

ارتباط سخت افزار با نرم افزار

هر دستگاه برنامه پذیر (مانند یک کامپیوتر یا یک میکروکنترلر) دارای دو بخش اصلی است: سخت افزار و نرم افزار. با چیسستی این دو بخش کم و بیش آشنا هستیم.

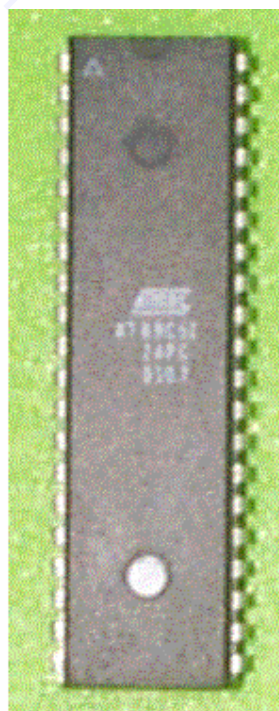
نکته بسیار مهم و در عین حال ساده ای که باید به آن توجه کرد نحوه برقراری ارتباط بین سخت افزار و نرم افزار در یک میکروکنترلر است.

باید راهی وجود داشته باشد که دستوراتی که نرم افزار صادر می کند، به سخت افزار منتقل شود تا سخت افزار به درستی آنها را اجرا کند. در میکروکنترلرها «واسطه ارتباطی میان سخت افزار و نرم افزار»، حافظه داخلی میکروکنترلر است. حافظه داخلی به دویخش تقسیم می شود که یکی از این دو بخش وظیفه برقراری ارتباط میان سخت افزار و نرم افزار را بر عهده دارد. هر بایت در این بخش یک «رجیستر» نامیده می شود. هر رجیستر کاربرد مشخصی دارد. به این ترتیب، نرم افزار به وسیله قرار دادن مقادیر مشخصی در این رجیسترها دستورات مشخصی به سخت افزار می دهد.

ورودی و خروجی معمولی (Simple I/O)

یک میکروکنترلر، بر خلاف یک کامپیوتر، مجهز به وسایل ورودی و خروجی پیشرفته ای مانند Keyboard، Speaker، Monitor و یا Mouse نیست. بلکه تنها راه ارتباط میکروکنترلرها (مانند هر IC دیگری) پایه های IC می باشد. (پایه های IC زائده های فلزی کوچکی هستند که اطراف IC قرار می گیرند. (شکل ۱)

P1.0	1	40	VCC
P1.1	2	39	P0.0 (AD0)
P1.2	3	38	P0.1 (AD1)
P1.3	4	37	P0.2 (AD2)
P1.4	5	36	P0.3 (AD3)
P1.5	6	35	P0.4 (AD4)
P1.6	7	34	P0.5 (AD5)
P1.7	8	33	P0.6 (AD6)
RST	9	32	P0.7 (AD7)
(RXD) P3.0	10	31	EA/VPP
(TXD) P3.1	11	30	ALE/PROG
(INT0) P3.2	12	29	PSEN
(INT1) P3.3	13	28	P2.7 (A15)
(T0) P3.4	14	27	P2.6 (A14)
(T1) P3.5	15	26	P2.5 (A13)
(WR) P3.6	16	25	P2.4 (A12)
(RD) P3.7	17	24	P2.3 (A11)
XTAL2	18	23	P2.2 (A10)
XTAL1	19	22	P2.1 (A9)
GND	20	21	P2.0 (A8)



شکل ۱. سمت راست: میکروکنترلر A89C51. سمت چپ: نام پایه های این میکروکنترلر.

میکروکنترلر AT89C51 دارای ۴۰ پین یا پایه است. ۳۲ تا از این پین ها، ورودی ها و خروجی های دیجیتال هستند. به این معنی که به عنوان خروجی ولتاژهای ۰ ولت و یا ۵ ولت را تولید می کنند،

(۰ یا ۱ منطقی). برای مثال یک خروجی دیجیتال نمی تواند یک موج سینوسی تولید کند. اما می تواند یک موج مربعی با دو سطح صفر و ۵ ولت ایجاد نماید.

برای یک ورودی دیجیتال نیز تنها دو مقدار ۰ یا ۱ منطقی قابل درک است. اگر ولتاژ اعمال شده از خارج میکرو از مقدار مشخصی (حدود ۲ ولت) بالاتر باشد از نظر میکرو ۱، و اگر از آن حد پایین تر باشد صفر است.

از میان این ۴۰ پایه، ۸ پایه کاربرد هایی غیر از I/O دارند و ۳۲ پایه دیگر در غالب ۴ «پورت» ۸ بیتی واسطه ارتباط میکروکنترلر با جهان خارج هستند. این چهار پورت از ۰ تا ۳ شماره گذاری شده اند. محل پایه های هر پورت در شکل ۱ نمایش داده شده است.

