**TRƯỜNG ĐẠI HỌC VINH**

VIỆN KỸ THUẬT CÔNG NGHỆ

**=====\*\*\*=====**



**BÀI TIỂU LUẬN**

**THIẾT KẾ HỆ THỐNG NHÚNG**

Giảng viên: TS. Mai Thế Anh

ThS.NCS. Lê Văn Chương

Họ và tên: Nguyễn Xuân Nam

SBD: 26

Ngày sinh: 01/02/1998

MSSV: 1755252021600054

Lớp: 58K – Kỹ thuật ĐK & TĐH

Nghệ An, 2021

***Tiểu luận: THIẾT KẾ HỆ THỐNG NHÚNG***

**Câu 1. Sử dụng vi điều khiển AT89C52, mô phỏng trên phần mềm Proteus và lập trình thực hiện các nhiệm vụ sau:**

**a.Hiển thị số 00 lên 2 LED 7 thanh nối vào cổng P2 theo phương pháp quét LED;**

**+Code:**

#include <REGX52.H>

charso[]= {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x89,0x06,0xC7,0x40};

#define led1 P2\_0

#define led2 P2\_1

#define sang 0

#define tat 1

char i;

void delay(int time){

while(time--);

}

void main(){

led1 = led2 = tat;

while (1){

led1 = sang;

P0 = so[0];

delay(100);

led1 = tat;

led2 = sang;

P0 = so[0];

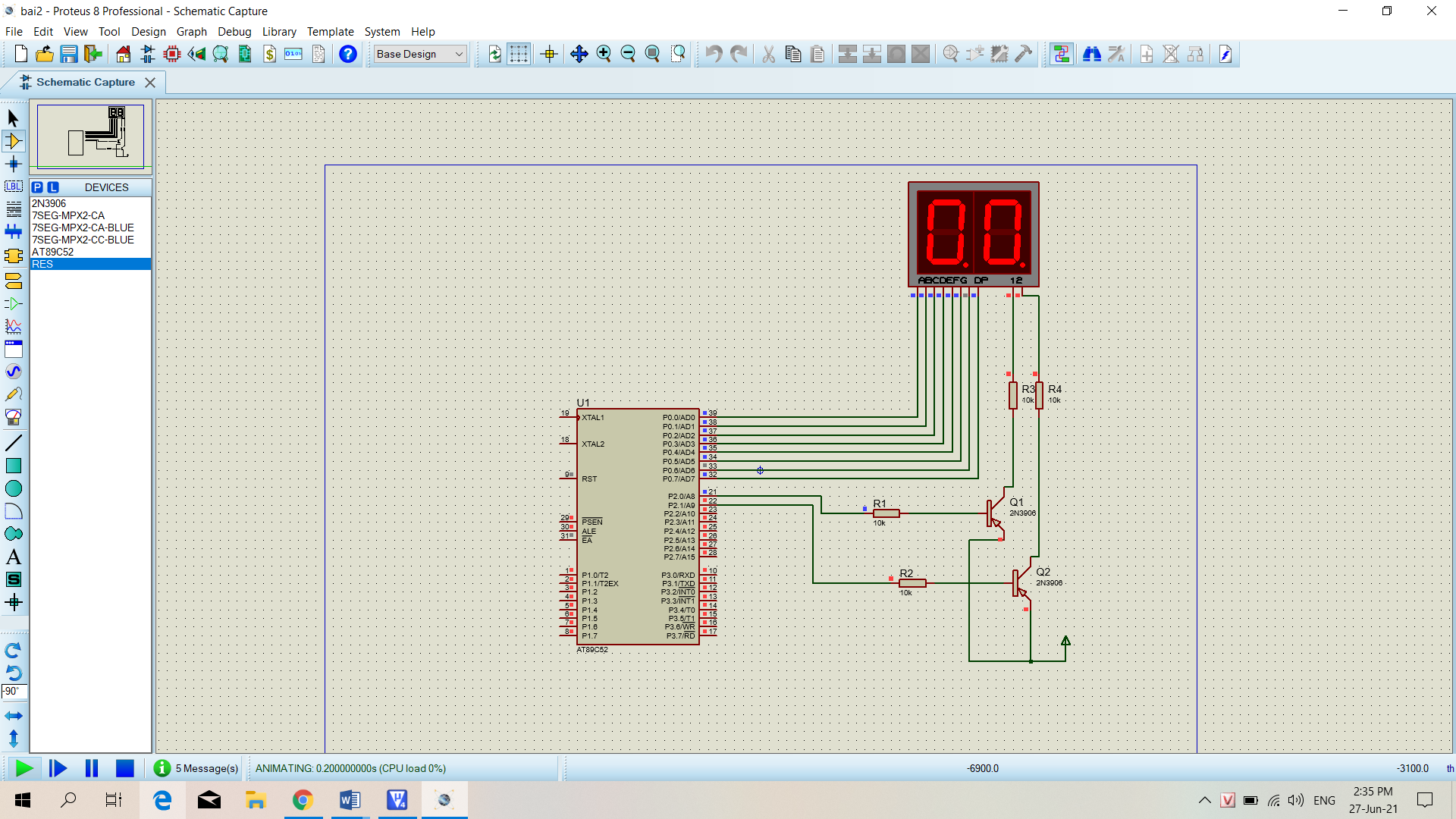
delay(100);

led2 = tat;

}

}

+**Mô phỏng trên phần mềm Proteus**



**b.Tăng số đếm sau mỗi 500ms, nếu số đếm bằng “SBD+20” thì dừng lại (sử dụng timer để định thời gian).**

**+CODE**

#include <REGX52.H>

char so[] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x89,0x06,0xC7,0x40};

char i;

int dem;

unsigned char chuc, donvi;

#define led1 P2\_0

#define led2 P2\_1

#define sang 0

#define tat 1

void delay(int time){

while(time--);

}

void main(){

led1 = led2 = tat;

while (1){

for (dem=0;dem<=21;dem++)

{

chuc = dem/10;

donvi = dem%10;

for (i=0; i<=10; i++)

{

led1 = sang;

P0 = so[chuc];

delay(500);

led1 = tat;

led2 = sang;

P0 = so[donvi];

delay(500);

led2 = tat;

