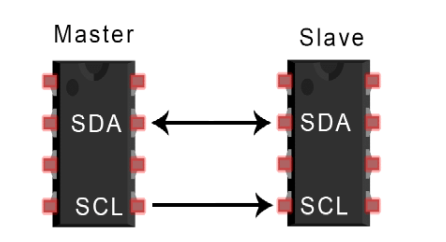
1. Chuẩn giao tiếp I2C:

**-Giới thiệu giao tiếp I2C**

I2C kết hợp các tính năng tốt nhất của SPI và UART. Với I2C, bạn có thể kết nối nhiều slave với một master duy nhất (như SPI) và bạn có thể có nhiều master điều khiển một hoặc nhiều slave. Điều này thực sự hữu ích khi bạn muốn có nhiều hơn một vi điều khiển ghi dữ liệu vào một thẻ nhớ duy nhất hoặc hiển thị văn bản trên một màn hình LCD.



Giống như giao tiếp UART, I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị:

SDA (Serial Data) - đường truyền cho master và slave để gửi và nhận dữ liệu.

SCL (Serial Clock) - đường mang tín hiệu xung nhịp.

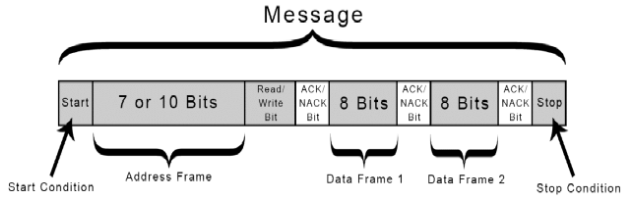
I2C là một giao thức truyền thông nối tiếp, vì vậy dữ liệu được truyền từng bit dọc theo một đường

duy nhất (đường SDA).

Giống như SPI, I2C là đồng bộ, do đó đầu ra của các bit được đồng bộ hóa với việc lấy mẫu các bit bởi một tín hiệu xung nhịp được chia sẻ giữa master và slave. Tín hiệu xung nhịp luôn được điều khiển bởi master.

**Cách hoạt động của I2C**

Với I2C, dữ liệu được truyền trong các tin nhắn. Tin nhắn được chia thành các khung dữ liệu. Mỗi tin nhắn có một khung địa chỉ chứa địa chỉ nhị phân của địa chỉ slave và một hoặc nhiều khung dữ liệu chứa dữ liệu đang được truyền. Thông điệp cũng bao gồm điều kiện khởi động và điều kiện dừng, các bit đọc / ghi và các bit ACK / NACK giữa mỗi khung dữ liệu:



Hình 1.Khung dữ liệu I2C

**Điều kiện khởi động:** Đường SDA chuyển từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ mức cao xuống mức thấp.

**-Điều kiện dừng:** Đường SDA chuyển từ mức điện áp thấp sang mức điện áp cao sau khi đường SCL chuyển từ mức thấp lên mức cao.

**-Khung địa chỉ:** Một chuỗi 7 hoặc 10 bit duy nhất cho mỗi slave để xác định slave khi master muốn giao tiếp với nó.

**-Bit Đọc / Ghi:** Một bit duy nhất chỉ định master đang gửi dữ liệu đến slave (mức điện áp thấp) hay yêu cầu dữ liệu từ nó (mức điện áp cao).

**-Bit ACK / NACK:** Mỗi khung trong một tin nhắn được theo sau bởi một bit xác nhận / không xác nhận. Nếu một khung địa chỉ hoặc khung dữ liệu được nhận thành công, một bit ACK sẽ được trả lại cho thiết bị gửi từ thiết bị nhận.

**-Địa chỉ**

I2C không có các đường Slave Select như SPI, vì vậy cần một cách khác để cho slave biết rằng dữ liệu đang được gửi đến slave này chứ không phải slave khác. Nó thực hiện điều này bằng cách định địa chỉ. Khung địa chỉ luôn là khung đầu tiên sau bit khởi động trong một tin nhắn mới.

Master gửi địa chỉ của slave mà nó muốn giao tiếp với mọi slave được kết nối với nó. Sau đó, mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ phù hợp, nó sẽ gửi lại một bit ACK điện áp thấp cho master. Nếu địa chỉ không khớp, slave không làm gì cả và đường SDA vẫn ở mức cao.

**-Bit đọc / ghi**

Khung địa chỉ bao gồm một bit duy nhất ở cuối tin nhắn cho slave biết master muốn ghi dữ liệu vào nó hay nhận dữ liệu từ nó. Nếu master muốn gửi dữ liệu đến slave, bit đọc / ghi ở mức điện áp thấp. Nếu master đang yêu cầu dữ liệu từ slave, thì bit ở mức điện áp cao.

**-Khung dữ liệu**

Sau khi master phát hiện bit ACK từ slave, khung dữ liệu đầu tiên đã sẵn sàng được gửi.

Khung dữ liệu luôn có độ dài 8 bit và được gửi với bit quan trọng nhất trước. Mỗi khung dữ liệu ngay sau đó là một bit ACK / NACK để xác minh rằng khung đã được nhận thành công. Bit ACK phải được nhận bởi master hoặc slave (tùy thuộc vào cái nào đang gửi dữ liệu) trước khi khung dữ liệu tiếp theo có thể được gửi.