همانطور که گفته شد وسیله ارتباط میان سخت افزار و نرم افزار، رجیسترها هستند. فرض کنید می خواهیم ولتاژ یکی از پایه های میکروکنترلر را ۵ ولت قرار دهیم (۱ منطقی). توجه کنید که در این حالت این پایه، یک خروجی است. باید رجیستری وجود داشته باشد که این امکان را برای برنامه نویس فراهم آورد تا مقدار منطقی دلخواهی را بر روی هر یک از پایه های میکروکنترلر قرار دهد. در فایل Header ای که ما به برنامه های خود اضافه (include) می کنیم (AT89X51.h) برای هر یک از این رجیستر ها نامی در نظر گرفته شده تا کار برنامه نویسی ساده تر شود^۱. به این ترتیب نیازی نیست که ما هر بار با مراجعه به شکل حافظه، آدرسها را بیابیم و می توانیم از این اسامی استفاده کنیم. هر پورت ۸ بیتی با یک بایت (۸ بیت) متناظر است. بنابراین تناظری یک به یک میان، بیت های هر یک از این رجیستر ها با پایه های میکروکنترلر به وجود می آید و مقدار هر بیت در رجیستر، تعیین کننده ولتاژ پایه متناظر آن خواهد بود. به عنوان مثال رجیستر متناظر پورت ۰، P0، در شکل ۲ نشان داده شده است.

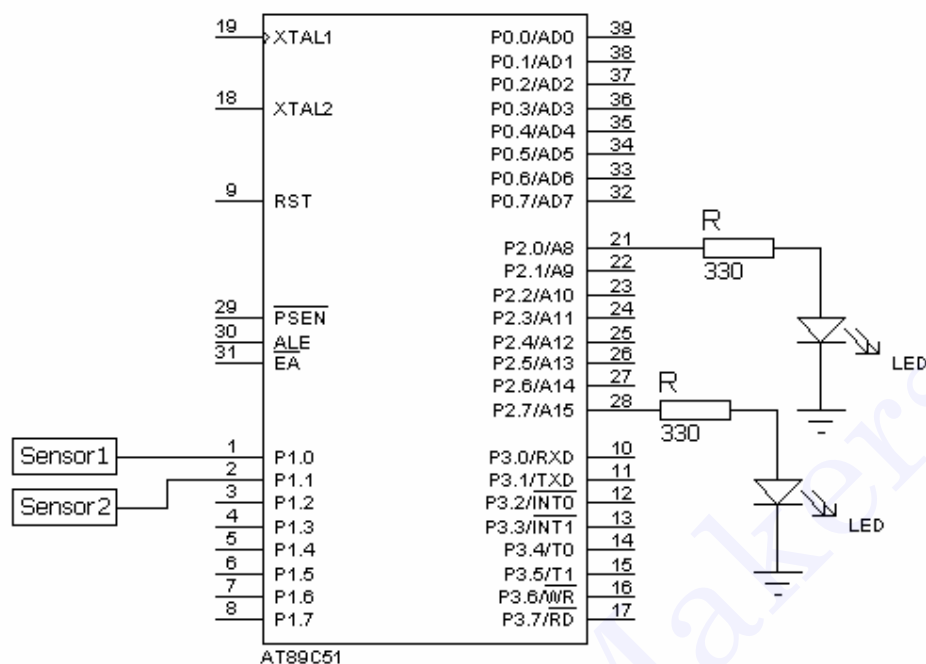
P0 Port 0								Value after reset	Bit name
1	1	1	1	1	1	1	1		
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0		
bit7				bit0					

شکل ۲. رجیستر P0. این رجیستر ارتباط بین نرم افزار و پین های پورت ۰ را برقرار می کند.

مثال ۱ :

فرض کنید خروجی دو سنسور تشخیص رنگ (سیاه و سفید) به دو پایه از میکروکنترلر متصل شده است. هر یک از این سنسورها اگر رنگ سفید را تشخیص دهد (بیند) خروجی خود را ۱ منطقی (۵ ولت) قرار می دهد، و برای سیاه، ۰ منطقی (صفر ولت). حال می خواهیم، برای آزمودن این سنسورها، دو چراغ (LED) را به میکروکنترلر متصل کرده و برنامه ای بنویسیم که با تشخیص سفید چراغ ها روشن شوند و با تشخیص سیاه، خاموش. مدار لازم در شکل ۳ نشان داده شده است.

