BÀI 4. CÁC BỘ ĐỊNH THỜI/ĐẾM

Chuẩn đầu ra của học phần:

L1: Giải thích cấu tạo, hoạt động của vi điều khiển, trình bày một số tài nguyên của PIC và lập trình được cho vi điều khiển

Mục tiêu bài học:

Bài học cung cấp kiến thức về các bộ định thời/đếm (timer/counter) trên PIC18F4520, cách lập trình sử dụng các bộ định thời/đếm.

Nội dung bài học trực tuyến:

(1) Khái quát về hoạt động định thời/đếm

(2) Cấu tạo và nguyên lý hoạt động của các bộ đếm

(3) Ứng dụng của bộ đếm

(4) Timer0

(5) Các bước lập trình sử dụng Timer0.

4.1. Khái quát về hoạt động định thời/đếm

Khái quát về hoạt động định thời/đếm

Trong các ứng dụng **định khoảng thời gian**, timer được lập trình sao cho sau một khoảng thời gian timer sẽ tràn. Khi tràn, cờ tràn của timer được đặt bằng 1. Căn cứ vào trạng thái của cờ tràn, người lập trình có thực hiện một thao tác điều khiển hoặc đo lường nào đó để tạo ra các ứng dụng đo lường/điều khiển theo thời gian.

Đếm sự kiện dùng để xác định số lần xảy ra của một sự kiện. Trong ứng dụng này người ta tìm cách quy các sự kiện thành sự chuyển mức từ '1' xuống '0' hoặc ngược lại trên các lối vào của các bộ timer. Sau mỗi sự chuyển mức, giá trị của timer sẽ thay đổi (tăng hoặc giảm đi 1 tùy theo chế độ đếm tiến hoặc lùi). Giá trị hiện thời của timer sẽ chính là số sự kiện đã xảy ra.

Cấu tạo và nguyên lý hoạt động của bộ định thời/đếm

Các bộ định thời (timer) được sử dụng rất rộng rãi trong các ứng dụng đo lường và điều khiển. Có thể coi một bộ định thời n bit là một bộ đếm n bit được tạo ra bởi n flip-flop mắc nối tiếp với nhau. Đầu vào của bộ định thời chính là đầu vào của flip-lop đầu tiên, đầu ra báo tràn (over flow) của bộ định thời phản ánh trạng thái tràn của nó. Đầu ra của các flip-flop phản ánh giá trị hiện thời của bộ đếm. Tuỳ thuộc vào ứng dụng, đầu vào bộ định thời có thể là nguồn xung lấy từ xung nhịp của vi điều khiển hoặc nguồn xung từ bên ngoài đưa đến. Vi điều khiển PIC18F4520 có bốn bộ định thời. Các bộ định thời được dùng để định khoảng thời gian hoặc đếm các sự kiện.

Một số ứng dụng của bộ định thời/đếm

Dựa trên hai chức năng chính này, trong thực tế, các bộ timer có thể được dùng để tạo xung, hẹn thời gian bật/tắt của một thiết bị, điều chế độ rộng xung (PWM – Pulse Width Modulation), đo tần số, đếm sản phẩm, tạo tốc độ baud cho PORT nối tiếp của vi điều khiển.

Ý nghĩa các bộ định thời/đếm

Vi điều khiển hoàn toàn có thể tạo ra các khoảng thời gian xác định hoặc đếm sự kiện bằng cách sử dụng vòng lặp hoặc ngắt (Bài 3). Tuy nhiên, sử dụng các bộ timer/counter giúp giải phóng CPU khỏi việc phải thực hiện các tác vụ lặp đi lặp lại chỉ để sử dụng cho mục đích này. Thay vào đó, nó có thể dành nhiều thời gian cũng như phân phối tài nguyên để thực hiện các tác vụ khác.

- Ví dụ khi sử dụng bộ định thời để tạo khoảng thời gian xác định: Xét ví dụ về mạng cảm biến giám sát nhiệt độ môi trường, node cảm biến cần định kỳ thực hiện đọc dữ liệu từ cảm biến và truyền về bộ thu thập trung tâm (cập nhật 1 lần/phút). Ngoài ra, các node cảm biến có thể phải thực hiện 1 số tác vụ khác trong quá trình hoạt động. Vậy điều gì đảm bảo được node cảm biến sẽ đọc và gửi dữ liệu về trung tâm đúng 1 lần/phút? Một cách đơn giản, ta có thể tính toán thời gian để thực hiện các tác vụ sao cho cứ đủ 1 phút thì đọc và gửi dữ liệu về trung tâm 1 lần. Tuy nhiên, giải pháp này gây ra nhiều khó khăn cho người lập trình vì nếu thêm hoặc bớt 1 tác vụ, lập tức thời gian đọc và gửi dữ liệu của node cảm biến sẽ không còn đúng như tính toán.