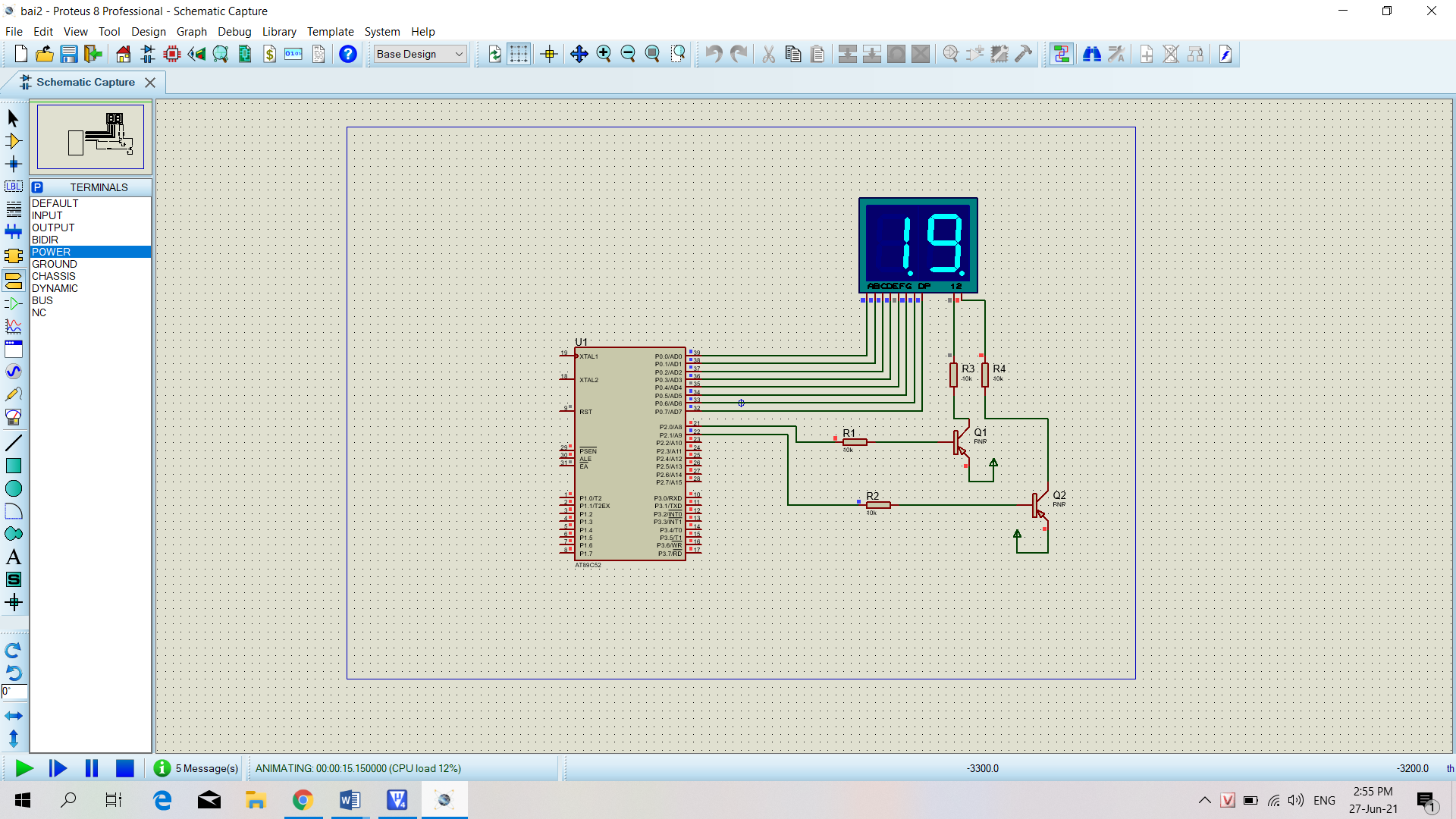
}

}

}

}

**+Mô phỏng trên Proteus**



**Câu 2. Sử dụng vi điều khiển AT89C52, mô phỏng trên phần mềm Proteus và lập trình thực hiện các nhiệm vụ sau:**

**a.Cấu hình ngắt ngoài INT0 ở chế độ ngắt sườn xuống;**

* **Code**

#include <REGX51.H>

#define LED7\_DATA P2

unsigned char ma\_led\_7[10] = {0XC0, 0XF9, 0XA4 ,0X99, 0X92, 0X82, 0XF8, 0X80, 0X90};

signed char x;

void main(void)

{

EX0 = 1;

EX1 = 1;

IT0 = 1;

IT1 = 1;

EA = 1;

while(1)

{

LED7\_DATA = ma\_led\_7[x];

}

}

void ISR\_EXO(void) interrupt 0

{

x++;

if(x>9)

x=0;

}

void ISR\_EX1(void) interrupt 2

{

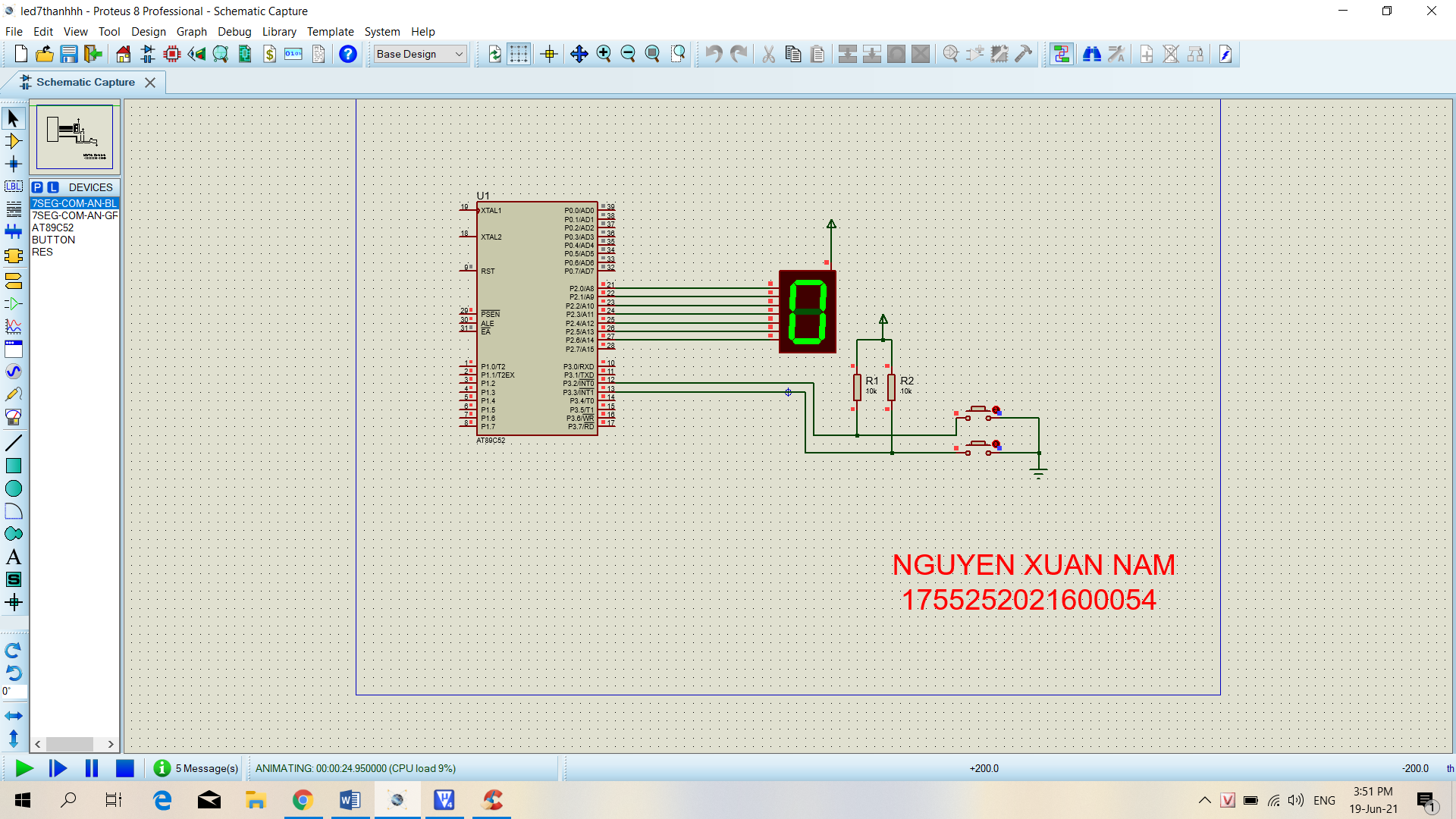
x--;

