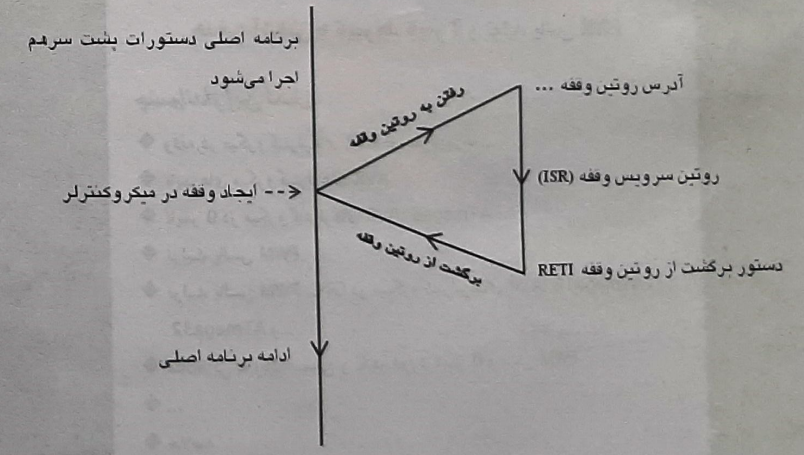
## ۵-۱: وقفه[1] در میکروکنترلرهای AVR توسط تایمر 0 ، تایمر 2 و...

میکروکنترلرها در حالت عادی دستورات برنامه‌ها را یکی، یکی، پشت سر هم، تا پایان برنامه اجرا می‌نمایند. ولی ممکن است توسط یک دستگاه ورودی- خروجی و یا از خارج از میکروکنترلر، تقاضای وقفه برنامه میکروکنترلر شود، تا میکروکنترلر برنامه دیگری به نام روتین سرویس وقفه ISR[2] را اجرا نماید. و پس از اتمام آن مجدداً به برنامه اصلی برگردد و کارش را ادامه دهد، در این حالت می‌گویند در میکروکنترلر وقفه رُخ داده است.

به‌عنوان مثال ممکن است درحالتی میکروکنترلر اطلاعات را از پورت D بخواند و در پورت B بنویسد، در ضمن پورت سری آن نیز اطلاعات دیگری را یک بیت، یک بیت دریافت کند. زمانی که پورت سری بیت‌های یک بایت اطلاعات مثلاً حرف A را دریافت کرد، باید در کار میکروکنترلر وقفه ایجاد کند تا میکروکنترلر موقتاً کار عادی خود را قطع نماید و اطلاعات دریافت شده در پورت سری را بردارد و سپس به برنامه اصلی برگردد و کار خود را ادامه دهد، یعنی از پورت D بخواند و بر پورت B بنویسد. به این ترتیب برنامه مطابق شکل (۵-۱) یک دستور، یک دستور ادامه می‌یابد و به محض وقوع وقفه برنامه جاری قطع می‌شود و برنامه مربوط به وقفه، یعنی خواندن از پورت سری شروع می‌شود و در پایان روتین سرویس وقفه، به برنامه اصلی برمی‌گردد و کار را ادامه می‌دهد.




*شکل (۵-۱) چگونگی ایجاد و اجرای روتین وقفه*

به‌عنوان مثال دیگر، ممکن است تایمر[3] میکروکنترلر شروع به کار کند و میکروکنترلر با اجرای دستوراتی در حلقه انتظار بماند و به محض اینکه تایمر پُر شد، تقاضای وقفه از میکروکنترلر می‌کند و

1- Interrupt
2- Interrupt Service Routine (ISR)
3- Timer



میکروکنترلر کار عادی خود را موقتاً قطع می‌کند و برنامه مربوط یا روتین سرویس وقفه مثلاً معکوس کردن پورت B را اجرا می‌کند و سپس به برنامه اصلی برمی‌گردد و منتظر وقفه بعدی می‌شود.

به این ترتیب دستگاه‌های جانبی مختلف ممکن است از میکروکنترلر تقاضای وقفه نماید و میکروکنترلر کار عادی خود را موقتاً قطع کند و برنامه یا روتین سرویس وقفه مربوط به آن دستگاه جانبی را اجرا کند و سپس به برنامه اصلی برگردد.

دستگاه‌های مختلف که می‌توانند از میکروکنترلرهای AVR تقاضای وقفه کنند عبارتند از: تایمر 0، تایمر 1*، تایمر 2*، پورت سری*، مبدل آنالوگ به دیجیتال ADC*، ... .

آدرس یا وکتور[1] روتین سرویس وقفه هر یک از دستگاه‌های فوق از محل بخصوص از حافظه برنامه Flash شروع می‌شود. به‌عنوان مثال روتین سرویس وقفه تایمر 0 در میکروکنترلر ATmega8:AVR از آدرس 0x009 هگزادسیمال حافظه شروع می‌شود و یا روتین سرویس وقفه مبدل ADC از آدرس 0x00E هگزادسیمال شروع می‌گردد. به این ترتیب تا دستگاهی در میکروکنترلر وقفه ایجاد می‌کند، میکروکنترلر به آدرس روتین سرویس وقفه مربوطه می‌رود و دستورات آن روتین را یکی یکی اجرا می‌کند و در انتها توسط دستور برگشت از روتین سرویس وقفه RETI به برنامه اصلی برمی‌گردد و کارش را ادامه می‌دهد. آدرس‌های روتین سرویس وقفه در میکروکنترلرهای AVR، آدرس‌های بخصوص هستند که توسط سازنده مشخص شده و به نام وکتور یا آدرس‌های روتین وقفه مطابق جدول (۵-۱) می‌باشد.