=> Giải pháp: sử dụng bộ định thời để tạo khoảng thời gian 1 phút này, điều này khiến cho khoảng thời gian đọc và gửi dữ liệu không phụ thuộc vào số tác vụ được thêm/bớt cho vi điều khiển.

- Ví dụ khi sử dụng bộ định thời để đếm sự kiện: Xét ví dụ 1 mạch cần đếm số sản phẩm trên băng tải đồng thời cũng phải thực hiện nhiều tác vụ khác, trong thực tế sản phẩm có thể xuất hiện (sườn dương/sườn âm) tại bất kỳ thời điểm nào. Giải pháp thủ công là vi điều khiển thực hiện kiểm tra mức logic tại chân kết nối với đầu vào xung đếm để biết có sản phẩm đi ngang qua hay không. Tuy nhiên, nếu đúng lúc đó mạch đang thực hiện 1 tác vụ khác thì nhiều khả năng sẽ không phát hiện đúng thời điểm xuất hiện sường dương/sườn âm dẫn đến đếm sai.

+ Một giải pháp khác khả thi hơn, thực tế hơn và phổ biến hơn là sử dụng ngắt ngoài của vi điều khiển (như đã học trong bài 3), việc sử dụng giải pháp này giúp cải thiện độ chính xác của đếm sản phẩm lên nhiều. Tuy nhiên ngắt thường được ưu tiên sử dụng hơn trong trường hợp có các tác vụ cần thiết hơn được thực hiện.

+ Để tận dụng và chia sẻ tài nguyên trong vi điều khiển, bộ Timer hoạt động ở chế độ đếm được lựa chọn phổ biến nhất trong việc đếm sản phẩm. Lý do: hoạt động đếm xảy ra hoàn toàn tự động với thời gian xử lý ngắn nhất (nhanh hơn sử dụng ngắt).

4.2. Các bộ định thời trong vi điều khiển PIC18F4520

Vi điều khiển PIC18F4520 có 4 bộ định thời/đếm:

- **Timer0**: có các đặc điểm cơ bản sau

Có thể lựa chọn bằng phần mềm để hoạt động ở chế độ định thời hoặc đếm, 8 bit hoặc 16 bit.

Các thanh ghi chứa số đếm có thể ghi/đọc được.

Có thể thiết lập hệ số chia tần bằng phần mềm.

Cho phép lựa chọn nguồn xung clock: xung nội hoặc xung ngoài.

Có thể lựa chọn được sườn tác động khi nguồn xung clock là xung ngoài.

Ngắt khi xảy ra tràn.

- **Timer1, Timer2, Timer3**: học trực tiếp trên lớp.

4.3. Timer0

4.3.1. Các thanh ghi liên quan đến Timer0

- Thanh ghi điều khiển Timer0: T0CON

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Ghi chú:

R = Cho phép đọc W = Cho phép ghi U = Không sử dụng, đọc bằng '0'

-n = Reset - POR '1' = Được thiết lập '0' = Được xóa -x = Reset không xác định

Bit 7 **TMR0ON:** Bit điều khiển Bật/Tắt Timer.

1 = Cho phép Timer0 (set bit này thành 1, Timer0 bắt đầu đếm).

0 = Dừng Timer0 (xóa bit này thành 0, Timer0 dừng đếm).

Bit 6 **T08BIT:** Bit lựa chọn 8-bit /16-bit của Timer0

1 = Timer0 được cấu hình là bộ đếm 8-bit (TMR0L)

0 = Timer0 được cấu hình là bộ đếm 16-bit

Bit 5 **T0CS:** Bit lựa chọn nguồn xung cấp cho Timer0

1 = Nguồn xung từ chân T0CKI

0 = Nguồn xung hệ thống (CLKO)

Bit 4 **T0SE:** Bit lựa chọn sườn xung đếm cho Timer0

1 = Lựa chọn sườn âm trên chân T0CKI

0 = Lựa chọn sườn dương trên chân T0CKI

Bit 3 **PSA:** Bit thiết lập bộ chia tần đầu vào

1 = Xung cấp vào Timer0 không qua bộ chia tần.