if(x<0)

x=9;

}

**+Mô phỏng trên Proteus**



**b.Đếm số lần nút bấm nút nối vào chân INT0 được bấm, hiển thị kết quả lên 2 LED 7 thanh nối vào cổng P2 theo phương pháp quét LED (nếu số lần bấm bằng “SBD+26” thì quay về 00).**

**+Code**

#include <REGX51.H>

sbit nutbam=P1^0;

int so[]={0xC0,0xF9, 0xA4, 0xB0, 0x99, 0x82, 0xF8, 0x80, 0x90};

int dem, k;

void delay(unsigned int time)

{

unsigned int x,y;

for(x=0;x<time;x++)

{

for(y=0; y<123;y++);

}

}

void hienthi(int giaitri)

{

for(k=0; k<10;k++)

{

P3=0x01;

P2=so[giaitri/10];

delay(10);

P3=0x02;

P2=so[giaitri%10];

delay(10);

}

}

void main()

{

P1=P3=0x00;

P2=0xff;

while(1)

{

hienthi(dem);

while(nutbam==1)

{

dem++;

if(dem>26) dem=0;

while(nutbam==1)

{

hienthi(dem);

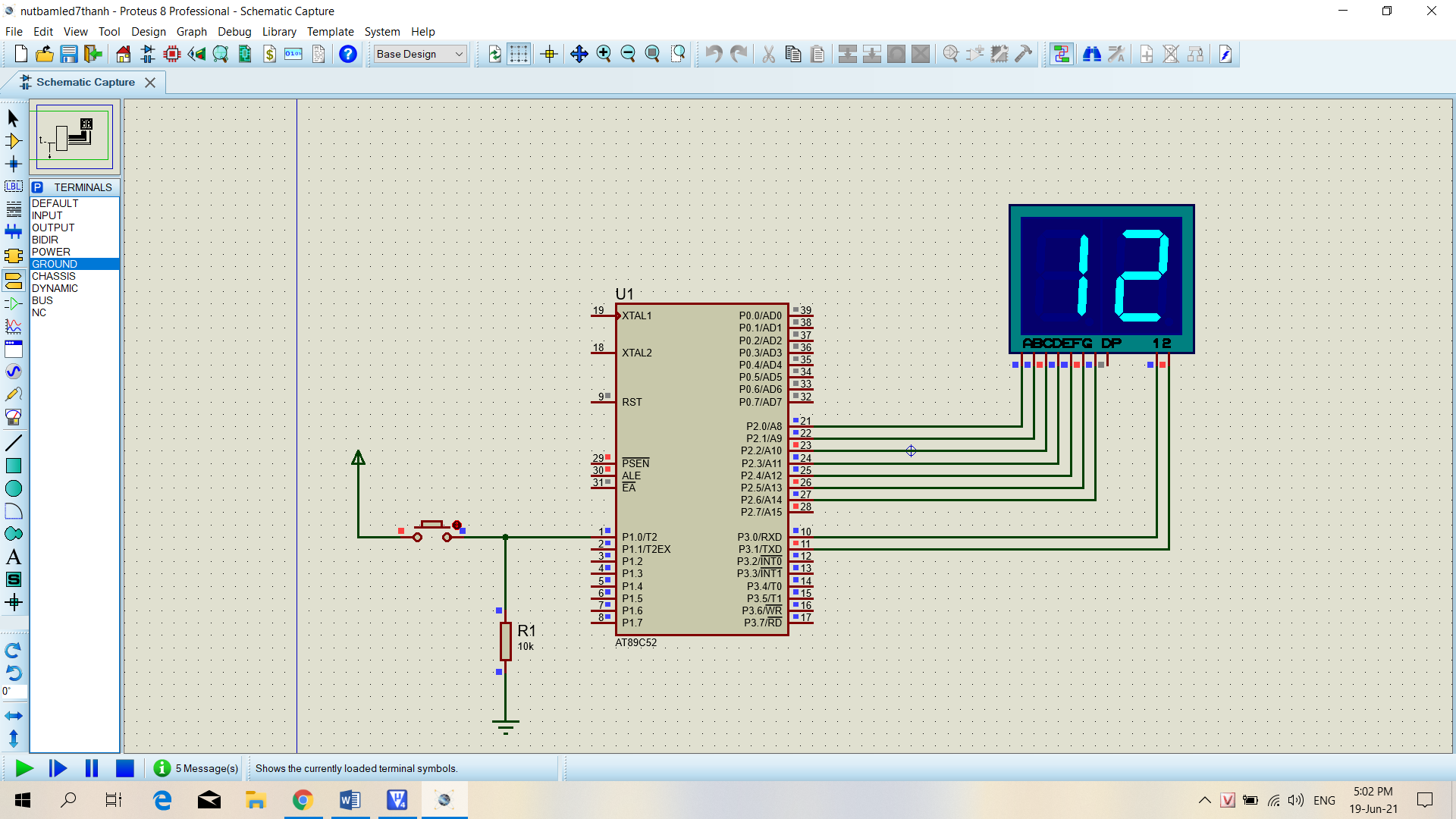
}

}

}

}

**+Mô phỏng trên Proteus**



**Câu 3. Sử dụng vi điều khiển AT89C52, thực hiện các nhiệm vụ sau:**

**a.Vẽ sơ đồ mô phỏng trên Proteus ghép nối với LCD theo chế độ 4bit, hiển thị họ tên, mã số sinh viên lên LCD.**

**+Code**

unsigned short kp, cnt, oldstate = 0;