Sau khi tất cả các khung dữ liệu đã được gửi, master có thể gửi một điều kiện dừng cho slave để tạm dừng quá trình truyền. Điều kiện dừng là sự chuyển đổi điện áp từ thấp lên cao trên đường SDA sau khi chuyển tiếp từ thấp lên cao trên đường SCL , với đường SCL vẫn ở mức cao.

**-Các bước truyền dữ liệu I2C**

1.Master gửi điều kiện khởi động đến mọi slave được kết nối bằng cách chuyển đường SDA từ mức điện áp cao sang mức điện áp thấp trước khi chuyển đường SCL từ mức cao xuống mức thấp.

2.Master gửi cho mỗi slave địa chỉ 7 hoặc 10 bit của slave mà nó muốn giao tiếp, cùng với bit đọc / ghi.

3.Mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ trùng khớp, slave sẽ trả về một bit ACK bằng cách kéo dòng SDA xuống thấp cho một bit. Nếu địa chỉ từ master không khớp với địa chỉ của slave, slave rời khỏi đường SDA cao.

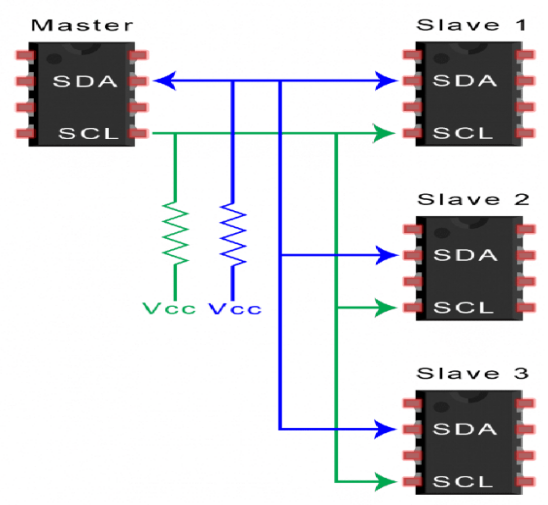
4.Master gửi hoặc nhận khung dữ liệu.

5.Sau khi mỗi khung dữ liệu được chuyển, thiết bị nhận trả về một bit ACK khác cho thiết bị gửi để xác nhận đã nhận thành công khung.

6.Để dừng truyền dữ liệu, master gửi kiều kiện dừng đến slave bằng cách chuyển đổi mức cao SCL trước khi chuyển mức cao SDA.

**Một master với nhiều slave**

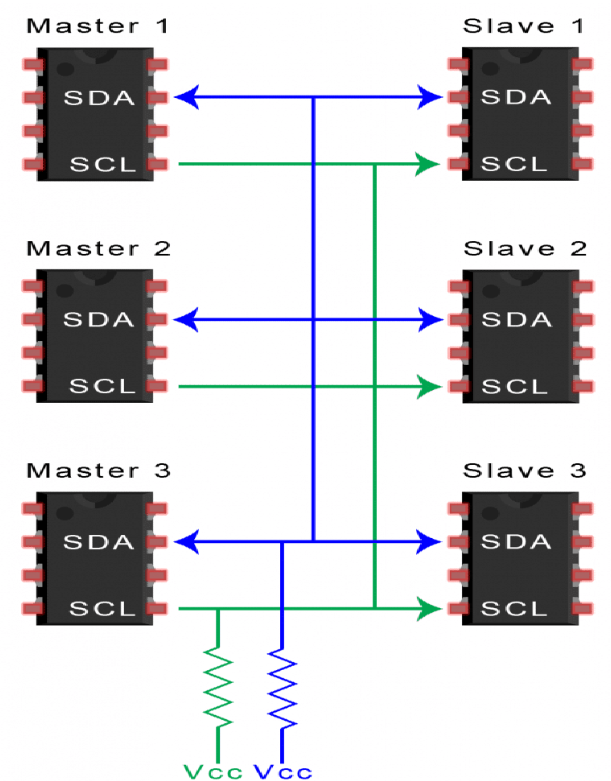
Vì I2C sử dụng định địa chỉ nên nhiều slave có thể được điều khiển từ một master duy nhất. Với địa chỉ 7 bit sẽ có 128 (2 mũ 7) địa chỉ duy nhất. Việc sử dụng địa chỉ 10 bit không phổ biến, nhưng nó cung cấp 1.024 (2 mũ 10) địa chỉ duy nhất. Để kết nối nhiều slave đến một master duy nhất, bạn có thể đấu dây như thế này, với điện trở kéo lên 4,7K Ohm kết nối đường SDA và SCL với Vcc:



Hình 2 Một master giao tiếp nhiều slave

**Nhiều master với nhiều slave**

Nhiều master có thể được kết nối với một slave hoặc nhiều slave. Sự cố với nhiều master trong cùng một hệ thống xảy ra khi hai master cố gắng gửi hoặc nhận dữ liệu cùng một lúc qua đường SDA. Để giải quyết vấn đề này, mỗi master cần phải phát hiện xem đường SDA thấp hay cao trước khi truyền tin nhắn. Nếu đường SDA thấp, điều này có nghĩa là một master khác có quyền điều khiển bus và master đó phải đợi để gửi tin nhắn. Nếu đường SDA cao thì có thể truyền tin nhắn an toàn. Để kết nối nhiều master với nhiều slave, hãy sử dụng sơ đồ sau, với các điện trở kéo lên 4,7K Ohm kết nối các đường SDA và SCL với Vcc:



Hình 3 Nhiều master giao tiếp nhiều slave

**Ưu điểm và nhược điểm của I2C**

Có rất nhiều điều ở I2C có thể khiến nó nghe có vẻ phức tạp so với các giao thức khác, nhưng có một số lý do chính đáng khiến bạn có thể muốn hoặc không muốn sử dụng I2C để kết nối với một thiết bị cụ thể:

-Ưu điểm

Chỉ sử dụng hai dây

Hỗ trợ nhiều master và nhiều slave

Bit ACK / NACK xác nhận mỗi khung được chuyển thành công

Phần cứng ít phức tạp hơn so với UART

Giao thức nổi tiếng và được sử dụng rộng rãi

-Nhược điểm

Tốc độ truyền dữ liệu chậm hơn SPI

Kích thước của khung dữ liệu bị giới hạn ở 8 bit

Cần phần cứng phức tạp hơn để triển khai so với SPI

1. EEPROM 24c02
   1. Giới thiệu

EEPROM 24c02 là một loại bộ nhớ EEPROM (Electrically Erasable Programmable Read-Only Memory) có dung lượng 2 kilobit, tương đương với 256 byte. Chữ "24c02" thường được sử dụng để chỉ định loại và dung lượng của chip EEPROM này.

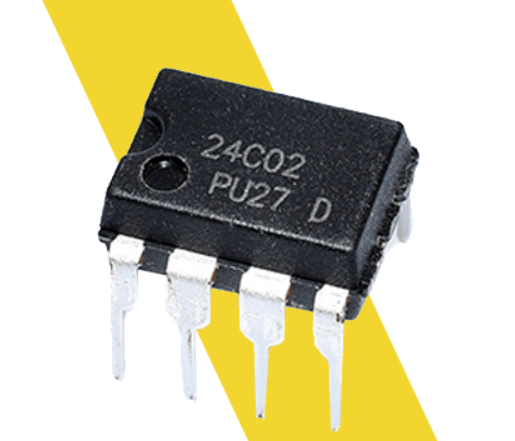
EEPROM là loại bộ nhớ không xóa được bằng điện áp, điều này có nghĩa là dữ liệu có thể được ghi và xóa đi mà không cần loại bỏ chip khỏi mạch điện. Điều này làm cho EEPROM rất hữu ích trong các ứng dụng yêu cầu lưu trữ dữ liệu thường xuyên, như trong các vi mạch nhúng, máy tính nhúng, và các thiết bị điện tử tiêu thụ ít điện năng.

Chip EEPROM 24c02 sử dụng giao thức I2C (Inter-Integrated Circuit) để truyền dữ liệu giữa nó và các thành phần khác trong mạch điện. Giao thức I2C cho phép nhiều thiết bị kết nối với nhau thông qua cùng một đường truyền dẫn và chia sẻ cùng một địa chỉ, giúp giảm số lượng chân kết nối cần thiết.

Dữ liệu lưu trữ trong EEPROM có thể bị xóa và ghi lại nhiều lần mà không gây tổn thương cho chip, điều này làm cho nó trở thành một phương tiện lưu trữ linh hoạt trong nhiều ứng dụng.

**Thông số:**

* Điện áp hoạt động: 2.7V to 5.5V
* Bộ nhở nội: 256x8 (2K)
* Tần số hoạt động: 100kHz
* Chế độ viết: 8Byte trang dữ liệu
* Thời gian nhớ dữ liệu: ~100năm
* Tự động hẹn giờ viết Cycles : 5ms
* Dải nhiệt dộ: -65c to 150C
* Dòng điện ra: 5mA



### Hình 4 IC Nhớ AT24C02 24C02 EEPROM

* 1. Các chân của EEPROM 24c02

|  |  |
| --- | --- |
| E0-E2 | Các chân cho phép EEPROM |
| SDA | Đường truyền cho vi điều khiển và EEPROM để gửi và nhận dữ liệu |
| SCL | Đường mang tín hiệu xung nhịp. |
| MODE | Chế độ viết nhiều trang(C version) |
|  | Chân điều khiển việc viết data (W version) |
| Vcc | Nguồn 5V |
| Vss | Chân nối đất |

**Serial Clock (SCL):** Được sử dụng để đồng bộ hóa dữ liệu vào và ra khỏi bộ nhớ. Bạn có thể kết nối một điện trở từ đường SCL đến VCC để làm nhiệm vụ pull-up.

**Serial Data (SDA):** Đường này là hai chiều và được sử dụng để truyền dữ liệu vào hoặc ra khỏi bộ nhớ. Nó là một đầu ra open drain có thể thực hiện phép tính OR với các tín hiệu open drain hoặc open collector khác trên bus. Một điện trở phải được kết nối từ đường SDA đến VCC để làm pull-up.