*جدول (۵-۱) آدرس** یا وکتور روتین سرویس وقفه دستگاه‌های جانبی میکروکنترلرهای AVR*

| دستگاه تقاضاکننده وقفه | آدرس یا وکتور روتین سرویس وقفه | | | |
|---|---|---|---|---|
| | میکروکنترلر ATmega8 :AVR | میکروکنترلر ATmega16 :AVR | میکروکنترلر ATmega32 :AVR | میکروکنترلر ATtiny :AVR |
| Reset*** | 0x000 | 0x000 | 0x000 | 0x000 |
| وقفه خارجی شماره 0 درپایه INT0 | 0x001 | 0x002 | 0x002 | 0x001 |
| وقفه خارجی شماره 1 در پایه INT1 | 0x002 | 0x004 | 0x004 | — |
| وقفه خارجی شماره 2 در پایه INT2 | — | 0x024 | 0x006 | — |
| تایمر 2 در Compare Match | 0x003 | 0x006 | 0x008 | — |
| تایمر 2 در سرریز Overflow | 0x004 | 0x008 | 0x00A | — |
| تایمر 1 در input capture | 0x005 | 0x00A | 0x00C | — |
| تایمر 1 در Compare Match A | 0x006 | 0x00C | 0x00E | 0x003 |

* در فصل‌های بعدی بحث می‌شود.
** آدرس یا وکتورهای کوچکتر دارای اولویت بالاتری هستند.
*** هر نوع Reset که در فصل (۷) بخش (۷-۲) بحث می‌شود.

1- Vector یا Address

| دستگاه تقاضاکننده وقفه | آدرس یا وکتور روتین سرویس وقفه | | | |
|---|---|---|---|---|
| | میکروکنترلر AVR:ATmega8 | میکروکنترلر AVR:ATmega16 | میکروکنترلر AVR:ATmega32 | میکروکنترلر AVR:ATtiny |
| تایمر 1 در Compare Match B | 0x007 | 0x00E | 0x010 | — |
| تایمر 1 در سرریز یا Overflow | 0x008 | 0x010 | 0x012 | 0x004 |
| تایمر 0 در سرریز یا Overflow | 0x009 | 0x012 | 0x016 | 0x005 |
| تایمر* 0 در Compare Match | ... | 0x026 | 0x014 | ... |
| پورت سری USART برای دریافت کامل یک بایت (RXC) | 0x00B | 0x016 | 0x01A | ... |
| ... | ... | ... | ... | ... |
| مبدل آنالوگ به دیجیتال ADC | 0x00E | 0x01C | 0x020 | 0x008 |
| ... | ... | ... | ... | ... |
| مقایسه‌کننده آنالوگ | 0x010 | 0x020 | 0x024 | 0x007 |
| ... | ... | ... | ... | ... |

در فصل‌های مربوط به این دستگاه‌های جانبی، در مورد وقفه‌های مربوطه و آدرس روتین سرویس آن‌ها بحث خواهد شد. باید دقت نمود که دستگاه‌های مذکور نمی‌توانند همیشه تقاضای وقفه از میکروکنترلر بنمایند، مگر اینکه ما توسط دستوراتی، وقفه آن‌ها را فعال کرده باشیم. به‌عنوان مثال ممکن است در برنامه‌ای ما میل داشته باشیم که فقط وقفه تایمر 0 فعال شود، یعنی موقعی که تایمر 0 پُر شد، تقاضای وقفه از میکروکنترلر بنماید و میکروکنترلر توسط دستورات روتین سرویس وقفه مثلاً پورت B را معکوس کند. در این حالت وقفه دستگاه‌های دیگری غیرفعال هستند. به‌طور پیش‌فرض وقفه تمام دستگاه‌ها غیرفعال هستند مگر اینکه ما آن‌ها را توسط دستوراتی فعال کنیم.

مسئله دیگر این است که اگر وقفه چند دستگاه فعال بود و دو دستگاه همزمان تقاضای وقفه از میکروکنترلر کردند، میکروکنترلر چگونه روتین یا برنامه‌های آن‌ها را اجرا می‌کند؟ در این حالت دستگاه‌هایی که در جدول (۵-۱) در بالای جدول قرار دارند نسبت به دستگاه‌های پایین جدول، اولویت دارند، یعنی اگر مثلاً وقفه خارجی INT0 و تایمر 0 همزمان تقاضای وقفه بنمایند، ابتدا روتین سرویس وقفه خارجی INT0 توسط میکروکنترلر اجرا می‌شود و سپس روتین سرویس وقفه تایمر 0 اجرا می‌گردد.

مطالب فوق با ذکر مثال‌های متعدد در فصل‌های مربوط به تایمر 0 ، تایمر 1 ، تایمر 2 ، مبدل آنالوگ به دیجیتال ADC ، پورت سری و... مورد بحث قرار خواهند گرفت. و در ادامه تایمر 0 و وقفه‌های آن نیز مورد بحث قرار می‌گیرد.

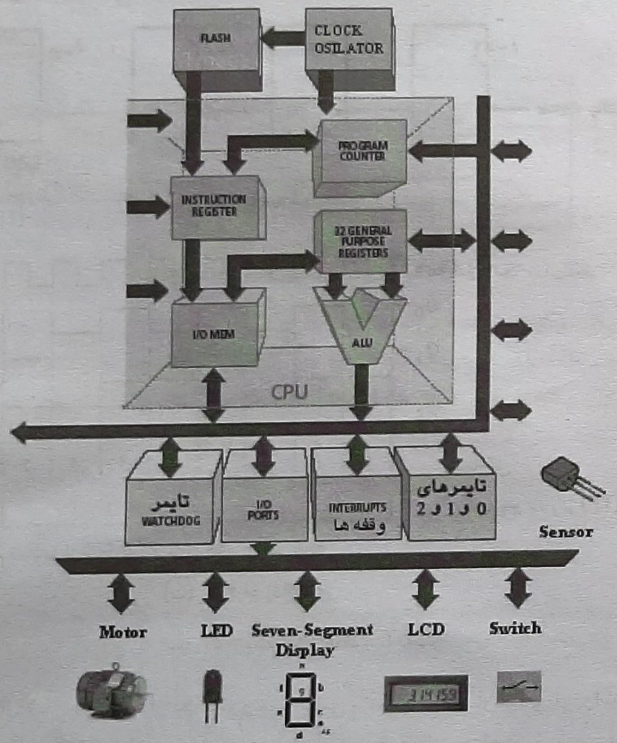
* در AVR:ATmega16 اولویت آنها بعد از مقایسه کننده آنالوگ است.