^۱ بد نیست یک بار محتویات این فایل را ببینید. با انتخاب File|Open در Keil، می توانید این فایل را از آدرس [Keil]\C51\INC\Atmel باز کنید. منظور از [Keil] محلی است که Keil را در آن نصب کرده اید.



شکل ۳. محل پایه ها جابجا شده، می توانید آنها را از روی اسم یا شماره پایه پیدا کنید.

Sensor ها به پین اول و دوم از پورت ۱ (P1.0 و P1.1) متصل شده اند. LED ها هم به P2.0 و P2.7.

برنامه زیر، عمل مورد نظر را انجام خواهد داد.

```
#include <AT89X51.h>

//Pin definitions
#define Sensor1 P1_0
#define Sensor2 P1_1
#define LED1 P2_0
#define LED2 P2_7

void main (void)
{
    Sensor1 = 1; //necessary before using it as an input
    Sensor2 = 1; //necessary before using it as an input
    while (1)
    {
        if (Sensor1 == 1) LED1 = 1; else LED1 = 0; //or LED1 = Sensor1;
        if (Sensor2 == 1) LED2 = 1; else LED2 = 0; //or LED2 = Sensor2;
    }
}
```

نکته مهم: قبل از اینکه بتوانیم از یکی از پایه های میکروکنترلر به عنوان ورودی استفاده کنیم، لازم است ابتدا همان پین را یک کنیم. این عمل پایه مورد نظر را آماده دریافت ورودی می کند.

Timer ها

میکروکنترلر AT89C51 دارای دو Timer است که می توانند به صورت مستقل از هم، برای سنجش بازه های زمانی مورد استفاده قرار گیرند.

چگونگی عملکرد Timer ها

برای اینکه سنجش زمان برای میکروکنترلر میسر باشد باید معیاری از زمان در اختیار سخت افزار قرار بگیرد. این کار به وسیله سیگنال Clock انجام می شود. سیگنال Clock، یک موج پریودیک با فرکانس مشخص است. فرکانس Clock برای یک 8051 حداکثر می تواند 24 MHz باشد اما مقدار معمول آن 12 MHz است. فرکانس این سیگنال، پیش از آنکه به Timer ها اعمال شود، به ۱۲ تقسیم می شود. یعنی فرکانس سیگنال Clock اعمال شده به Timer ها 1 MHz خواهد بود. (از این پس فرکانس Clock اعمال شده به Timer را 1 MHz در نظر می گیریم، بنابراین هر پریود Clock که به آن «سیکل ماشین» نیز گفته می شود، ۱ میکروثانیه خواهد بود)

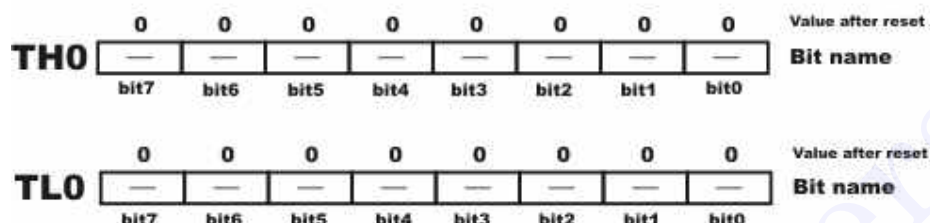
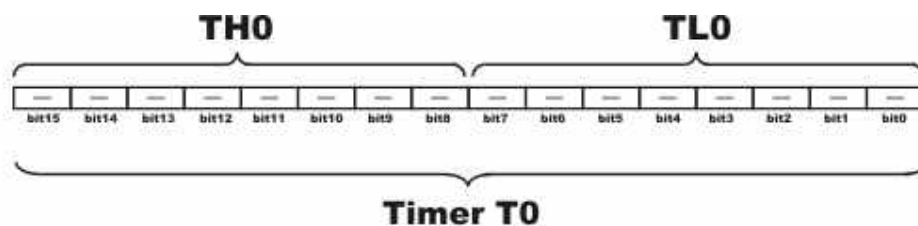
هر Timer در واقع یک شمارنده (Counter) است که تعداد پریودهای سیگنال Clock را (بعد از تقسیم فرکانس بر ۱۲) می شمارد. اگر فرکانس Clock اعمال شده به Timer را 1 MHz فرض کنیم، هر ۱ میکرو ثانیه، محتوای این شمارنده یک واحد افزایش می یابد. می توان با ضرب کردن محتوای این شمارنده در پریود Clock مدت زمان سپری شده را محاسبه نمود.