**Chip Enable (E2 - E0):** Các đầu vào này được sử dụng để đặt 3 bit ít trọng số nhất (b3, b2, b1) của mã lựa chọn thiết bị 7 bit. Các đầu vào này có thể được đặt động hoặc kết nối với VCC hoặc VSS để xác định mã lựa chọn thiết bị.

**Mode (MODE):** Đầu vào này có sẵn ở chân 7 và có thể được đặt động. Nó phải ở mức thấp (VIL) hoặc cao (VIH) cho chế độ ghi byte, VIH cho chế độ ghi nhiều byte hoặc VIL cho chế độ ghi trang. Khi không kết nối, đầu vào MODE được đọc nội tại như VIH (chế độ ghi nhiều byte).

**Write Control (WC):** Chức năng Write Control chỉ được cung cấp cho các phiên bản ST24W02 và ST25W02 ở chân 7. Chức năng này hữu ích để bảo vệ nội dung của bộ nhớ khỏi bất kỳ chu kỳ xóa/ghi lỗi nào. Tín hiệu Write Control được sử dụng để bật (WC = VIH) hoặc tắt (WC = VIL) bảo vệ ghi nội tại. Khi không kết nối, đầu vào WC nhận giá trị là VIL và khu vực bộ nhớ không được bảo vệ khi ghi.

* 1. Các chế độ hoạt động

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Chế độ | R bit | Chế độ | Bytes | Trình tự thực hiện |
| Đọc địa chỉ hiện tại | ‘1’ | X | 1 | START,Chọn chip,R=’1’ |
| Đọc địa chỉ bất kì | ‘0’ | X | 1 | START,Chọn chip,R=’0’,Địa chỉ |
| ‘1’ |
| Đọc liên tục | ‘1’ | X | 1 đến 256 | reSTART,Chọn chip,R=’1’ |
| Viết một Byte | ‘0’ | X | 1 | START,Chọn chip,R=’0’ |
| Viết nhiều Byte | ‘0’ | Vih | 4 | START,Chọn chip,R=’0’ |
| Viết một trang | ‘0’ | ViL | 8 | START,Chọn chip,R=’0’ |

**I2C Bus Background:** ST24/25x02 hỗ trợ giao thức I2C. Giao thức này định nghĩa bất kỳ thiết bị nào gửi dữ liệu vào bus làm bộ truyền, và bất kỳ thiết bị nào đọc dữ liệu làm bộ nhận. Thiết bị kiểm soát truyền dữ liệu được gọi là master và thiết bị còn lại được gọi là slave. Master luôn khởi tạo một truyền dữ liệu và cung cấp xung đồng bộ chuỗi cho đồng bộ hóa. ST24/25x02 luôn là thiết bị slave trong tất cả các truyền thông.

**Điều Kiện Bắt Đầu (Start Condition):** Bắt đầu được xác định bằng một chuyển đổi từ cao xuống thấp của dòng SDA trong khi xung đồng hồ SCL ổn định ở trạng thái cao. Một điều kiện bắt đầu phải đi trước bất kỳ lệnh truyền dữ liệu nào. Trừ khi trong chu kỳ lập trình, ST24/25x02 liên tục giám sát tín hiệu SDA và SCL để xác định một điều kiện bắt đầu và sẽ không phản ứng trừ khi có một điều kiện bắt đầu.

**Điều Kiện Kết Thúc (Stop Condition):** Kết thúc được xác định bằng một chuyển đổi từ thấp lên cao của dòng SDA trong khi xung đồng hồ SCL ổn định ở trạng thái cao. Một điều kiện kết thúc chấm dứt truyền thông giữa ST24/25x02 và master bus. Một điều kiện kết thúc sau lệnh đọc, chỉ sau một No Acknowledge, buộc thiết bị chuyển sang trạng thái chờ. Một điều kiện kết thúc sau lệnh ghi kích thích chu kỳ ghi nội bộ EEPROM.

**Bit Ghi Nhận (Acknowledge Bit - ACK):** Tín hiệu chấp nhận được sử dụng để chỉ định một truyền dữ liệu thành công. Bộ truyền bus, master hoặc slave, sẽ thả bus SDA sau khi gửi 8 bit dữ liệu. Trong khoảng thời gian xung đồng hồ thứ 9, bộ nhận kéo bus thấp SDA để xác nhận việc nhận 8 bit dữ liệu.