## ◆ ۵-۲: تایمرها در میکروکنترلرهای AVR

میکروکنترلرهای AVR دارای تعدادی تایمر به نام تایمر 0 ، تایمر 1 ، تایمر 2 و تایمر Watchdog به‌عنوان دستگاه‌های ورودی خروجی هستند (شکل (۵-۲)).

تایمرهای 0 ، 2 و Watchdog هشت بیتی و تایمر 1 شانزده بیتی هستند که هر یک دارای امکانات و ویژگی‌های خاص می‌باشند. در این فصل تایمر 0 و در فصل‌های بعدی تایمرهای 1 ، 2 و تایمر Watchdog مورد بحث قرار می‌گیرد.

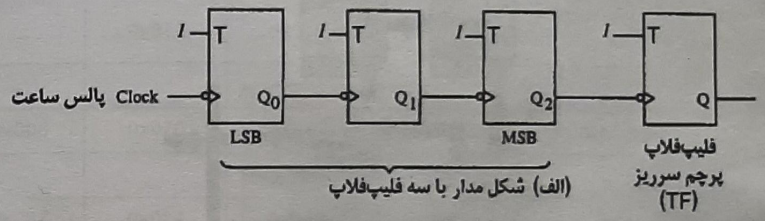



*شکل (۵-۲) تایمرهای 0 ، 1 ، 2 ، وقفه و تایمر Watchdog*

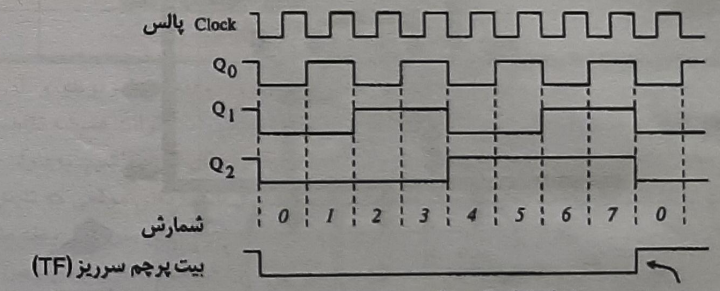
تایمر در حقیت یک کنتور است که پالس ورودی را می‌شمارد. درصورتی که پالس ورودی کنتور از میکروکنترلر تأمین شود، تایمر نام دارد، چون در این صورت زمان پالس ورودی کنتور مشخص است، لذا عدد شمارش شده در کنتور متناسب با زمان است، به عبارت دیگر کنتور زمان را اندازه‌گیری می‌کند.

یک نوع کنتور از یک سری فلیپ فلاپ T تشکیل می‌شود که هر فلیپ فلاپ T ، پالس ساعت ورودی خود را بر 2 تقسیم می‌کند (شکل (۵-۳)). در این کنتور پالس ساعت اصلی به ورودی اولین فلیپ فلاپ ($Q_0$) وارد می‌شود. در نتیجه فرکانس خروجی آن فلیپ فلاپ، نصف فرکانس پالس ساعت اصلی ورودی Clock می‌باشد. خروجی اولین فلیپ فلاپ، به‌عنوان پالس ساعت ورودی برای فلیپ فلاپ دوم ($Q_1$) به کار می‌رود، که در این صورت، فرکانس پالس خروجی فلیپ فلاپ دوم نیز، نصف فرکانس پالس ورودی آن می‌باشد ( $\frac{1}{4}$ پالس ساعت ورودی اصلی) و... .

پس خروجی آخرین فلیپ فلاپ تایمر n طبقه، فرکانس پالس ورودی خود را بر $2^n$ تقسیم می‌کند.




(الف) شکل مدار با سه فلیپ‌فلاپ




(ب) دیاگرام زمانبندی

*شکل (۵-۳) طرز کار کنتور یا تایمر*

به‌عنوان مثال شکل (۵-۳) مدار یک کنتور سه بیتی و دیاگرام زمان‌بندی آن را نشان می‌دهد. همانطور که از شکل مذکور مشاهده می‌شود، در لبه پایین رونده پالس ساعت، خروجی هر فلیپ فلاپ عوض می‌شود، در حقیقت خروجی فلیپ فلاپ‌های $Q_0$ ، $Q_1$ و $Q_2$ در هر لحظه تعداد پالس ساعت Clock ورودی را که بین 0 تا 7 می‌باشد، نشان می‌دهند.

علاوه بر این مشاهده می‌شود زمانی که تایمر پُر شد، یعنی به عدد 7 یا 111 باینری رسید، با پالس Clock بعدی، خروجی فلیپ فلاپ ها 0 می‌شوند، یعنی تایمر 0 می‌گردد و فلیپ فلاپ دیگری به نام فلیپ فلاپ پرچم TF مساوی 1 می‌گردد که نشانه پُر شدن تایمر است.

