (uint8\_t i = 0; i < (16 - x - len); i++) LCD\_char(' ');

}

void lcdRunning(uint8\_t x, uint8\_t y, uint8\_t pos, uint8\_t dir) {

    switch (pos) {

        case 1:

        LCD\_setPos(x, y);

        LCD\_string((dir) ? ">  " : "  <");

        break;

        case 2:

        LCD\_setPos(x, y);

        LCD\_string((dir) ? ">> " : " <<");

        break;

        case 3:

        LCD\_setPos(x, y);

        LCD\_string((dir) ? ">>>" : "<<<");

        break;

        case 4:

        LCD\_setPos(x, y);

        LCD\_string((dir) ? " >>" : "<< ");

        break;

        case 5:

        LCD\_setPos(x, y);

        LCD\_string((dir) ? "  >" : "<  ");

        break;

        default:

        LCD\_setPos(x, y);

        LCD\_string("   ");

    }

}