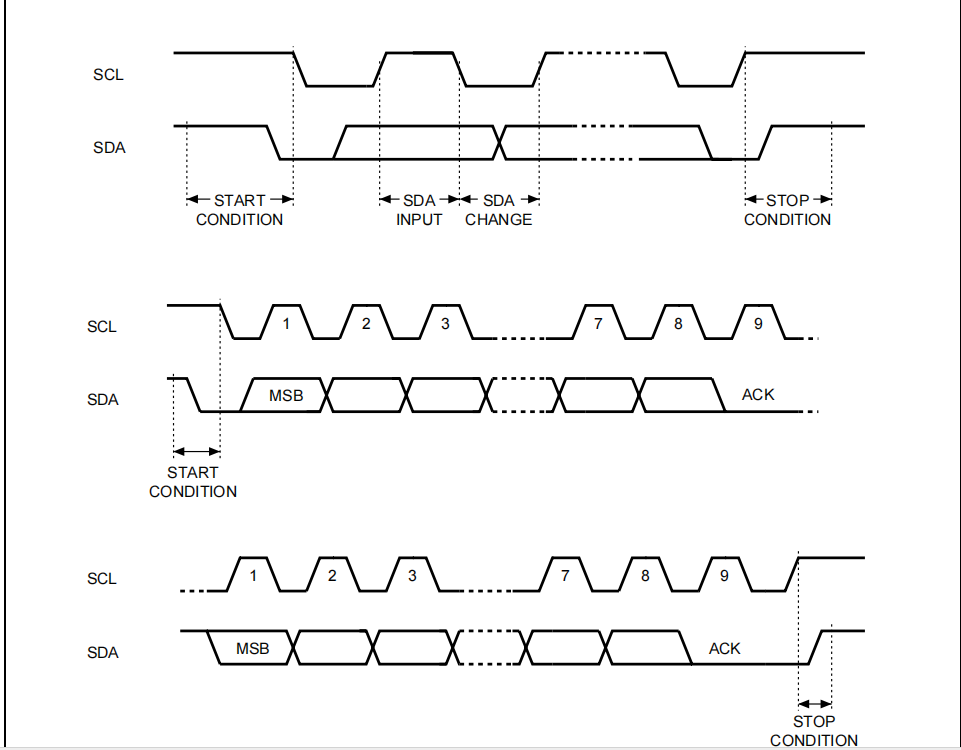
**Dữ Liệu Đầu Vào:** Trong quá trình đầu vào dữ liệu, ST24/25x02 lấy mẫu tín hiệu bus SDA vào lúc cạnh dương của xung đồng hồ SCL. Lưu ý rằng để hoạt động đúng của thiết bị, tín hiệu SDA phải ổn định trong khi xung đồng hồ giảm xuống và dữ liệu chỉ được thay đổi KHI và chỉ KHI đường SCL ở trạng thái thấp.

**Địa Chỉ Bộ Nhớ:** Để bắt đầu truyền thông giữa master bus và slave ST24/25x02, master phải khởi tạo một điều kiện bắt đầu. Sau đó, master gửi 8 bit (MSB trước) tương ứng với mã chọn thiết bị (7 bit) và một bit READ hoặc WRITE lên đường SDA.

* 1. Vận hành

-Bốn bit quan trọng nhất của mã chọn thiết bị định danh là người định loại thiết bị, tương ứng với định nghĩa bus I2C. Đối với những bộ nhớ này, 4 bit này cố định là 1010b. Ba bit tiếp theo xác định bộ nhớ cụ thể trên bus. Chúng được kết hợp với các tín hiệu kích hoạt chip E2, E1, E0. Do đó, có thể kết nối tối đa 8 bộ nhớ 2K trên cùng một bus, cung cấp tổng dung lượng bộ nhớ là 16K bit. Sau một điều kiện bắt đầu, bất kỳ bộ nhớ nào trên bus đều xác định mã thiết bị và so sánh 3 bit tiếp theo với các đầu vào kích hoạt chip E2, E1, E0 của nó.

-Bit thứ 8 được gửi là bit đọc hoặc ghi (RW), bit này được đặt là '1' cho các hoạt động đọc và '0' cho các hoạt động ghi. Nếu tìm thấy sự phù hợp, bộ nhớ tương ứng sẽ xác nhận việc định danh trên bus SDA trong khoảng thời gian bit thứ 9.



Hình 5 I2C Bus Protocol

1. Giải thích code

#include<reg51.h>

#include"i2c.h"

#define DIG P0

sbit LSA=P2^2;

sbit LSB=P2^3;

sbit LSC=P2^4;

sbit K1=P3^1;

sbit K2=P3^0;

sbit K3=P3^2;

sbit K4=P3^3;

void At24c02Write(unsigned char ,unsigned char );

unsigned char At24c02Read(unsigned char );

void Delay1ms();

void Timer0Configuration();

unsigned char code DIG\_CODE[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

unsigned char Num=0;

unsigned int disp[8]={0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f};

void main()

{

unsigned int num0=0,num1=0,n;

Timer0Configuration();

while(1)

{ // Nút K1: Ghi giá trị vào EEPROM

if(K1==0) // ghi giá tri vào eeprom

{

Delay1ms();

if(K1==0)

At24c02Write(2,num0);

// Đợi nút K3 được nhả ra

while((n<200)&&(K3==0))

{

n++;

Delay1ms();

}

n=0;

n=0;

}

// Nút K2: Đọc giá trị từ EEPROM

if(K2==0) // doc giá tri tu eeprom

{

Delay1ms();

if(K2==0)

num0=At24c02Read(2);

// Đợi nút K3 được nhả ra

while((n<200)&&(K3==0))

{

n++;

Delay1ms();

}

n=0;

}

// Nút K3: Tăng giá trị

if(K3==0)

{

Delay1ms();

if(K3==0)

num0++;

// Đợi nút K3 được nhả ra

while((n<200)&&(K3==0))

{

n++;

Delay1ms();

}

n=0;

if(num0==256)

num0=0;

}

// Nút K4: Đặt lại giá trị

if(K4==0)