هر یک از Timer های ۸۰۵۱، ۱۶ بیتی هستند. یعنی مقداری که Timer می شمارد در دو بایت (۱۶ بیت) نگهداری می شود. بنابراین بزرگترین عددی که یک Timer می تواند ذخیره کند، $2^{16} - 1$ ، یعنی ۶۵۵۳۵ است. اگر رجیستر Timer حاوی این عدد باشد، با اعمال پالس بعدی Clock چه اتفاقی خواهد افتاد؟ مقدار بعدی رجیستر ۰ است. در واقع رجیستر Timer، Overflow یا سرریز می شود و بیت دیگری به جز ۱۶ بیت شمارنده (Flag) به نشان Overflow، یک می شود. از این اتفاق (Overflow و یک شدن بیت Flag) برای ایجاد تأخیر به میزان مشخص استفاده می شود.

فرض کنید قصد داریم تأخیری به اندازه ۱۰۰ میکروثانیه ایجاد کنیم. در این صورت، عدد 100 - 65535 را در رجیستر Timer قرار می دهیم. بعد از ۱۰۰ پریود Clock محتوای رجیستر به حداکثر خود می رسد و Overflow می شود و همزمان با آن بیت نشان دهنده سرریز نیز یک می شود. در واقع یک شدن بیت Overflow به معنی سپری شدن زمان مورد نظر است.

همانطور که گفته شد میکروکنترلر AT89C51 دارای دو Timer است. این دو Timer با اعداد ۰ و ۱ نشان داده می شوند. همچنین رجیسترهای هر کدام از این دو Timer با همین دو رقم از هم تمیز داده می شوند. از این پس، برای حفظ کلیت مطالب، به جای ۱ یا ۰ از x استفاده می کنیم، زیرا این دو Timer (در حوزه کاری ما) کاملاً مشابهند.

حال با دقت بیشتری سخت افزار Timer را بررسی می کنیم. رجیستر ۱۶ بیتی Timer در واقع از دو رجیستر ۸ بیتی به نامهای THx و TLx تشکیل می شود. THx (Timer High byte)، ۸ بیت بالا و TLx (Timer Low byte) ۸ بیت پایین هستند. شکل ۴ رجیستر Timer را نشان می دهد.



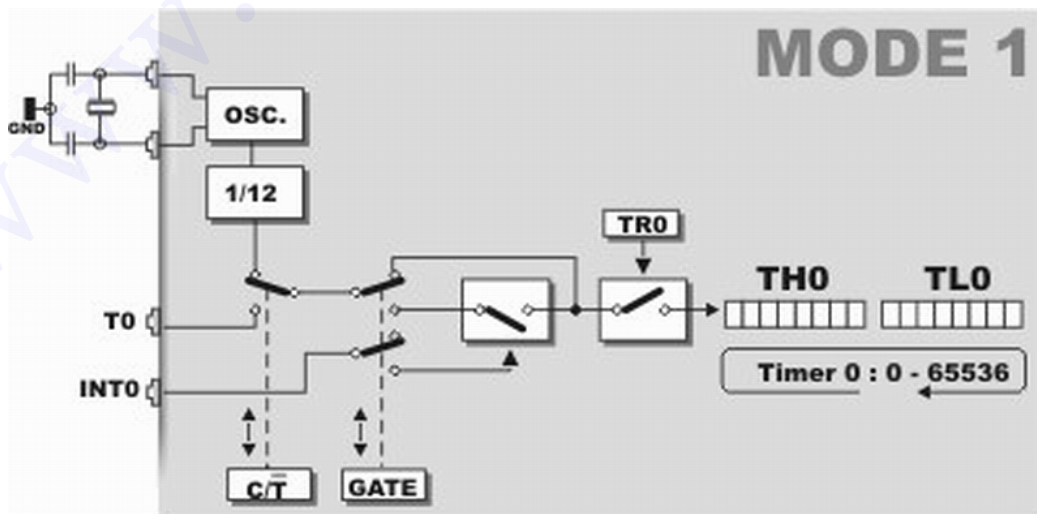
شکل ۴. رجیسترهای شمارنده Timer. ۸ بیت بالا در TH و ۸ بیت پایین در TL قرار می گیرند.

مدهای کاری Timer

هر Timer می تواند چهار "رفتار" متفاوت داشته باشد که به هر یک از اینها يك "مد کاری" می گویند. از این چهار مد، ما تنها دو مد را بررسی می کنیم. مد ۱ و مد ۲.