char txt[6];

char keypadPort at P0;

sbit LCD\_RS at P2\_0\_bit;

sbit LCD\_EN at P2\_1\_bit;

sbit LCD\_D4 at P2\_2\_bit;

sbit LCD\_D5 at P2\_3\_bit;

sbit LCD\_D6 at P2\_4\_bit;

sbit LCD\_D7 at P2\_5\_bit;

void main() {

cnt = 0;

Keypad\_Init();

Lcd\_Init();

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Cmd(\_LCD\_CURSOR\_OFF);

Lcd\_Out(1, 1, "NGUYEN XUAN NAM");

Lcd\_Out(2, 1, "175525202100054");

do{

kp = 0;

do

kp = Keypad\_Key\_Click();

while(!kp);

switch(kp){

case 1: kp = 49; break;

case 2: kp = 50; break;

case 3: kp = 51; break;

case 4: kp = 65; break;

case 5: kp = 52; break;

case 6: kp = 53; break;

case 7: kp = 54; break;

case 8: kp = 66; break;

case 9: kp = 55; break;

case 10: kp = 56; break;

case 11: kp = 57; break;

case 12: kp = 67; break;

case 13: kp = 42; break;

case 14: kp = 48; break;

case 15: kp = 35; break;

case 16: kp = 68; break;

}

if(kp = oldstate)

{

cnt = 1;

oldstate = kp;

}

else{

cnt++;

}

Lcd\_Chr(1, 10, kp);

if(cnt == 255){

cnt = 0;

Lcd\_Out(2, 10, " ");

}

WordToStr(cnt, txt);

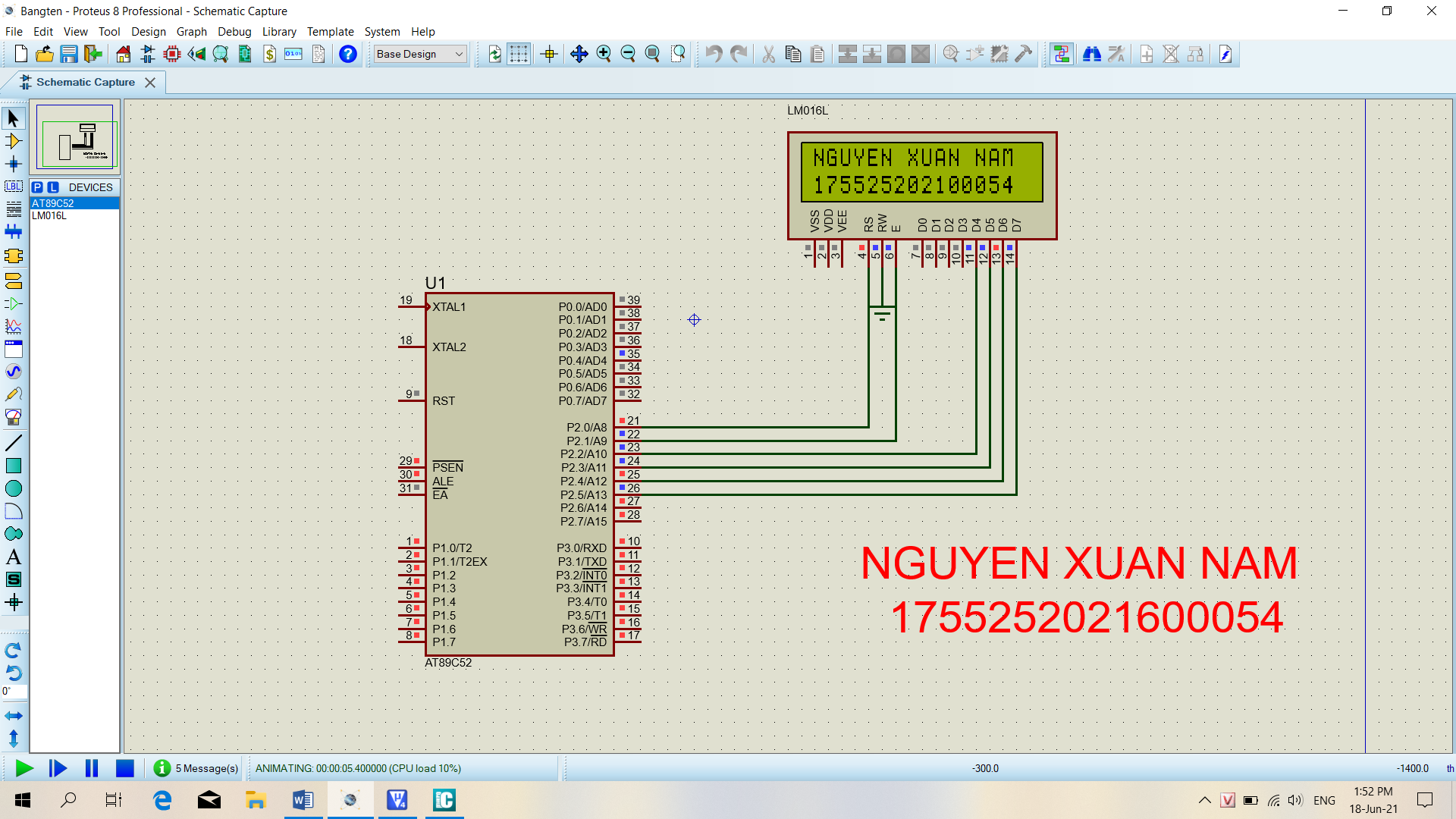
Lcd\_Out(2, 10, txt);

}

while(1);