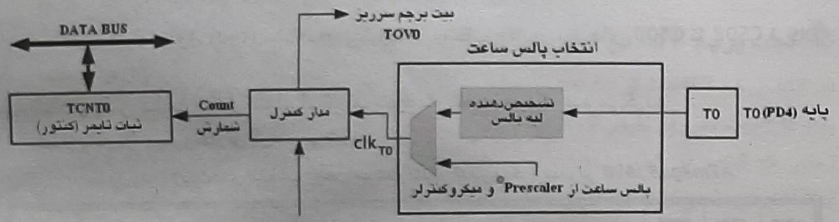


## ۵-۳: تایمر 0[1]

میکروکنترلرهای AVR، یک تایمر هشت بیتی به نام تایمر 0 دارند که پالس ساعت آن از میکروکنترلر و یا از خارج از میکروکنترلر (پایه T0) گرفته می‌شود و برخی از آن‌ها امکان تولید پالس PWM نیز دارند.*

## ۵-۴: تایمر 0 در میکروکنترلر AVR: ATmega8

ساختار اصولی تایمر 0 میکروکنترلرهای AVR: ATmega8 در شکل (۵-۴) نشان داده شده است.




*شکل (۵-۴)* ** *بلوک دیاگرام تایمر 0 در میکروکنترلر AVR: ATmega8*

همانطور که از شکل مذکور مشاهده می‌شود تایمر 0 دارای:

- یک ثبات تایمر TCNT0[2] است که با هر پالس ساعت تایمر $clk_{T0}$ یک شماره می‌اندازد و موقعی که به مقدار ماکزیمم MAX یعنی 11111111 (یا 0xFF هگزادسیمال) رسید، با پالس بعدی مقدار آن به پایین‌ترین[3] مقدار یعنی 0 برمی‌گردد و در این حالت بیت پرچم سرریز TOV0 برابر 1 می‌گردد.

با دستورات میکروکنترلر می‌توان به ثبات تایمر TCNT0 مقدار اولیه داد تا تایمر از آن به بعد شمارش را آغاز کند و تا ماکزیمم مقدار یعنی 0xFF ادامه دهد. شکل (۵-۵) ثبات تایمر 0 هشت بیتی TCNT0 را نشان می‌دهد.

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TCNT0 | TCNT0[7:0] | | | | | | | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W*** |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۵) ثبات تایمر 0 هشت بیتی TCNT0 در میکروکنترلر AVR: ATmega8*

در هر لحظه می‌توان مقدار تایمر را از ثبات TCNT0 خواند و یا مقدار جدیدی در آن نوشت.

---

* بعداً بحث می‌شود.

** برای واضح بودن شکل، ثبات‌های دیگر در شکل نشان داده نشده است.

*** قابل خواندن و نوشتن

1- Timer / Counter 0
2- Timer Counter Register 0 (TCNT0)
3- Bottom

## ◆ ثبات کنترل TCCR0 در میکروکنترلر AVR : ATmega8

فرکانس پالس ساعت تایمر 0 ، توسط بلوک انتخاب پالس ساعت (شکل (۵-۴)) و همچنین به‌وسیله بیت‌های انتخاب پالس ساعت CS00 تا CS02 ثبات کنترل تایمر 0 ، (TCCR0)[1] تعیین می‌شود (شکل (۵-۶)).

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TCCR0 | – | – | – | – | – | CS02 | CS01 | CS00 |
| | R | R | R | R | R* | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۶) ثبات کنترل تایمر 0 ، TCCR0 در میکروکنترلر AVR: ATmega8*

همانطور که از جدول (۵-۲) مشاهده می‌شود، با انتخاب مقادیر بیت‌های CS00 تا CS02 فرکانس پالس ساعت تایمر:

الف: از پالس ساعت میکروکنترلر گرفته می‌شود که تقسیم بر یک عدد می‌گردد.**

ب: از پایه T0 میکروکنترلر گرفته می‌شود.

*جدول (۵-۲) بیت‌های انتخاب پالس ساعت تایمر 0 در میکروکنترلر AVR: ATmega8*

| CS02 | CS01 | CS00 | توضیح |
|---|---|---|---|
| 0 | 0 | 0 | تایمر متوقف است |
| 0 | 0 | 1 | پالس ساعت تایمر همان پالس ساعت میکروکنترلر است |
| 0 | 1 | 0 | پالس ساعت تایمر برابر 1/8 پالس ساعت میکروکنترلر است |
| 0 | 1 | 1 | پالس ساعت تایمر برابر 1/64 پالس ساعت میکروکنترلر است |
| 1 | 0 | 0 | پالس ساعت تایمر برابر 1/256 پالس ساعت میکروکنترلر است |
| 1 | 0 | 1 | پالس ساعت تایمر برابر 1/1024 پالس ساعت میکروکنترلر است |
| 1 | 1 | 0 | در لبه پایین‌رونده پالس در پایه T0 ، کنتور یک شماره می‌اندازد |
| 1 | 1 | 1 | در لبه بالارونده پالس در پایه T0 ، کنتور یک شماره می‌اندازد |

همانطور که از جدول مذکور مشاهده می‌شود، زمانی که بیت‌های CS00 تا CS02 برابر 000 هستند، تایمر متوقف است و نمی‌شمارد، به محض اینکه تغییری در این بیت‌ها انجام شود، تایمر با فرکانس مربوطه شروع به شمارش می‌کند.

در ضمن مشاهده می‌شود به ازای بیت‌های CS00 تا CS02 برابر 110 ، تایمر به‌عنوان کنتور با لبه پایین رونده پالس ورودی، در پایه T0 می‌شمارد و به ازای CS00 تا CS02 مساوی 111 ، تایمر با لبه بالا رونده پالس ورودی، در پایه T0 می‌شمارد.

* فقط قابل خواندن

** عملیات پالس ساعت میکروکنترلر تقسیم بر یک عدد، توسط Prescaler انجام می‌شود که در ادامه این فصل راجع به آن بحث خواهد شد.