0 = Xung cấp vào Timer0 qua bộ chia tần (Prescaler).

Bit 2 **T0PS<2 :0>:** Bit lựa chọn hệ số chia tần (trong trường hợp chọn bit **PSA=0**, xung clock ở đầu vào của Timer0 sẽ bị chia trước theo hệ số chia tần trước khi thành xung đếm của Timer0)

111 = 1:256

110 = 1:128

101 = 1:64

100 = 1:32

011 = 1:16

010 = 1:8

001 = 1:4

000 = 1:2

- Thanh ghi chứa byte thấp của Timer0: TMR0L (8 bit, không định địa chỉ bit)

- Thanh ghi chứa byte cao của Timer0: TMR0H (8 bit, không định địa chỉ bit)

- Thanh ghi điều khiển ngắt : INTCON (xem phần ngắt và xử lý ngắt)

- Thanh ghi điều hướng cổng A: TRISA (xác định để RA4 có chiều vào khi Timer0 hoạt động ở chế độ đếm sự kiện ngoài).

Các thanh ghi liên quan tới Timer0 khi không sử dụng ngắt (***các bit được sử dụng là các bit in đậm màu đỏ trong bảng dưới đây***):

Tên	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
TMR0L	Thanh ghi chứa giá trị đếm byte thấp của Timer0							
TMR0H	Thanh ghi chứa giá trị đếm byte cao của Timer0							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	**TMR0IF**	INT0IF	RBIF
T0CON	**TMR0ON**	**T08BIT**	**T0CS**	**T0SE**	**PSA**	**T0PS2**	**T0PS1**	**T0PS0**
TRISA	RA7(1)	RA6(1)	RA5	**RA4**	RA3	RA2	RA1	RA0

4.3.2. Chế độ hoạt động của Timer0

Timer0 có thể hoạt động ở chế độ 8 bit hoặc 16 bit, để có thêm thông tin về các chế độ này, sinh viên xem clip bài giảng và đọc mục 5.2.2 trong giáo trình [1].

4.4. Lập trình điều khiển Timer0

- **Viết khung chương trình theo đúng định dạng**
- **Viết các lệnh của chương trình chính:**
  - ✓ B1: Khởi tạo các PORT (ADCON1, TRIS)
  - ✓ B2: Khởi tạo thanh ghi T0CON
  - ✓ B3: Nạp giá trị bắt đầu cho các thanh ghi TMR0H và TMR0L đáp ứng yêu cầu đề bài

Để tính toán giá trị nạp cho thanh ghi chứa số đếm, ta sử dụng các công thức tính toán sau (giải thích chi tiết được trình bày trong clip):

$$F_{TimerClock} = F_{OSC} / 4 / PrescaleValue$$

$$T_{TimerClock} = \frac{1}{F_{TimerClock}} = \frac{PrescaleValue}{F_{OSC} / 4}$$

$$Delay = TimerCount * \frac{PrescaleValue}{F_{OSC} / 4}$$

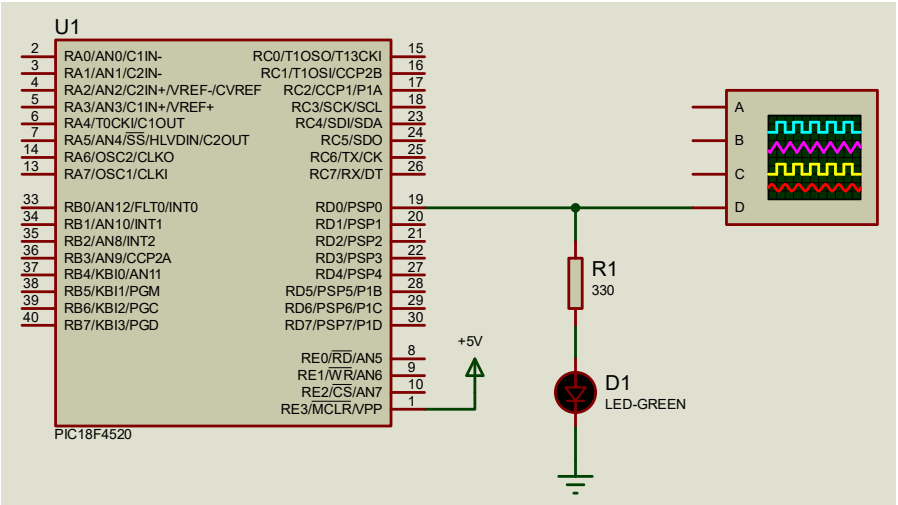
$$TimerCount = \frac{Delay * F_{OSC} / 4}{PrescaleValue}$$