}

**+Mô phỏng trên Proteus**



**b.Vẽ sơ đồ mô phỏng trên Proteus, lập trình hiển thị “Họ tên và mã số sinh viên” qua chuẩn truyền thông UART;**

**+Code**

char rd;

void main() {

UART1\_Init(4800);

Delay\_ms(100);

UART1\_Write\_Text("NGUYEN XUAN NAM,MSSV:1755252021600054");

while(1)

{

if(UART1\_Data\_Ready()==1)

{

rd=UART1\_Read();

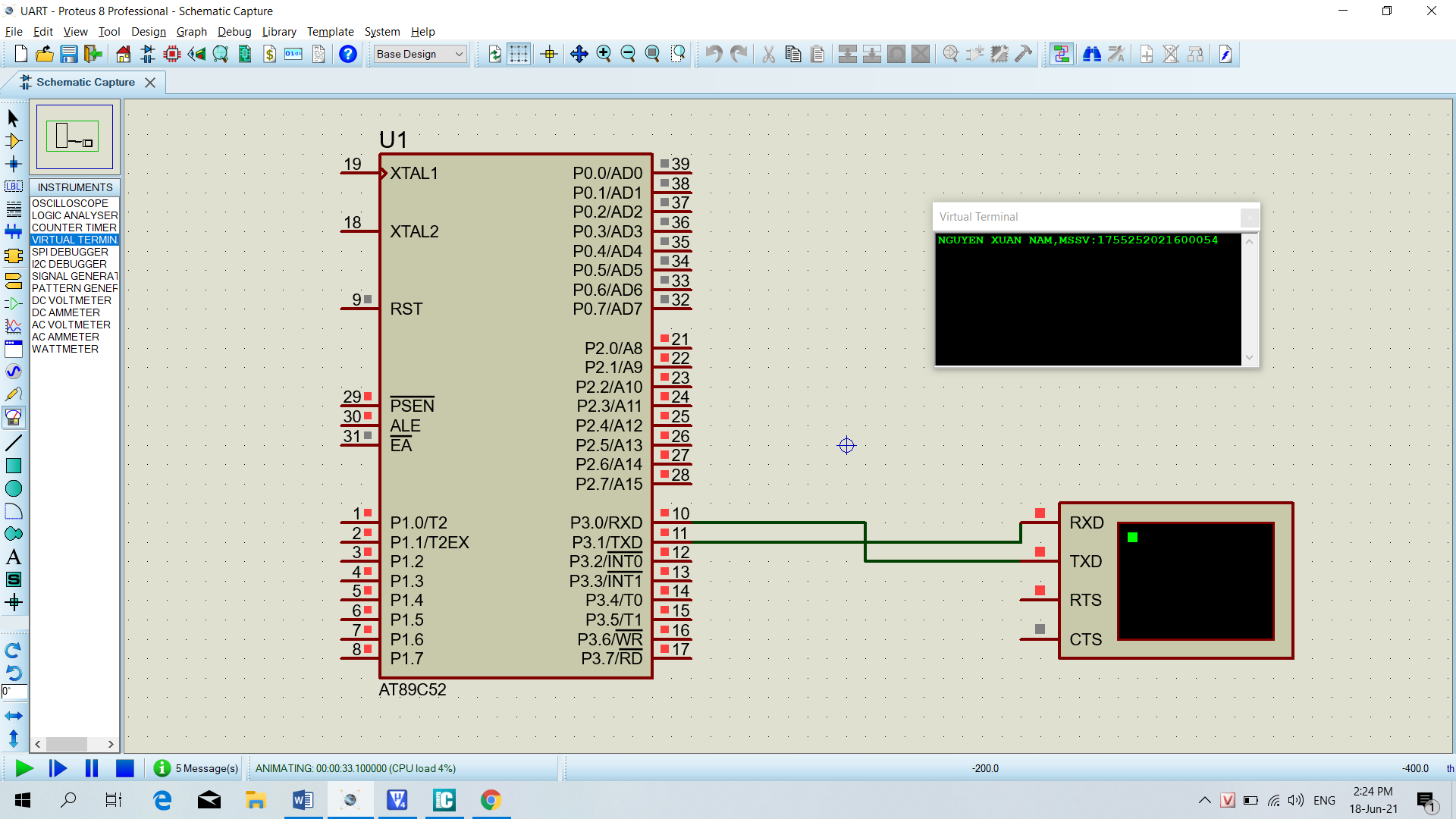
UART1\_Write(rd);

}

}

}

**+Mô phỏng trên proteus**



**Câu 4. Sử dụng vi điều khiển AT89C52, vẽ sơ đồ mô phỏng trên Proteus ghép nối với Led D1 qua cổng P1.2, BUTTON B1 qua cổng P1.3. Sử dụng hệ điều hành RTX51 lập trình ngắt USART, tast BUTTON, tast LED. Thực hiện gửi “signal” từ ngắt USART và task BUTTON đến tast LED. Task LED thực hiện đảo trạng thái của Led D1 khi nhận được tín hiệu task khác gửi tới.**

**+Code**

#include <REGX52.H>

#include <RTX51TNY.H> //Su dung thu vien RTX51 Tiny Real-Time

#define INIT 0 //Dinh nghia INIT = 0

#define DO 1 //Dinh nghia DO = 1

#define BUTT 2 //Dinh nghia BUTTTON = 2

sbit LED\_DO = P1^0; //Dinh nghia chan LED\_DO

sbit BUTTON = P1^1; //Dinh nghia chan BUTTON

void USART(void) interrupt 4 //Ngat nhan USART

{

if(RI)//Flag nhan duoc ki tu

{ //Clear flag

RI=0; //Nhan ki tu

isr\_send\_signal(DO); //Gui signal cho task DO

}

}

//================================

void Startup(void) \_task\_ INIT

{

SCON=0x52; //USART che do 1

TMOD=0x21; //Timer 1 mode 2

TH1=TL1=-3; //baudrate 9600

TR1=1;

IE=0x90; //Ngat USART

os\_create\_task (DO); //Tao Task\_Led\_Do

os\_create\_task (BUTT); //Tao Task BUTTON

os\_delete\_task (INIT); //Xoa Task hien tai (Task 0)

}

void Task\_Led\_Do(void) \_task\_ DO

{

while(1)

{

os\_wait2(K\_SIG ,50); //Cho signal voi time out 50 ticks

LED\_DO ^= 1; //Tao trang thai Led Do

}

}

void Task\_BUTTON(void) \_task\_ BUTT

{

while(1)