عملکرد Timer در مد ۱

Timer در مد ۱ از تمام ۱۶ بیت خود برای شمارش استفاده می کند. یعنی بزرگترین عدد در Timer می تواند 65535 باشد. نتیجتاً طولانی ترین تأخیری که Timer در مد ۱ می تواند به تنهایی ایجاد کند، کمی بیش از ۶۵ میلی ثانیه است. در این مد، هر بار که Timer سرریز می شود، باید مقدار مورد نظر را دوباره در آن Load کرد. این عمل نیاز به چند میکروثانیه زمان دارد و زمانی که بازه زمانی مورد نظر کوچک باشد (مثلاً تولید موج 100 KHz) این مسئله دقت را کاهش می دهد. (مثلاً به جای 100 KHz، 80 KHz خواهد شد)



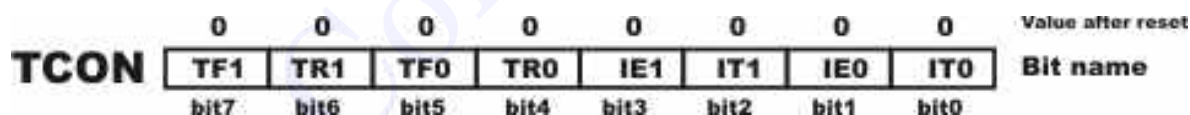
شرح کار این بیتها در جدول زیر آمده است. بیتهای GATE و C/T (بیتهای ۱ و ۲ و ۶ و ۷) فعلاً برای ما کاربردی ندارند.

Bit	Bit Name	Purpose	Timer
7	GATE1	1 Timer works only if INT1 (P3.3) is set 0 Timer works regardless of INT1 (P3.3)	T1
6	C/T1	1 Timer counts impulses on T1 (P3.5) 0 Timer counts impulses of internal oscillator	T1
5	T1M1	Timer mode	T1
4	T1M0	Timer mode	T1
3	GATE0	1 Timer works only if INT0 (P3.2) is set 0 Timer works regardless of INT0 (P3.2)	T0
2	C/T0	1 Timer counts impulses on T0 (P3.4) 0 Timer counts impulses of internal oscillator	T0
1	T0M1	Timer mode	T0
0	T0M0	Timer mode	T0

مدهای مختلف Timer ها که به وسیله بیت‌های رجیستر TMODE تعیین می شوند به شرح زیر می باشند. ما تنها از مدهای ۱ و ۲ استفاده خواهیم کرد.

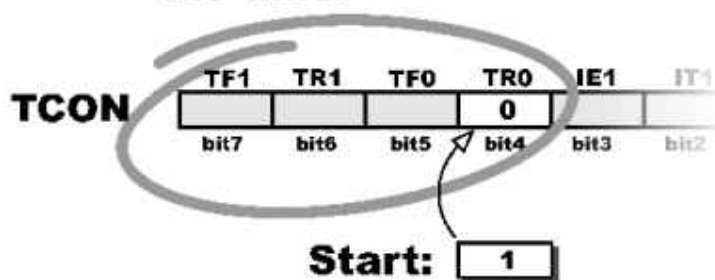
TOM1	TOM0	Mode	Description
0	0	0	13-bit Timer
0	1	1	16-bit Timer
1	0	2	8-bit <i>auto-reload</i>
1	1	3	<i>Split mode</i>

رجیستر TCON (Timer Control):



از میان این بیت‌ها تنها چهار بیت بالا به Timer مربوط می شوند.

Bits for controlling the timer



شرح کار این بیتها در جدول زیر آمده است.

Bit	Bit Name	Purpose	Timer
7	TF1	This bit is automatically set in case of overflow in Timer T1	T1
6	TR1	1 - Timer T1 is on 0 - Timer T1 is off	T1
5	TF0	This bit is automatically set in case of overflow in Timer T0	T0
4	TR0	1 - Timer T0 is on 0 - Timer T0 is off	T0

بیت TF (Timer Flag) : TFX در واقع همان بیت نشان دهنده Overflow می باشد.
بیت TRx (Timer Run bit) : اگر این بیت صفر شود، Timer دیگر پالس های سیگنال Clock را نخواهد شمرد و محتوای رجیستر شمارنده بدون تغییر باقی می ماند.

برنامه ریزی Timer در مد ۱

۱. با توجه به اینکه هر Timer را در چه مدی استفاده خواهید کرد، مقدار مناسبی در TMOD قرار دهید. GATE و C/T را صفر کنید.
۲. با توجه به مقدار تأخیر مورد نیاز، مقادیر مناسب را در TH و TL قرار دهید.
۳. Timer را روشن کنید.
۴. منتظر بمانید تا بیت Overflow (TFx) یک شود.
۵. Timer را متوقف کنید.
۶. بیت سرریز را صفر کنید، تا در سرریز بعدی یک شدن آن قابل تشخیص باشد.
۷. به مرحله ۲ باز گردید.