1- Timer Counter Control Register (TCCR0)



## ◆ وقفه تایمر 0 در میکروکنترلر AVR: ATmega8

موقعی که تایمر 0 ، برابر 11111111 (یا 0xFF هگزادسیمال) گردید، با پالس بعدی تایمر 0 مساوی 0 می‌شود که در این حالت بیت پرچم سرریز[1] TOV0 در ثبات پرچم تایمر[2] TIFR برابر 1 می‌گردد. درصورتی که بیت فعال کردن وقفه کلی I در ثبات SREG برابر 1 باشد و بیت فعال کردن وقفه سرریز تایمر[3] 0 یعنی TOIE0 نیز در ثبات* TIMSK برابر 1 باشد، آن وقت روتین سرویس وقفه اجرا می‌شود. پس از اجرای روتین سرویس وقفه بیت پرچم سرریز TOV0 برابر 0 می‌گردد. البته با نوشتن 1 با دستورات میکروکنترلر در این بیت نیز، بیت پرچم سرریز مذکور Clear می‌شود.

## ◆ ثبات پرچم TIFR در میکروکنترلر AVR: ATmega8

در ثبات پرچم TIFR (شکل (۵-۷)) بیت‌های پرچم تایمرهای مختلف قرار دارند که در:

**بیت 0 – بیت سرریز تایمر 0 ، TOV0**

زمانی که کنتور پُر شد، و مقدار آن از 11111111 به 0 رسید، آن وقت بیت سرریز تایمر 0 یعنی TOV0 برابر 1 می‌گردد، که در صورت اجرای روتین وقفه، این بیت به‌طور اتوماتیک 0 می‌گردد. البته با دستورات میکروکنترلر می‌توان در این بیت 1 نوشت تا این بیت Clear شود.

**بیت‌های 1 تا 7**

مربوط به بقیه تایمرها است که در فصل‌های مربوطه درباره آن‌ها بحث می‌شود.

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TIFR | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | – | TOV0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۷) ثبات پرچم تایمرها TIFR در میکروکنترلر AVR: ATmega8*

## ◆ ثبات فعال کردن وقفه تایمرها TIMSK[4] در میکروکنترلر AVR: ATmega8

بیت‌های این ثبات (شکل (۵-۸)) برای فعال کردن وقفه تایمرها است.

**بیت 0 – بیت فعال کردن وقفه تایمر 0 ، TOIE0**

اگر با دستورات میکروکنترلر این بیت را 1 کنیم وقفه سرریز تایمر 0 فعال می‌شود، که در صورت وجود سرریز، یعنی با 1 شدن بیت سرریز پرچم TOV0 در ثبات پرچم TIFR ، روتین سرویس وقفه اجرا می‌شود.

**بیت‌های 1 تا 7 :**

مربوط به بقیه تایمرها است که در فصل‌های مربوطه راجع به آن‌ها بحث خواهد شد.

* در ادامه بحث می‌شود.

1- Timer/Countor0 Overflow Flag (TOV0)
2- Timer/Counter Interrupt Flag Register (TIFR)
3- Timer/Counter0 Overflow Interrupt Enable (TOIE0)
4- Timer/Counter Interrupt Mask Register (TIMSK)

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | – | TOIE0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۸) ثبات فعال کردن وقفه تایمرها TIMSK در میکروکنترلر AVR: ATmega8*

**مثال**

دستوراتی بنویسید که پالس ساعت تایمر 0 برابر پالس ساعت میکروکنترلر ATmega8 شود و در ضمن وقفه نیز فعال گردد.

**حل**

**الف:** برای اینکه پالس ساعت تایمر همان پالس ساعت میکروکنترلر باشد، باید مطابق جدول (۵-۲) بیت CS00 در ثبات کنترل TCCR0 شکل (۵-۶) برابر 1 گردد.

لذا دستورات زیر را می‌نویسیم:

```
ldi r16 , 1<<CS00          ; (1)
out TCCR0 , r16            ; (2)
```

◆ دستور (۱): مقدار CS00 را برابر 1 می‌کند و در کم‌ارزش‌ترین بیت ثبات r16 قرار می‌دهد.

◆ دستور (۲): مقدار r16 را در ثبات کنترل تایمر TCCR0 قرار می‌دهد.

در نتیجه بیت 0 ثبات TCCR0 برابر 1 می‌گردد.

البته دستورات فوق را به‌صورت زیر نیز می‌توان نوشت:

```
ldi r16 , 0b00000001       ; (3)
out TCCR0 , r16            ; (4)
```

در نتیجه بیت 0 ثبات TCCR0 یعنی CS00 برابر 1 می‌گردد، لذا پالس ساعت تایمر همان پالس ساعت میکروکنترلر می‌گردد و تایمر شروع به کار می‌کند.

**ب:** برای فعال کردن وقفه تایمر 0 باید:

● بیت فعال کردن وقفه T0IE0 در ثبات فعال کردن وقفه تایمرها TIMSK (شکل (۵-۸)) برابر 1 گردد.

برای این کار دستورات زیر را می‌نویسیم:

```
ldi r16 , 1 << TOIE0       ; (1)
out TIMSK , r16            ; (2)
```

یا

```
ldi r16 , 0b00000001       ; (3)
out TIMSK , r16            ; (4)
```

دستورات (۱) و (۲) با دستورات (۳) و (۴) معادل هستند و باعث می‌شوند که:

بیت 0 فعال کردن وقفه تایمر 0 یعنی TOIE0 در ثبات TIMSK برابر 1 گردد.