{

if(BUTTON == 0) //Nhan nut nhan = 0

{

os\_send\_signal(DO); //Gui signal cho task DO

while(BUTTON==0); //Cho nut nhan = 1(Chong nhieu)

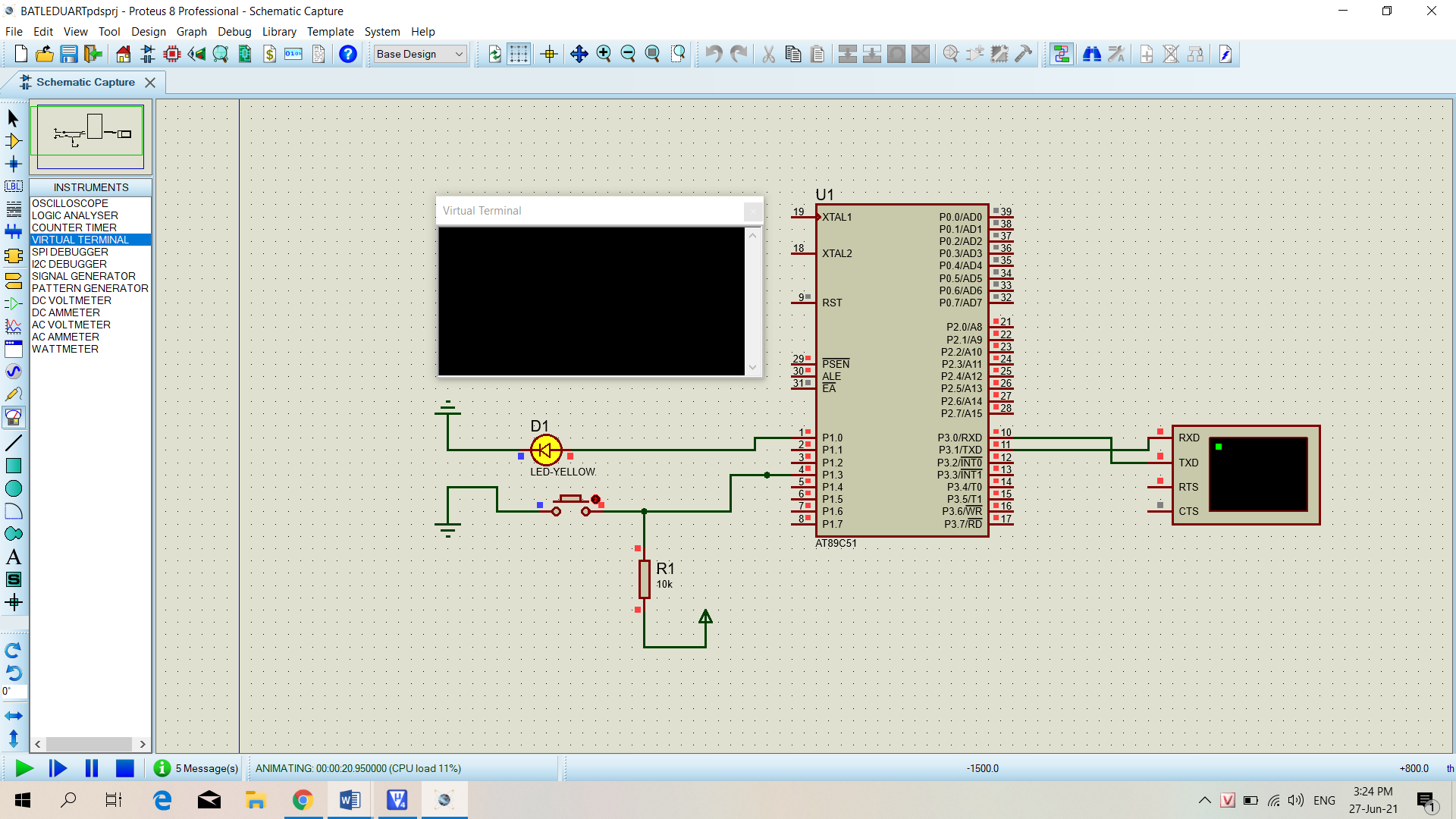
}

os\_wait2(K\_TMO, 10); //Cho 10 ticks = 100ms

}

}

**+Mô phỏng trên Proteus**

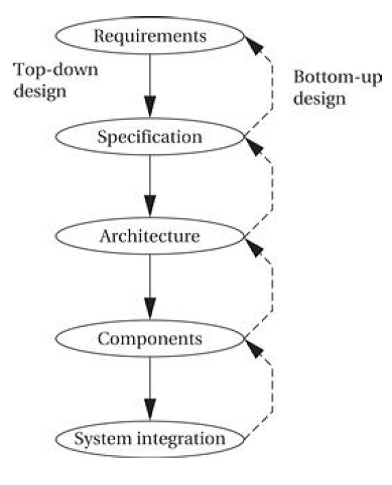


**Câu 5. Hãy trình bày:**

1. **Trình bày quy trình thiết kế hệ thống nhúng sử dụng vi điều khiển.**

Một phướng pháp thiết kế rất quan trọng bởi ba lý do như sau. Đầu tiên, nó cho phép chúng ta giữ một bảng ghi chú về một thiết kế để đảm bảo rằng chúng ta đã làm mọi thứ chúng ta cần làm, chẳng hạn như có cần tối ưu hóa hiệu năng hay không hoặc khi nào thực hiện kiểm tra chức năng của thiết kế. Thứ hai, nó cho phép chúng ta phát triển các công cụ thiết kế được hỗ trợ bởi máy tính. Phát triển một công cụ duy nhất để hỗ trợ toàn bộ quá trình thiết kế một hệ thống máy tính nhúng sẽ là một nhiệm vụ khó khăn, nhưng bằng cách chia quy trình thiết kế thành các bước nhỏ có thể quản lý, chúng ta có thể tạo ra các công cụ tự động hóa (hoặc ít nhất là bán tự động hóa) quá trình thiết kế cho mỗi bước trong quy trình. Thứ ba, một phương pháp thiết kế giúp các thành viên của nhóm thiết kế trao đổi với nhau dễ dàng hơn. Bằng cách xác định một quy trình thiết kế tổng thể, các thành viên trong nhóm có thể dễ dàng hiểu những gì họ phải làm, những gì họ sẽ nhận được từ các thành viên khác trong nhóm vào những thời điểm nhất định và những gì họ phải bàn giao khi hoàn thành các bước được giao. Bởi vì phần lớn các hệ thống máy tính nhúng được thiết kế và thực hiện bởi các nhóm, sự phối hợp nhịp nhàng giữa các nhóm thể hiện vai trò quan trọng nhất trong phương pháp thiết kế.