برنامه ریزی Timer در مد ۲

۱. با توجه به اینکه هر Timer را در چه مدی استفاده خواهید کرد، مقدار مناسبی در TMOD قرار دهید. GATE و C/T را صفر کنید.
۲. رجیستر THx را Load کنید. با توجه به امکان Auto Reload نیازی به Load کردن TLx نداریم. به محض روشن شدن Timer مقدار درون THx به TLx کپی شده و شمارش آغاز خواهد شد.
۳. Timer را روشن کنید.
۴. منتظر یک شدن TFX بمانید.
۵. TF را صفر کنید.
۶. به مرحله ۴ باز گردید.

مثال ۲ :

می خواهیم موج مربعی با فرکانس 25 KHz روی پایه ۱ ول از پورت ۱ (P1.0) ایجاد کنیم. در اینصورت لازم است که در هر نیم پریود یک بار ولتاژ این پایه تغییر کند. پریود این موج ۴۰ میکروثانیه است و نصف آن ۲۰ میکروثانیه خواهد شد. اگر از مد ۱ Timer استفاده کنیم، زمان لازم جهت اجرا شدن مراحل ۵، ۶، ۲ و ۳ حدود ۱۰ میکروثانیه است و در این مدت Timer خاموش است و زمان را اندازه

گیری نمی کند. بنابراین این ۱۰ میکروثانیه خطا محسوب می شود. از آنجا که این خطا با ۲۰ میکرو ثانیه قابل مقایسه است و فرکانس موج حاصله را به شدت تغییر می دهد (16 KHz به جای 25 KHz) بنابراین نمی توانیم از مد ۱ استفاده کنیم و باید از خاصیت Auto Reload مد ۲ بهره ببریم که موجب حذف این خطا می شود.

برنامه زیر استفاده از Timer 0 را برای تولید موجی با فرکانس 25 KHz نشان می دهد.

```
#include <AT89X51.H>

void main (void)
{
    TMOD = 0x02;    //We want to use timer 0 in mode 2
                    //GATE = 0, C/T = 0 for both
    TH0 = -20;      //20 us delay
    TR0 = 1;        //Start the timer
    while (1)
    {
        while (!TF0); //wait for TF0
        P1_0 = !P1_0; //Toggle output pin
        TF0 = 0;      // clear TF0
    } //And again and again and ...
}
```

همان طور که می بینید به هنگام استفاده از مد ۲ روشن کردن Timer و مقدار دهی به TH تنها یک بار انجام می شود (بر خلاف مد ۱)

وقفه ها

فرض کنید بخواهیم در مثال ۲ «هم زمان» دو موج مربعی تولید کنیم. با روشی که در بالا اراوه شد این امر میسر نمی باشد. علت آنست که حلقه while (!TF0) اجرای برنامه را در همان خط متوقف خواهد کرد و اگر TF1 (مربوط به موج مربعی دوم) قبل از TF0 یک شود، برنامه پاسخ مناسب، یعنی Not کردن یه موقع بین خروجی مربوط به موج مربعی دوم را به یک شدن آن نمی دهد.

بنابراین لازم است روش دیگری را برای رسیدن به این منظور به کار ببریم. مکانیسمی که برای حل این مشکل به کار می رود وقفه یا Interrupt نام دارد. وقفه در واقع خبر وقوع اتفاق مشخصی است که از سخت افزار به نرم افزار داده می شود و نرم افزار عکس العمل مناسب را به آن نشان می دهد.

میکروکنترلر AT89C51 دارای ۵ منبع وقفه مختلف است که در ادامه به بررسی آنها خواهیم پرداخت. دو عدد از این وقفه ها به Timer ها اختصاص دارند (هر Timer یک وقفه). هر وقفه دارای یک ISR (Interrupt Service Routine) می باشد. هر وقفه، تابعی است که به هنگام وقوع آن یه طور خودکار اجرا می شود. این تابع در واقع همان پاسخ نرم افزار به سخت افزار است. در زبان C مخصوص میکروکنترلرها، تابع وقفه (ISR) به وسیله کلمه Interrupt و شماره وقفه مورد نظر مشخص می شود. شماره وقفه ها در ۸۰۵۱ در جدول زیر مشخص شده است. جدول زیر شماره های قراردادی برای وقفه ها را نشان می دهد. ما فعلاً تنها با وقفه های ۱ و ۲ (Timer ها) سروکار داریم.

Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

وقفه ها را می توان فعال یا غیر فعال کرد. ممکن است شما در طراحی خاصی به بعضی وقفه ها نیازی نداشته باشید در این صورت می توانید آنها را غیر فعال کنید. در مقابل باید تمام وقفه هایی که قصد دارید مورد استفاده قرار دهید را فعال کنید. رجیستر IE برای تعیین اینکه کدام وقفه فعال و کدام یک غیر فعال باشد استفاده می شود.