**نکته:** *اختلاف دستورات (۱) و (۲) با دستورات (۳) و (۴) این است که در دستورات (۱) و (۲) محل بیت فعال‌ساز وقفه تایمر 0 یعنی TOIE0 هر محلی در ثبات TIMSK که باشد، مترجم اسمبلی محل آن را می‌شناسد و مقدار TOIE0 را برابر 1 می‌کند و در ثبات TIMSK قرار می‌دهد. به‌عنوان مثال محل بیت TOIE0:*

- *در میکروکنترلر ATmega8 بیت 0 ثبات TIMSK است.*
- *در میکروکنترلر ATtiny15 بیت 1 ثبات TIMSK است.*
- *در میکروکنترلرهای ATmega16 ، ATmega32 و... بیت 0 ثبات TIMSK است.*

*به این ترتیب دستورات (۱) و (۲) برای هر یک از میکروکنترلرهای AVR قابل اجرا می‌باشند.*

*درصورتی که دستورات (۳) و (۴) فقط در برنامه‌های میکروکنترلرهای ATmega8 ، ATmega16 ، ATmega32 و... صحیح می‌باشد ولی در برنامه‌های میکروکنترلر ATtiny15 و... اشتباه است.*

*لذا بهتر است دستورات به شکل استاندارد فرم (۱) و (۲) در برنامه‌ها باشند تا برنامه با میکروکنترلرهای مختلف قابل اجرا باشند*

● علاوه بر این باید بیت فعال کردن وقفه کلی I در ثبات SREG نیز برابر 1 گردد، که برای این کار کافیست دستور زیر را بنویسیم:

```
sei
```

در نتیجه وقفه سرریز تایمر 0 فعال می‌شود.

**نکته:** *تمام ثبات‌های تایمر 0 و دستگاه‌های ورودی خروجی فقط با دستورات IN و OUT کار می‌کنند.*

**نکته:** *در برنامه‌های تایمر 0 ، در ابتدا باید به ثبات‌های تایمر 0 و... مقدار اولیه داده شود.*

**نکته:** *به‌محض قرار دادن مقدارهای CS00 ، CS01 و CS02 در ثبات کنترل TCCR0 ، علاوه بر اینکه پالس ساعت تایمر انتخاب می‌شود، تایمر 0 نیز شروع به کار می‌کند.*

**نکته:** *برای ایجاد وقفه در تایمر 0 ، علاوه بر اینکه بیت فعال‌ساز وقفه TOIE0 در ثبات TIMSK باید برابر 1 گردد، بیت فعال‌ساز وقفه کلی I در ثبات SREG نیز توسط دستور Sei باید مساوی 1 شود.*

## ◆ ۵-۵: ساختار برنامه اسمبلی میکروکنترلرهای AVR با سابروتین و روتین سرویس وقفه ISR

درصورتی‌که بخواهیم در برنامه میکروکنترلرهای AVR از سابروتین و همچنین روتین سرویس وقفه ISR استفاده کنیم، باید برنامه مطابق شکل (۵-۹) باشد. در این صورت در:

**بخش A:**

نوع میکروکنترلر و نام فایل مشخص می‌شود.

**بخش B:**

فایل Header File و آدرس برنامه اصلی و آدرس یا وکتور روتین سرویس وقفه مشخص می‌گردد. به این ترتیب:

- عبارت (۱): نام فایل Header File است.
- عبارت (۲): در آدرس 00 ، پرش به ابتدای برنامه، یعنی به بخش C با برچسب reset می‌باشد.
- عبارت (۳): در آدرس روتین وقفه، به‌عنوان مثال در آدرس 0x009 ، پرش به روتین سرویس وقفه، یعنی پرش به قسمتی از بخش E که برچسب آن ISR است، انجام می‌شود.

**بخش C:**

با دستورات (۴) تا (۷) مطابق فصل (۳) بخش (۳-۱۴)، به اشاره‌گر پُشته (SP) مقدار اولیه داده می‌شود تا محل حافظه پُشته در انتهای حافظه SRAM تعیین گردد. همچنین در این بخش به دستگاه‌های ورودی خروجی مانند پورت‌های B ، C ، D و... مقدار اولیه داده می‌شود. علاوه بر این به تایمر 0 ، تایمر 1 ، تایمر 2 و... مبدل آنالوگ به دیجیتال ADC ، ... نیز اگر به کار گرفته شوند، مقدار اولیه داده می‌شود. در این بخش باید وقفه کلی میکروکنترلر توسط 1 کردن بیت I در ثبات SREG فعال گردد.

**بخش D:**

دستور CALL برای فراخوانی سابروتین* و سایر دستورات موردنیاز نوشته می‌شود و عملیات تکرار می‌گردد تا وقفه در میکروکنترلر ایجاد گردد.

```
;          nemuneh3.asm
;*************************************************
;A Section
;AVR number        :atmega8
;File name     :nemuneh3.asm
;Description  :This is a sample assembly with
;              subroutine & Interrupt
;*************************************************
;B Section
.include "m8def.inc"                ;(1)This header file
;                ; is needed for all avr
```

* اگر سابروتین داشته باشیم.



```
;                ; atmega8 assembly language
;Interrupt vectors table
.org    0x00            ;Reset address
   rjmp Reset                       ;(2)Jump to reset
;
.org   0x009
   rjmp ISR                         ;(3) Interrupt vector,jump
;               ;to interrupt service routin
;
;*************************************************
;C Section
;*******Main program entry point on reset
reset:
;*******Initialization of the stack pointer
   ldi r16,low (RAMEND)             ;(4)
   out SPL,r16                      ;(5)
   ldi r16,high (RAMEND)            ;(6)
   out SPH,r16                      ;(7)
;*******Input output ports initialization
;PORTB initialization (if it is used)
;PORTC initialization (if it is used)
;PORTD initialization (if it is used)
;*******Initialize others I/O devices like:
; timer0, timer1, timer2, watchdog timer,
; external interrupt,ADC, Analog comparator...
;  (if they are used).
;********golbal interrupt enable
;
;*************************************************
;D Section
;*****loopforever
loopforever:
;   Insert instruction here
   rjmp loopforever                 ;(8)
;*************************************************
;E Section
;*******Write subroutine or proccedrure here
SUB1:
;   Insert instructions
   RET                              ;(9)
;*******Interrupt service routine
ISR:
;   Insert instructions here
   RETI                             ;(10)
;*************************************************
```

*شکل (۵-۹) نمونه برنامه اسمبلی میکروکنترلرهای AVR با سابروتین و روتین سرویس وقفه*

به محض وقوع وقفه، موقتاً اجرای دستورات بخش D متوقف می‌شود و میکروکنترلر به آدرس روتین سرویس وقفه، مثلاً 0x009 در بخش B می‌رود که در آنجا به وسیله دستور (۳) به قسمت ISR بخش E پرش می‌کند.