Tóm tắt các bước chính trong quy trình thiết kế hệ thống nhúng. Trong luồng thiết kế từ trên xuống, chúng ta bắt đầu với việc đưa ra các yêu cầu thiết kế (Requirements) của hệ thống. Trong bước tiếp theo, đặc tả kỹ thuật (specification), chúng ta tạo ra một bản mô tả chi tiết hơn về những gì chúng ta muốn thiết kế. Tuy nhiên, bản đặc tả chỉ nêu ra cách hệ thống hoạt động mà không phải cách nó được xây dựng thế nào. Các chi tiết bên trong của hệ thống bắt đầu hình thành khi chúng ta phát triển kiến trúc – một sơ đồ khối chỉ ra cấu trúc của hệ thống dưới dạng các khối chức năng. Khi chúng ta xác định các khối chức năng chính cấu thành lên hệ thống chúng ta có thể thiết kế các khối chức năng đó, bao gồm cả các khối chức năng phần mềm và bất kỳ khối chức năng phần cứng chuyên dụng nào mà chúng ta cần. Dựa trên các khối chức năng đó, cuối cùng chúng ta có thể tích hợp chúng lại với nhau để xây dựng một hệ thống hoàn chỉnh.



Nhưng các bước trong quy trình thiết chỉ là một trục mà chúng ta có thể xem xét quá trình thiết kế hệ thống máy tính nhúng. Bên cạnh đó chúng ta cũng cần xem xét thêm các mục tiêu chính của thiết kế:

* Chi phí sản xuất
* Hiệu năng (bao gồm tốc độ tính toán và thời hạn một tính toán phải hoàn thành)
* Sự tiêu thụ năng lượng.

Chúng ta cũng cần xem xét các nhiệm vụ mà chúng ta cần thực hiện ở mọi bước trong thiết kế, cụ thể như:

* Chúng ta phải phân tích thiết kế ở mỗi bước để xác định cách chúng ta có thể thỏa mãn các yêu cầu kỹ thuật trong bản đặc tả
* Sau đó chúng ta phải tinh chỉnh thiết kế để đi sâu vào thiết kế chi tiết
* Chúng ta cần kiểm thử thiết kế để đảm bảo rằng nó vẫn đáp ứng được tất cả các mục tiêu của hệ thống chẳng hạn như chi phí, tốc độ, …

1. **Hệ điều hành thời gian thực (RTOS). Ưu điểm, nhược điểm và ứng dụng của hệ điều hành thời gian thực trong thiết kế các hệ thống nhúng.**

Hiện nay có rất nhiều hệ điều hành thời gian thực ( RTOS) dành cho phát triển hệ thống nhúng, việc sử dụng còn phụ thuộc vào khả năng đáp ứng của vi của Vi điều khiển đặc biệt là RAM và tốc độ xung clock. Những hệ điều hành này được phát triển trên các nền ngôn ngữ lập trình khác nhau, phổ biến nhất là ngôn ngữ C, và cũng có thế là JAVA….Trong khuông khổ luận văn này chỉ xin được giới thiệu những hệ điệu hành thời gian thực dành cho những hệ thống nhỏ.Hiện tại có nhiều hệ điều hành thời gian thực dành cho các dòng vi điều khiển thông thường, từ Open Source đến có bản quyền.Open source phổ biến như FreeRTOS hay CooCox CoOS, ở đây có xin được giới thiệu về FreeRTOS vì tính phổ biến của nó cũng như đơn giản, dễ sử dụng nhưng vẫn đầy đủ các tính năng của một RTOS.FreeRTOS được nghiên cứu bởi Richard Barry với tên ban đầu là FRTOS07. Mục đích của FreeRTOS là portable( khả năng linh động) , open source ( mã nguồn mở) , mini kernel (là một hệ điều hành thời gian thực nhỏ) mà có thể được thao tác trong chế độ ưu tiên (Pre-emptive) cũng như phối hợp (Cooperative).  
\*FreeRTOS hiệ tại hỗ trợ lên đến 27 kiến trúc vi điều khiển khác nhau khác nhau :

+Altera Nios II

+ARM7, ARM9, Cortex M3

+AVR, PIC, 8051

+SH, H8S, PowerPC, x86

\*Các chức năng chính của FreeRTOS được giới thiệu như sau:

-*Thời gian thực*

Thời gian thực hoặc hoạt động đồng bộ

Trong chế độ đồng bộ thì việc đặt lịch không được thực hiện bởi 1 điểm xác định của timer mà được xử lý qua việc điều khiển từ task này đến task khác bởi cơ chế yielding.

-*Dynamic Scheduling*

Thời điểm xắp xếp lại lịch xảy ra sau 1 khoảng thời gian cụ thể (phụ thuộc vào tần số xung clock)

-*Giải thuật đặc lịch*

Là giải thuật có độ ưu tiên cao nhất trước tiên. Nếu có nhiều hơn 1 tast có cùng độ ưu tiên cao nhất thì các task sẽ được thực hiện theo kiểu round-robin

*-Giao tiếp giữa Tasks*Inter-process comm được thực hiện bằng queue, thì hầu hết thông tin được truyền bằng giá trị chứ không phải tham chiếu nên vấn đề ràng buộc vê bộ nhớ phải được xét đến.Queue read hoặc write từ bên trong ISR là : non-blocking.Queue read hoặc write với timeout=0 : non-blocking

-*Synchroniztion ( Đồng bộ):* FreeRTOS cho phép tạo ra và sử dụng binary semaphore. Bản thân Semaphore là 1 trường hợp đặc biệt của queue với chiều dài =1 và kích thước dữ liệu là =0. Vì điều này nên việc lấy và trả semaphore là các thao tác nguyên thủy, do đó các ngắt sẽ bị vô hiệu hóa và bộ đặt lịch phải bị treo để nhận 1 block trên queue này.

-*Tránh Blocking và Deadlock:*Trong FreeRTOS các task hoặc là non-blocking hoặc sẽ bị block với 1 khoảng thời gian cố định. Các task mà thức dậy khi hết timeout và vẫn không thể truy nhập tới 1 tài nguyên vì đôi khi lời gọi API tới 1 tài nguyên có thể trả về “không thành công” (failed)

*-Xử lý critical SectionCritical Section* :được xử lý bởi việc vô hiệu hóa tất cả các ngắt. Critical section bên trong mỗi task có thể được xếp chồng lên nhau và mỗi task sẽ ghi dấu lại bởi việc tăng giá trị xếp chồng của nó.