{

Delay1ms();

if(K4==0)

num0=0;

// Đợi nút K3 được nhả ra

while((n<200)&&(K3==0))

{

n++;

Delay1ms();

}

n=0;

}

// Cập nhật giá trị hiển thị trên đèn 7 đoạn

disp[0]=DIG\_CODE[num1/1000];

disp[1]=DIG\_CODE[num1%1000/100];

disp[2]=DIG\_CODE[num1%1000%100/10];

disp[3]=DIG\_CODE[num1%1000%100%10];

disp[4]=DIG\_CODE[num0/1000];

disp[5]=DIG\_CODE[num0%1000/100];

disp[6]=DIG\_CODE[num0%1000%100/10];

disp[7]=DIG\_CODE[num0%1000%100%10];

}

}

void Timer0Configuration() //Cấu hình Timer 0: Timer 0 được cấu hình để tạo ra ngắt mỗi 1ms.//

{

TMOD=0X02;

TH0=0X9C;

TL0=0X9C;

ET0=1;

EA=1;

TR0=1;

}

void Delay1ms() //Hàm trễ 1ms:

Hàm này được sử dụng để tạo thời gian trễ 1ms.//

{

unsigned char a,b,c;

for(c=1;c>0;c--)

for(b=142;b>0;b--)

for(a=2;a>0;a--);

}

//Hàm ghi vào EEPROM:

//Hàm này sử dụng giao thức I2C để gửi địa chỉ, địa chỉ trong EEPROM, và dữ liệu cần ghi.

void At24c02Write(unsigned char addr,unsigned char dat)

{

I2cStart();

I2cSendByte(0xa2); // ten eeprom

I2cSendByte(addr); //dia chi trong eeprom

I2cSendByte(dat); //·data eeprom

I2cStop();

}

unsigned char At24c02Read(unsigned char addr)

{

unsigned char num;

I2cStart();

I2cSendByte(0xa2);

I2cSendByte(addr);

I2cStart();

I2cSendByte(0xa3);

num=I2cReadByte();

I2cStop();

return num;

}

//\*Ngắt hiển thị số trên đèn 7 đoạn:

Ngắt này được kích hoạt bởi Timer 0 mỗi 1ms.

Các chân LSA, LSB, LSC được thiết lập để chọn một trong tám đèn 7 đoạn.

Dữ liệu từ mảng disp được hiển thị lên đèn 7 đoạn thông qua DIG.

Biến Num được sử dụng để lựa chọn đèn 7 đoạn nào sẽ được hiển thị. Sau mỗi lần ngắt, Num tăng lên và nếu vượt quá 7, nó sẽ quay lại 0.\*/

void DigDisplay() interrupt 1

{

// TH0=0X9c;

// TL0=0X00;

// DIG=0;

P1 = disp[7];

switch(Num)

{

case(7):

LSA=0;LSB=0;LSC=0; break;

case(6):

LSA=1;LSB=0;LSC=0; break;

case(5):

LSA=0;LSB=1;LSC=0; break;

case(4):

LSA=1;LSB=1;LSC=0; break;

case(3):

LSA=0;LSB=0;LSC=1; break;

case(2):

LSA=1;LSB=0;LSC=1; break;

case(1):

LSA=0;LSB=1;LSC=1; break;

case(0):

LSA=1;LSB=1;LSC=1; break;

}

DIG=disp[Num];

Num++;

if(Num>7)

Num=0;

}

**Chú thích chung:** Chương trình chính thực hiện các thao tác ghi và đọc từ EEPROM thông qua các nút K1 và K2, tăng giá trị thông qua nút K3, và đặt lại giá trị thông qua nút K4. Giá trị được hiển thị trên đèn 7 đoạn.

**Nút K1:**

Khi nhấn K1, chương trình gọi hàm At24c02Write để ghi giá trị num0 vào địa chỉ 2 của EEPROM.

Sau đó, chương trình đợi nút K3 được nhả ra trước khi tiếp tục.

**Nút K2:**

Khi nhấn K2, chương trình gọi hàm At24c02Read để đọc giá trị từ địa chỉ 2 của EEPROM và lưu vào num0.

Chương trình đợi nút K3 được nhả ra trước khi tiếp tục.

**Nút K3:**

Khi nhấn K3, giá trị num0 được tăng lên 1.

Chương trình đợi nút K3 được nhả ra trước khi tiếp tục. Nếu giữ nút K3, giá trị sẽ tiếp tục tăng.

**Nút K4:**

Khi nhấn K4, giá trị num0 được đặt lại về 0.

Chương trình đợi nút K3 được nhả ra trước khi tiếp tục.

**Cập nhật hiển thị:**

Các giá trị num0 và num1 được chuyển thành các đoạn hiển thị và được lưu vào mảng disp để hiển thị lên đèn 7 đoạn.