$$TimerRegInit = TimerMaxValue - TimerCount + 1$$

- ✓ B4: Set bit TMR0ON lên 1 để Timer0 bắt đầu đếm và các lệnh thực hiện yêu cầu của đề bài

4.4.1. Lập trình tạo khoảng thời gian xác định cho Timer0

Cho mạch điện như hình vẽ, lập trình điều khiển để LED D1 sáng nhấp nháy với chu kỳ 1 giây (giả sử vi điều khiển hoạt động ở tần số Fosc=4MHz)



Bài làm:

Phân tích yêu cầu bài toán và tính chọn tham số

- ➢ **B1: Khởi tạo các PORT**
- ➢ **B2: Khởi tạo thanh ghi T0CON:**
  - ✓ Timer0 chưa chạy ngay từ đầu -> TMR0ON = 0;
  - ✓ Lựa chọn Timer0 ở chế độ 16 bit -> T08BIT = 0;
  - ✓ Hoạt động định thời -> cần lựa chọn nguồn xung clock nội -> T0CS = 0;

- ✓ Lựa chọn sườn tác động: do T0CS = 0 -> bit T0SE không có tác dụng -> có thể chọn là 0 hoặc 1;
- ✓ Lựa chọn có sử dụng hệ số chia tần hay không? Phụ thuộc vào yêu cầu của đề bài để quyết định. Đối với bài toán cụ thể này, ta thấy: Fosc/4 = 1MHz -> để tạo khoảng thời gian 0.5 giây cần đếm 500000 xung -> vượt quá giới hạn của 2 thanh ghi chứa số đếm (tổng cộng 16 bit, giá trị tối đa là 65535) -> cần phải sử dụng hệ số chia tần (PSA=0) với hệ số chia không nhỏ hơn 500000/65536 = 7.6, ví dụ chọn T0PS2:T0PS0=100 (hệ số chia trước là 32).

➢ **B3: Tính toán giá trị nạp lại cho 2 thanh ghi TMR0H và TMR0L:**

- ✓ Do hệ số chia trước là 32 -> tổng số xung clock để Timer0 đếm từ khi bắt đầu đến khi tràn để tạo thời gian trễ 0,5 giây là: (Fosc/4)/32/2=15625 (giá trị này phải là số nguyên).
- ✓ Tính toán giá trị nạp cho thanh ghi chứa 8 bit cao: TMR0H = 15625/256 (phép chia lấy phần nguyên).
- ✓ Tính toán giá trị nạp cho thanh ghi chứa 8 bit thấp: TMR0L = 15625%256 (phép chia lấy phần dư).

➢ **B4: Viết chương trình điều khiển theo yêu cầu của bài toán:**

- ✓ Để viết chương trình điều khiển, ta có 2 phương án thực hiện như sau:
  - o Sử dụng trực tiếp các thanh ghi của vi điều khiển PIC18F4520.
  - o Sử dụng các hàm có sẵn trong thư viện của MPLAB C18. Đối với timer0, ta sử dụng một số hàm sau:

CloseTimer0
Chức năng: Cấm ngắt và timer0.
Nguyên hàm: void CloseTimer0(void);
Ví dụ sử dụng:

```
CloseTimer0();
```

OpenTimer0
Chức năng: Cấu hình và bật timer0.