**بخش E :**

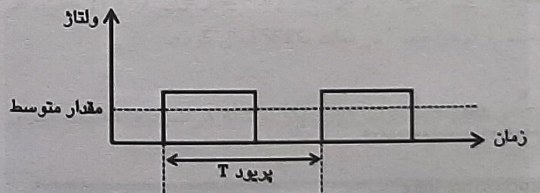
در قسمتی که برچسب آن ISR است، دستورات روتین سرویس وقفه ISR نوشته می‌شود و در انتهای روتین سرویس وقفه، دستور (۱۰) برگشت از روتین سرویس وقفه به محلی از بخش D است که وقفه رُخ داده است.

به این ترتیب هر موقع که وقفه رُخ می‌دهد. میکروکنترلر به روتین سرویس وقفه پرش می‌کند و آن را اجرا می‌نماید، سپس به محل اولیه برنامه اصلی برمی‌گردد. برای روشن شدن مطلب مثال‌هایی در این مورد خواهیم داشت.

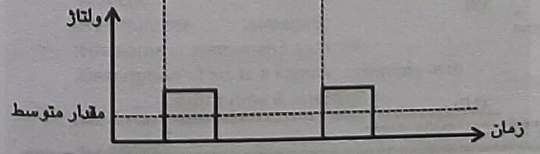
**نکته:** *در برنامه مذکور می‌توان بخش E را قبل از بخش C نوشت.*

## ◆ ۵-۶: طرز تولید پالس PWM[1] برای کنترل موتورها، سیستم‌های سوئیچینگ و...

پالس PWM یعنی پالسی که عرض[2] آن را می‌توان مدوله کرد[3] و یا تغییر داد. شکل‌های (۵-۱۰) و (۵-۱۱) نمونه‌ای از پالس PWM را نشان می‌دهد.




*شکل (۵-۱۰) نمونه‌ای از پالس PWM*




*شکل (۵-۱۱) نمونه‌ای از پالس PWM*

1- Pulse Width Modulation (PWM)
2- Width
3- Modulation



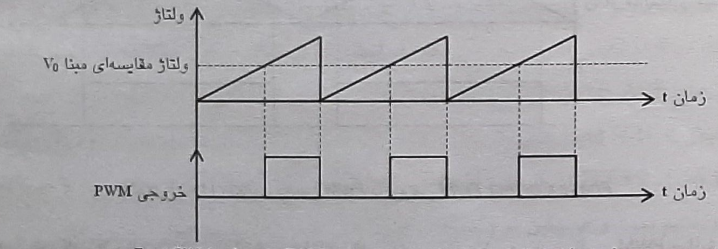
در شکل‌های مذکور، پریود پالس T ثابت می‌باشد ولی زمانی که پالس 1 است در آن‌ها متفاوت می‌باشد. و یا عرض پالس در شکل (۵-۱۰) بیشتر از شکل (۵-۱۱) می‌باشد، در نتیجه مقدار متوسط ولتاژ در شکل (۵-۱۰) بیش از شکل (۵-۱۱) است. اگر پالس‌های شکل‌های (۵-۱۰) و (۵-۱۱) را به موتوری متصل کنیم، چون پالس شکل (۵-۱۰) عرض بیشتری نسبت به شکل (۵-۱۱) دارد و مقدار متوسط ولتاژ آن بیش از مقدار متوسط شکل (۵-۱۱) است، لذا به موتور مقدار انرژی بیشتری می‌رسد و سرعت و توان آن بیشتر می‌شود. به این ترتیب با پالس PWM می‌توان سرعت موتور را کنترل نمود. علاوه بر این، با پالس PWM یا با تغییر عرض پالس، می‌توان زمانی که یک سوئیچ بسته یا باز است را نیز، تغییر داد که این موضوع کاربرد فراوانی در رگولاتورها، رکتیفایرها[1]، دیمرها، کنترل سرعت موتور... دارد.

برای تولید پالس PWM دو روش وجود دارد:

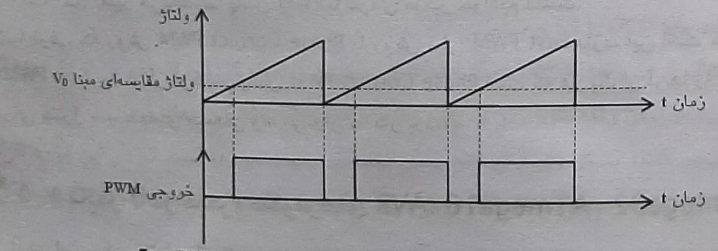
### ◆ الف: روش Fast PWM

در این روش برای تولید پالس PWM ، می‌توان:

- سیگنال دندانه‌اره‌ای مطابق شکل‌های ((۵-۱۲) و (۵-۱۳)) تولید نمود و مداری طراحی کرد که: زمانی که ولتاژ دندانه‌اره‌ای بالا می‌رود، هر وقت این ولتاژ، مساوی ولتاژ مقایسه مبنایی مانند $V_0$ شد، خروجی پالس PWM برابر 1 و هنگامی که ولتاژ دندانه‌اره‌ای برابر 0 شد، خروجی پالس PWM ، مساوی 0 گردد (شکل‌های (۵-۱۲) و (۵-۱۳)). به این ترتیب با تغییر ولتاژ مقایسه‌ای مبنا $V_0$ ، عرض پالس PWM مطابق شکل‌های مذکور تغییر می‌یابد، لذا پالس PWM با عرض متغیر خواهیم داشت.