Giải thích code i2c

#ifndef \_\_I2C\_H\_

#define \_\_I2C\_H\_

#include<reg51.h>

sbit SCL=P2^1;

sbit SDA=P2^0;

void I2cStart(); //Tạo dấu Start trên giao thức I2C.

void I2cStop(); //Tạo dấu Stop trên giao thức I2C.

unsigned char I2cSendByte(unsigned char dat); //Gửi một byte dữ liệu thông qua giao thức I2C.

unsigned char I2cReadByte(); //Đọc một byte dữ liệu từ giao thức I2C

void I2cReadRespon(); //Gửi một tín hiệu xác nhận sau khi đọc xong một byte dữ liệu

#endif

Giải thích code i2c.h

#include"i2c.h"

void Delay10us() // deplay 10us // Hàm đợi 10 micro giây

{

unsigned char a,b;

for(b=1;b>0;b--)

for(a=2;a>0;a--);

}

void I2cStart() // Hàm I2c Start // Hàm bắt đầu truyền thông qua I2C (Start condition)

{

SDA=1; // Dữ liệu tại chân SDA là 1

Delay10us(); // Đợi 10 micro giây

SCL=1; // Tăng cường xung clock

Delay10us(); // Đợi 10 micro giây

SDA=0; // Bắt đầu truyền thông qua I2C (Start condition)

Delay10us(); // Đợi 10 micro giây

SCL=0; // Giảm xung clock

Delay10us(); // Đợi 10 micro giây

}

}

void I2cStop() // Đợi 10 micro giây

{

SDA=0; // Dữ liệu tại chân SDA là 0

Delay10us(); // Đợi 10 micro giây

SCL=1; // Tăng cường xung clock

Delay10us(); // Đợi 10 micro giây

SDA=1; // Kết thúc truyền thông qua I2C (Stop condition)

Delay10us(); // Đợi 10 micro giây

}

}

unsigned char I2cSendByte(unsigned char dat) // Hàm gửi byte thông qua I2C

{

unsigned char a=0,b=0;

for(a=0;a<8;a++)

{

SDA=dat>>7; // Chuyển bit đầu tiên của byte vào chân SDA

dat=dat<<1; // Dịch trái byte để chuẩn bị cho bit tiếp theo

Delay10us(); // Đợi 10 micro giây

SCL=1; // Tăng cường xung clock

Delay10us(); // Đợi 10 micro giây

SCL=0; // Giảm xung clock

Delay10us(); // Đợi 10 micro giây

}

SDA=1; // Kết thúc truyền thông qua I2C (Stop condition)

Delay10us(); // Đợi 10 micro giây

SCL=1; // Tăng cường xung clock

while(SDA) // Kiểm tra sự xác nhận từ thiết bị nếu cần

{

b++;

if(b>200) // Thời gian chờ xác nhận, tránh lặp vô hạn

{

SCL=0; // Giảm xung clock

Delay10us(); // Đợi 10 micro giây

return 0; // Trả về 0 nếu không có xác nhận

}

}

SCL=0; // Giảm xung clock

Delay10us(); // Đợi 10 micro giây

return 1;

}

unsigned char I2cReadByte() // Hàm đọc byte thông qua I2C

{

unsigned char a=0,dat=0;

SDA=1; // Dữ liệu tại chân SDA là 1

Delay10us(); // Đợi 10 micro giây

for(a=0;a<8;a++)