رجیستر IE :



Bit	Purpose
EA	Enables/disables all interrupt sources
ET2	Timer T2 interrupt
ES	UART and SPI interrupts
ET1	Timer T1 interrupt
EX1	External interrupt: pin INT1
ET0	Timer T0 interrupt
EX0	External interrupt: pin INTO

در مورد بیت EA توجه کنید که صفر بودن این بیت، تمام وقفه ها را غیر فعال می کند، اما یک بودن آن به تنهایی وقفه ای را فعال نمی کند بلکه لازم است بیت خاص وقفه مورد نظر نیز یک شود.

مثال ۳ :

در این مثال قصد داریم با استفاده از Timer ها دو موج مربعی با فرکانسهای 20 KHz و 10 KHz تولید کنیم. با استفاده از وقفه ها «به نظر می رسد» که میکروکنترلر دو کار را همزمان انجام می دهد، اما در واقع اینطور نیست!

```
#include <AT89X51.H>

void Timer0ISR (void) interrupt 1
{
    P1_0 = !P1_0;
}

void Timer1ISR (void) interrupt 3
{
    P1_1 = !P1_1;
}

void main (void)
{
    ETO = 1;          //Enable Timer0 Interrupt
    ET1 = 1;          //Enable Timer1 Interrupt
    EA = 1;            //Enable all
    TMOD = 0x22;       //both timers in mode 2
    TH0 = -50;         //50 us delay
    TH1 = -25;         //20 us delay
    TR0 = 1;           //Start timer0
    TR1 = 1;           //Start timer1
    while (1);         //Do nothing!!
}
```

نکته مهم :

دو تابع ISR هیچ جا در طول برنامه فراخوانی نشده اند. در صورتی که سعی کنید یک ISR را (که با کلمه Interrupt مشخص می شود) در برنامه فراخوانی کنید کامپایلر پیغام خطایی نمایش خواهد داد! یک تابع ISR توسط سخت افزار و به طور خودکار فراخوانی می شود. بر خلاف توابع متداول که توسط نرم افزار (در طول برنامه) فراخوانی می شوند.

در این برنامه میکروکنترلر دو موج مربعی تولید می کند اما تابع main عملاً کاری انجام نمی دهد. چند دستور اول تنظیماتی هستند که ظرف چند میکروثانیه اجرا می شوند و سپس حلقه while (1); تابع main را معلق نگاه می دارد. این نیز به خاطر استفاده از Interrupt است.

وقفه خارجی

همانطور که تا کنون توضیح داده شد، فراخوانی خودکار ISR مربوط به هر وقفه سبب می شود که میکروکنترلر بتواند چند کار را به طور همزمان انجام دهد. در واقع میکروکنترلر در هر لحظه تنها یک کار انجام می دهد، اما به خاطر سرعت زیاد اجرای برنامه (حدود نیم میلیون دستور در ثانیه) اینطور به نظر می رسد که کارها همزمان انجام می شوند.

وقفه خارجی به میکروکنترلر اجازه می دهد که در کنار کارهای دیگری که انجام می دهد از وقوع "اتفاق" مشخصی در دنیای خارج نیز "با کمک یک وقفه" مطلع شود. در ساده ترین حالت این اتفاق می تواند فشرده شدن یک کلید باشد.

دو پایه از پایه های میکروکنترلر به نامهای INT0 و INT1 مربوط به دو وقفه خارجی هستند. این وقفه ها می توانند در دو حالت حساس به لبه و حساس به سطح کار کنند.

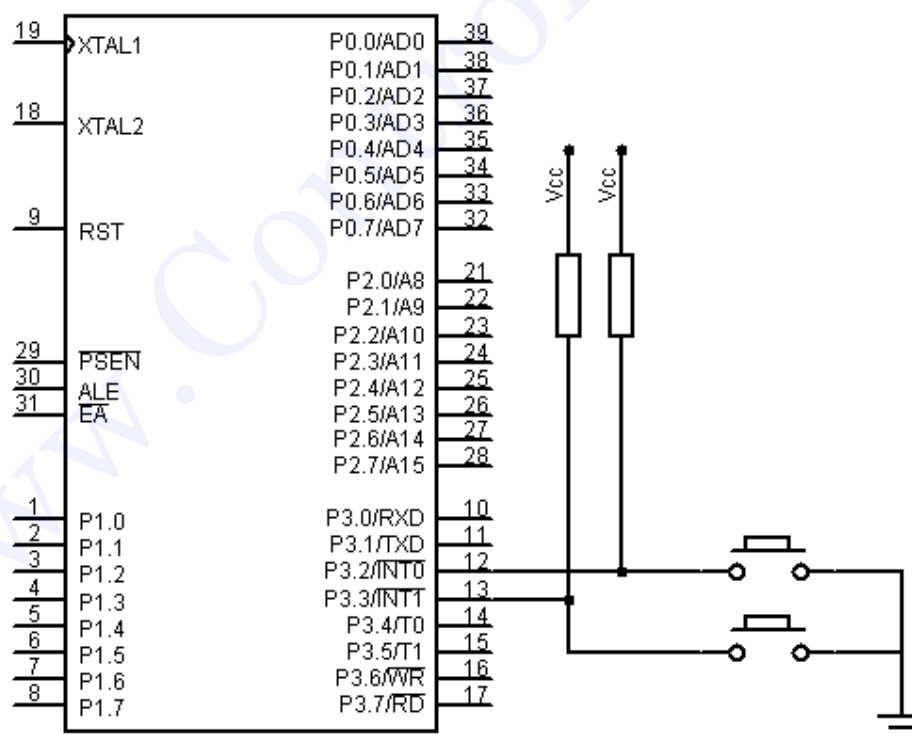
در حالت حساس به لبه اگر سطح منطقی این پین ها از يك به صفر تغییر کند (از ۵ ولت به ۰ ولت) يك وقفه رخ خواهد داد. این تغییر سطح را اصطلاحاً لبه گویند. در واقع وقفه با يك لبه منفی (گذر ۱ به ۰) اجرا می شود.