-*Treo bộ đặt lịch*:Khi dành riêng việc truy nhập tới MCU mà việc MCU bị yêu cầu là không gây nguy hiểm cho thao tác của ISR, bộ đặt lịch có thể bị treo. Việc treo này đảm bảo răng tiến trình hiện tại sẽ không được ưu tiên bởi 1 đặt lịch (scheduler)

-*Cấp phát bộ nhớ:*FreeRTOS cung cấp 3 mô hình heap. Mô hình đơn giản nhất là việc cấp phát cố định, phức tạp hơn là mô hình cho phép cấp phát và giải phóng bằng việc sử dụng giải thuật “best fit” và phức tạp nhất là heap\_3 cho phép sử dụng malloc và calloc()

*-Đảo ngược độ ưu tiên:* FreeRTOS không thực hiện bất kỳ kỹ thuật nâng cao nào với độ ưu tiên tiến trình như kế thừa độ ưu tiên hay xoay vòng độ ưu tiên để thực hiện đảo ngược độ ưu tiên

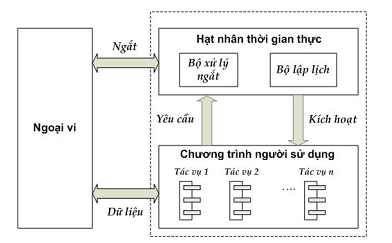
**Kiến trúc hệ điều hành thời gian thực (RTOS)?**

- Hệ điều hành thời gian thực – RealTime Operating Systems(RTOS), là phần mềm điều khiển chuyên dụng thường được dùng trong những ứng dụng điện toán nhúng có tài nguyên bộ nhớ hạn chế và yêu cầu ngặt nghèo về thời gian đáp ứng tức thời, tính sẵn sàng cao và khả năng tự kiểm soát một cách chính xác. Công nghệ đa nhiệm: mỗi quá trình có một không gian bộ nhớ riêng, các quá trình phải được chia nhỏ thành các Thread cùng chia sẻ không gian bộ nhớ. Các dịch vụ cung cấp bởi hạt nhân: tạo và kết thúc quá trình/tác vụ, truyền thông điệp giữa các quá trình, một số các dịch vụ cung cấp hỗ trợ việc thực thi liên quan đến điều khiển hệ thống. 

Khác biệt so với các kiến trúc khác:

* Quá trình điều khiển giữa các chu trình ngắt và mã nguồn thực thi tác vụ được quản lý bởi RTOS ( không cần dùng các biến chia sẻ)
* RTOS quyết định lập lịch các tác vụ thay vì dùng một vòng lặp chính để quyết định xem là tiếp theo là thực thi tác vụ gì hoặc ngắt gì
* RTOS có thể tạm dừng một tác vụ đang thực thi để thực thi tác vụ khác

Hệ thống điều hành với phần lõi là hạt nhân phải đảm nhiệm các tác vụ chính như sau:



* Xử lý ngắt

 Lưu trữ ngữ cảnh chương trình tại thời điểm xuất hiện ngắt

 Nhận dạng và lựa chọn đúng bộ xử lý và phục vụ dịch vụ ngắt

* Điều khiển quá trình

Tạo và kết thúc quá trình/tác vụ

Lập lịch và điều phối hoạt động hệ thống

Định thời

* Điều khiển ngoại vi

Xử lý ngắt

Khởi tạo giao tiếp vào ra

\***Ưu điểm:**

-Ưu điểm lớn của RTOS là xử lý nhanh chóng vì thế nó sẽ dành cho các thiết bị đòi hỏi khả năng xử lý có độ trễ thấp nhất có thể. Lợi ích nó đem lại bao gồm đa nhiệm tốt, ưu tiên các nhiệm vụ và quản lý chia sẻ các tài nguyên. Ngoài ra nó cũng không đòi hỏi nhiều về tài nguyên hay bộ nhớ RAM quá lớn.

Hệ điều hành RTOS được sử dụng phổ biến trong nhiều ngành công nghiệp, thiết bị chăm sóc sức khỏe IoT (như smartwatch) hay ngành hàng không

-RTOS là một kiến trúc tốt hơn các hệ điều hành nhúng khác vì nó có thể đáp ứng các yêu cầu ràng buộc về thời gian (ràng buộc thời gian cứng – không cho phép thời gian xử lý chậm và ràng buộc thời gian mềm – cho phép xử lý chậm trong một lân cận nào đó), chịu lỗi cao và cho phép xử lý đa nhiệm (định danh tiến trình và độ ưu tiên - thông qua một số phương thức: semaphores, mailbox, queue...).

- Thay đổi bất kỳ tác vụ nào trong Round Robin hoặc lập lịch theo hàng đợi đều có một nhược điểm là ảnh hưởng đến tổng thể toàn bộ các tác vụ.

- Thay đổi tới tác vụ có độ ưu tiên thấp hơn trong RTOS không ảnh hưởng đến thời gian đáp ứng của các tác vụ có độ ưu tiên cao hơn.

- RTOS được sử dụng rộng rãi và là giải pháp thật sự cần thiết cho các hệ thống yêu cầu ràng buộc về thời gian đáp ứng (ràng buộc thời gian cứng, mềm)

**\*Nhược điểm:**

- RTOS cần thêm một số thời gian để xử lý các thông tin về tác vụ trước và sau khi đưa nó vào xử lý trong CPU nên hiệu suất sử dụng bị ảnh hưởng (do yêu cầu tính ràng buộc về thời gian nên độ phức tạp cao và xử lý cần đảm bảo độ an toàn). Chi phí cao khi mua các sản phẩm thương mại.

**\*Các hệ thời gian thực điển hình**

Adeos, ART Linux, ChorusOS, eCos, ELinOS, FreeRTOS, iRmx, ITRON, Linux, LynxOS, MicroC/OS-II, Nucleus, OS-9, OSE, OSEK/VDX, pSOS, PikeOS, QNX, RedHawk, RSX-11, VxWorks, Windows CE, Xenomai

**\*Ứng dụng của hệ điều hành thời gian thực trong thiết kế các hệ thống nhúng**

-RTOS sử dụng trong những ứng dụng yêu cầu thời gian đáp ứng nhanh, chính xác về thời gian

-Chạy các dự án lớn đòi hỏi xử lý nhiều công việc nhưng vẫn phải đáp ứng được về mặt thời gian

-Các ứng dụng về viễn thông và IOT, các thiết bị liên quan đến y tế…