Nguyên hàm: void OpenTimer0(unsigned char ***config***);

Tham số ***config*** để lựa chọn cấu hình cho timer0 theo bảng sau:

TIMER_INT_ON TIMER_INT_OFF	Cho phép ngắt Không cho phép ngắt
T0_8BIT T0_16BIT	Chế độ 8 bit Chế độ 16 bit
T0_SOURCE_EXT T0_SOURCE_INT	Nguồn xung clock ngoài (RA4/T0CKI) Nguồn xung clock nội (Fosc/4)

T0_EDGE_FALL T0_EDGE_RISE	Sườn âm -> tăng số đếm Sườn dương -> tăng số đếm
T0_PS_1_1 T0_PS_1_2 T0_PS_1_4 T0_PS_1_8 T0_PS_1_16 T0_PS_1_32 T0_PS_1_64 T0_PS_1_128 T0_PS_1_256	Hệ số chia tần 1:1 Hệ số chia tần 1:2 Hệ số chia tần 1:4 Hệ số chia tần 1:8 Hệ số chia tần 1:16 Hệ số chia tần 1:32 Hệ số chia tần 1:64 Hệ số chia tần 1:128 Hệ số chia tần 1:256

Ví dụ sử dụng:

```
OpenTimer0( TIMER_INT_OFF &        // không cho phép ngắt
        T0_8BIT &                  // chế độ 8 bit
        T0_SOURCE_INT &            // nguồn xung clock nội
        T0_PS_1_32 );              // hệ số chia tần 1:32
```

ReadTimer0

Chức năng: Đọc giá trị hiện thời trong thanh ghi chứa số đếm TMR0L và TMR0H của timer0.

Nguyên hàm: unsigned int ReadTimer0(void);

Ví dụ sử dụng:

```
unsigned int result;
result = ReadTimer0();
```

WriteTimer0

Chức năng: Ghi giá trị khởi tạo cho thanh ghi chứa số đếm TMR0L và TMR0H của timer0.

Nguyên hàm: void WriteTimer0(unsigned int ***timer***);

Ví dụ sử dụng:

```
WriteTimer0(50000);
```

- Các hàm cho Timer1, Timer2 và Timer3 tương tự như của Timer0.

Để sử dụng các hàm trên, ta cần khai báo sử dụng "timers.h".

Code chương trình điều khiển sử dụng trực tiếp thanh ghi:

```
#include<p18f4520.h>
#pragma config OSC = HS   // xung he thong o che do HS
```

```
#pragma config WDT = OFF  // tat che do watchdog
#pragma config MCLRE = ON // cho phep Reset tren chan MCLR
#define xung PORTDbits.RD0
void main()
{
	T0CON = 0b00000100; // Timer0 16 bit, chia truoc 32
	ADCON1=0x0f;  // chon tat cac cac chan I/O la vao ra so
	TRISD = 0x00;	// thiet lap dau ra cho portB
	while(1){
		TMR0H = (65536-15625)/256; // so xung can dem trong nua chu ky [(Fosc/4)/32]/2
		TMR0L = (65536-15625)%256;
		T0CONbits.TMR0ON = 1;	// cho phep Timer hoat dong
		while(!INTCONbits.TMR0IF); // cho co tran
		T0CONbits.TMR0ON = 0;	// dung Timer
		INTCONbits.TMR0IF=0;	// xoa co tran
		xung=~xung;
	};
}
```

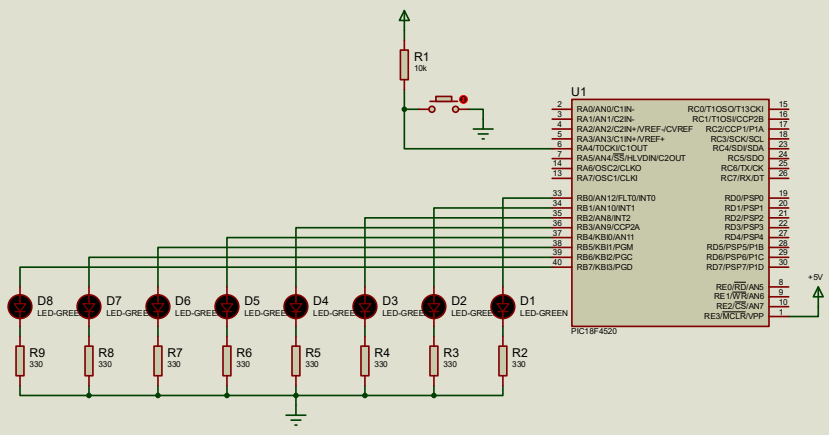
Code chương trình điều khiển sử dụng các hàm có sẵn trong thư viện MPLAB C18:

```
#include<p18f4520.h>
#include<timers.h>	// khai báo thêm thư viện cho các hàm điều khiển timer
#pragma config OSC = HS   // xung he thong o che do HS
#pragma config WDT = OFF  // tat che do watchdog
#pragma config MCLRE = ON // cho phep Reset tren chan MCLR
#define xung PORTDbits.RD0
void main()
{
	TRISD = 0x00;	// thiet lap dau ra cho portB
	// Hàm OpenTimer0 dùng để cấu hình và bật Timer0
	OpenTimer0( TIMER_INT_OFF &		// khong su dung ngat timer
			T0_16BIT &		// chon che do 16 bit
			T0_SOURCE_INT &	// lua chon nguon xung clock noi
			T0_PS_1_32 );		// he so chia tan 1 (khong dung chia tan)
	while(1){
```

```
            WriteTimer0(65536-15625);  // ghi gia tri khoi tao cho timer0
            while(!INTCONbits.TMR0IF); // cho co tran
            xung=~xung;
            INTCONbits.TMR0IF=0;       // xoa co tran
        }
}
```