وقفه حساس به سطح با صفر شدن سطح منطقی رخ می دهد و تا زمانی که ولتاژ اعمال شده به پایه وقفه صفر باشد ISR وقفه خارجی پی در پی اجرا خواهد شد.

مثال ۴ :

در این مثال قصد داریم مداری با دو خروجی طراحی کنیم که هر خروجی یک موج مربعی تولید می کند. (البته با میکروکنترلر کارهای زیادی می توان انجام داد و تولید موج مربعی تنها کاربرد آن نیست!!) این مدار دو کلید دارد که در حکم کلید ON/OFF خروجی ها هستند.

مدار زیر اتصال کلید ها را برای وقفه دادن به میکروکنترلر نشان می دهد. مستطیل ها مقاومت هستند!! وقتی کلید قطع است به علت نا چیز بودن جریان ورودی پایه INTX ولتاژ دو سر مقاومت صفر است، بنابراین پایه INTX یک خواهد بود. زمانی که کلید وصل است پایه INTX مستقیماً به زمین وصل می شود و این پایه صفر می شود. پس فشردن کلید سبب ایجاد یک گذر از یک به صفر یا یک لبه منفی (۱ به ۰) روی پایه INTX می شود. اگر وقفه را در وضعیت حساس به لبه قرار دهیم فشردن کلید سبب اجرا شدن وقفه می شود. اما وصل نگه داشتن آن تأثیری ندارد. زیرا لبه دیگری ایجاد نمی کند.



برنامه زیر عمل مورد نظر را انجام خواهد داد.

```
#include <AT89X51.H>

void Timer0ISR (void) interrupt 1
{
    P1_0 = !P1_0;
}

void Timer1ISR (void) interrupt 3
{
    P1_1 = !P1_1;
}

void Ex0ISR (void) interrupt 0
{
    TR0 = !TR0;
}

void Ex1ISR (void) interrupt 2
{
    TR1 = !TR1;
}

void main (void)
{
    ETO = 1;          //Enable Timer0 Interrupt
    ET1 = 1;          //Enable Timer1 Interrupt
    EX0 = 1;          //Enable External Interrupt 0
    EX1 = 1;          //Enable External Interrupt 1
    ITO = 1;          //Edge-triggered, Interrupt 0
    IT1 = 1;          //Edge-triggered, Interrupt 1
    EA = 1;           //Enable all
    TMOD = 0x22;      //both timers in mode 2
    TH0 = -50;        //50 us delay
    TH1 = -25;        //20 us delay
    while (1) {P1_2 = !P1_2;}    //Do nothing!!
}
```

توضیح : بیت‌های IT0 و IT1 حساس به لبه یا سطح بودن وقفه را تعیین می‌کنند. در ابتدا Timer ها روشن نشده‌اند، زیرا در تابع main TRx مقدار دهی نشده و مقدار پیش فرض آن صفر است. بنابراین Timer ها خاموشند و سرریزی رخ نمی‌دهد. از این رو وقفه هم اتفاق نمی‌افتد. ISR نیز اجرا نمی‌شود و از موج مربعی خبری نیست.

با فشردن هر کلید یکی از وقفه‌های خارجی اجرا می‌شوند و بیت TRx را تغییر می‌دهند. اگر Timer ها خاموش باشند روشن می‌شوند (و بالعکس). با روشن شدن هر Timer موج مربعی مربوط به آن روی پایه مربوطه ظاهر خواهد شد.