{

SCL=1; // Tăng cường xung clock

Delay10us(); // Đợi 10 micro giây

dat<<=1; // Dịch trái byte để chuẩn bị cho bit tiếp theo

dat|=SDA; // Gán bit mới từ chân SDA vào byte đọc được

Delay10us(); // Đợi 10 micro giây

SCL=0; // Giảm xung clock

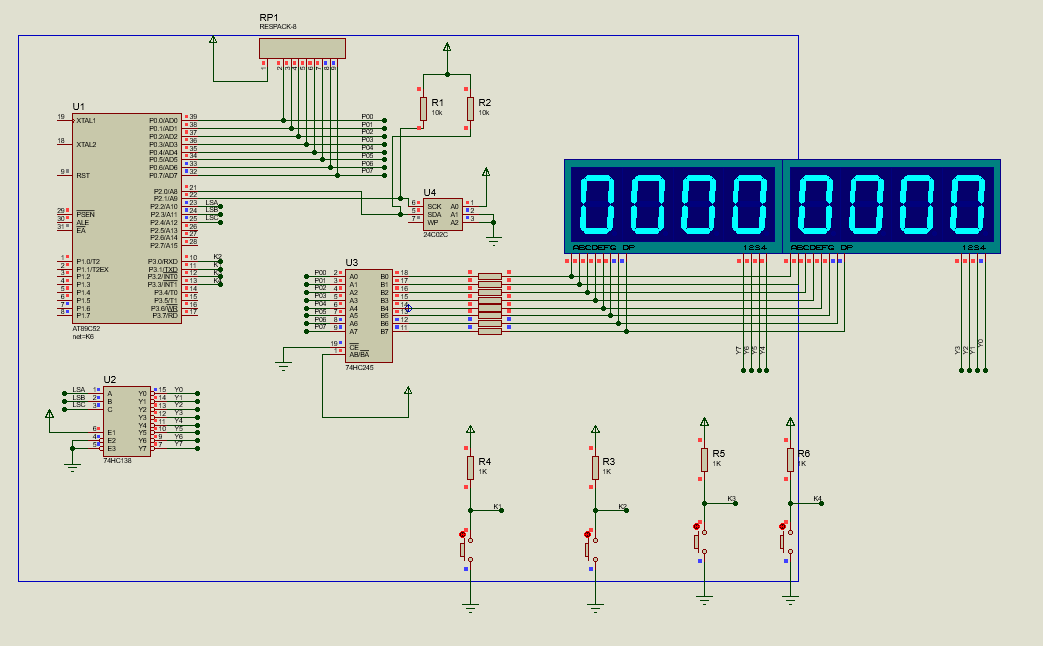
Delay10us(); // Đợi 10 micro giây

}

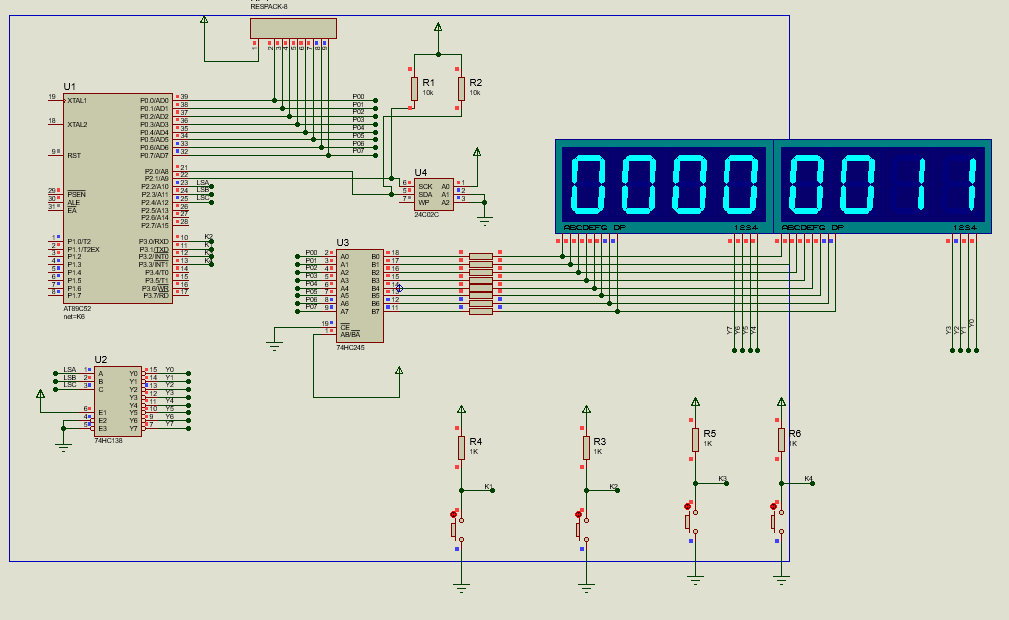
return dat; // Trả về byte đọc được

}

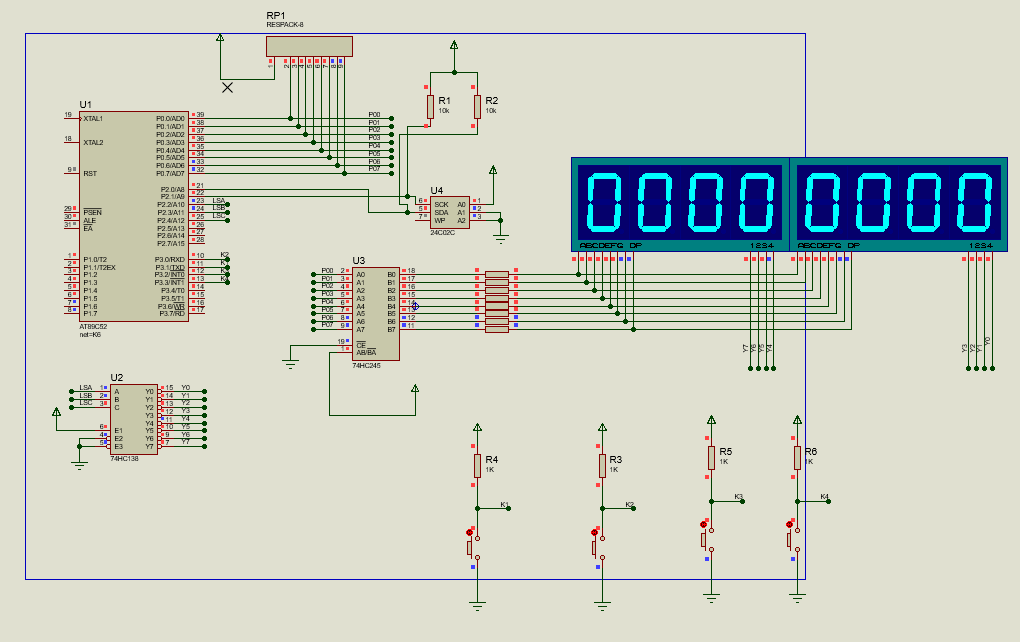
1. Proteus



Khi nút K3 được nhấn giá trị sẽ tăng lên 1 , khi nhấn rút K1 giá trị sẽ được lưu vào epproom.



Khi nút K4 được nhấn thì giá trị sẽ được reset về 0 và hiển thị trên led



Khi k2 được nhấn thì giá trị được đọc từ epproom sẽ được hiển thị