4.4.2. Lập trình đếm sự kiện ngoài sử dụng Timer0

Cho mạch điện như hình vẽ, lập trình điều khiển sử dụng Timer0 đếm số lần nhấn nút, hiển thị số lần đếm được dưới dạng số nhị phân trên 8 LED (D8: MSB, D1: LSB).



Bài làm:

Phân tích yêu cầu bài toán và tính chọn tham số

- **B1: Khởi tạo các PORT**
- **B2: Khởi tạo thanh ghi T0CON:**
  - ✓ Timer0 chưa chạy ngay từ đầu -> TMR0ON = 0
  - ✓ Lựa chọn Timer0 ở chế độ 8 bit -> T08BIT = 1
  - ✓ Đếm xung clock ngoài -> T0CS = 1
  - ✓ Lựa chọn sườn tác động: có 2 phương án:
    - **Bộ đếm tăng khi nút được nhấn (sườn âm) -> T0SE = 1 (trong code ví dụ này)**
    - Bộ đếm tăng khi nút được nhả (sườn dương) -> T0SE = 0
  - ✓ Do dùng timer0 để đếm số lần nhấn nút, mỗi lần nhấn số nhị phân tăng lên 1 -> không sử dụng bộ chia tần -> PSA = 1.

- ✓ T0PS2:T0PS0: không quan tâm (do PSA = 1)

- **B3: Tính toán giá trị nạp lại cho thanh ghi TMR0L**
- **B4: Viết chương trình điều khiển theo yêu cầu của bài toán:**

Code chương trình điều khiển sử dụng trực tiếp thanh ghi:

```
#include<p18f4520.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config MCLRE = ON
void main()
{
        // chon che do cho Timer0
        T0CON = 0b01111000;//8bit,dem xung tren chan T0CKI,suon am,khong chia tan
        // thiet dat cho PORT
        ADCON1= 0x0f;      // vao ra so
        TRISAbits.TRISA4 = 1;  // chan RA4 chieu vao
        TRISB = 0x00;      // dat dau ra cho PortB
        LATB  = 0x00;      // dat truoc dau ra PortB = 0
        TMR0L = 0;
        T0CONbits.TMR0ON=1;// cho phep Counter bat dau dem
        while(1)
        {
                PORTB=TMR0L;          // xuat gia tri hien thoi trong TMR0L ra PORTB
        }
}
```

Code chương trình điều khiển sử dụng các hàm có sẵn trong thư viện MPLAB C18:

```
#include<p18f4520.h>
#include<timers.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config MCLRE = ON
void main()
{
        TRISB = 0x00;      // dat dau ra cho PortB
```

```
    LATB  = 0x00;      // dat truoc dau ra PortB = 0
    OpenTimer0( TIMER_INT_OFF &          // khong su dung ngat timer
                T0_8BIT &                // chon che do 8 bit
                T0_EDGE_FALL &           // timer tang so dem khi co suon am
                T0_SOURCE_EXT &          // lua chon nguon xung clock ngoai
                T0_PS_1_1 );    // he so chia tan 1 (khong dung chia tan)
    WriteTimer0(0); // ghi gia tri khoi tao cho timer0
    while(1)
    {
        // Ép kiểu dữ liệu về 8 bit không dấu để xuất ra PORTB
        PORTB=(unsigned char) ReadTimer0();
    }
}
```

- **Sinh viên vẽ mạch mô phỏng trên Proteus, tạo project, viết code theo ví dụ và chạy mô phỏng để kiểm chứng kết quả.**