*شکل (۵-۱۲) نمونه‌ای از تولید پالس PWM با روش Fast PWM*



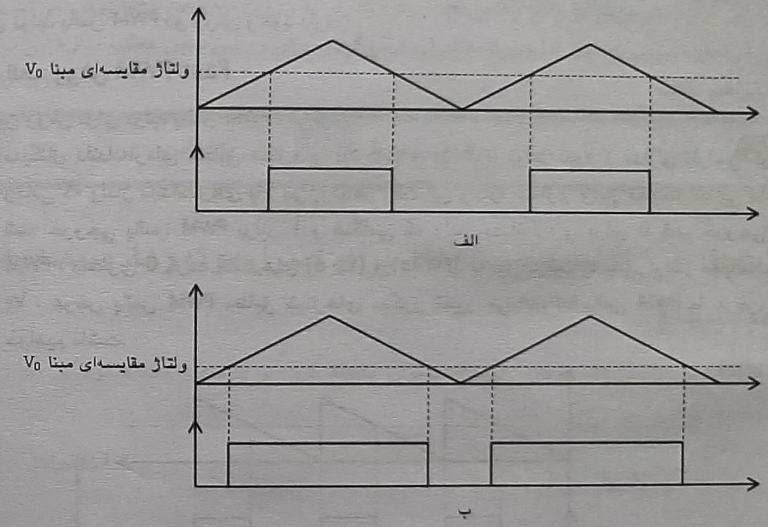

*شکل (۵-۱۳) نمونه‌ای از تولید پالس PWM با روش Fast PWM*

1- Rectifier

### ◆ ب: روش Phase Correct PWM

در این روش به جای تولید موج دندانه‌اره‌ای، یک موج مثلثی تولید می‌کنیم و مدار را طوری طراحی می‌کنیم که:

- در زمانی که ولتاژ مثلثی بالا می‌رود، اگر این ولتاژ برابر ولتاژ مقایسه‌ای مبنا $V_0$ شد، خروجی پالس PWM برابر 1 شود.
- در زمانی که ولتاژ مثلثی پایین می‌آید، هرگاه این ولتاژ مساوی ولتاژ مقایسه‌ای مبنا $V_0$ شد، خروجی پالس PWM مساوی 0 شود.




*شکل (۵-۱۴) تولید پالس PWM با روش Phase Correct PWM*

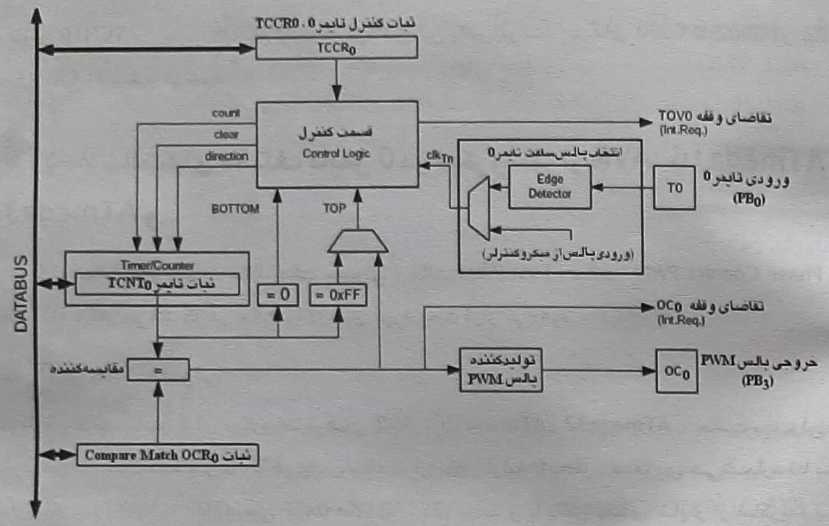
همانطورکه ملاحظه می‌شود، با تغییر ولتاژ مقایسه‌ای مبنا $V_0$ ، عرض پالس PWM مطابق شکل‌های (۵-۱۴- الف- ب) تغییر می‌یابد، لذا پالس PWM با عرض متغیر خواهیم داشت.

تنها فرقی که روش Phase Correct PWM با روش قبلی Fast PWM دارد، این است که فرکانس پالس Fast PWM بیشتر از فرکانس پالس Phase Correct PWM است که هر یک از آن‌ها را متناسب با نیاز، برای کنترل سیستم‌های صنعتی و... می‌توان به کار برد.

## ◆ ۵-۷: تایمر 0 در میکروکنترلرهای AVR: ATmega16 ، ATmega32 ...

تایمر 0 در این میکروکنترلر یک تایمر هشت بیتی است که در حالت عادی مانند یک تایمر کار می‌کند، علاوه بر این می‌تواند پالس PWM در خروجی OC0 (پایه PB3) تولید کند.






*شکل (۵-۱۵) بلوک دیاگرام تایمر 0 ، با امکانات تولید پالس PWM در میکروکنترلر AVR: ATmega16 ، ATmega32 و...*

پالس ساعت تایمر 0 از میکروکنترلر یا از خارج از میکروکنترلر (پایه T0) گرفته می‌شود. شکل (۵-۱۵) بلوک دیاگرام تایمر 0 را نشان می‌دهد. همانطورکه از شکل مذکور ملاحظه می‌شود، تایمر 0 در میکروکنترلرهای AVR: ATmega16 ، ATmega32 و... دارای ثبات‌های زیر است.

۱. ثبات تایمر TCNT0 (شکل‌های (۵-۱۵) و (۵-۲۰)) که مقدار شمارش تایمر را در خود نگه می‌دارد.

۲. با ثبات کنترل TCCR0 (شکل‌های (۵-۱۵) و (۵-۲۳)) می‌توان پالس ساعت تایمر را، با فرکانس‌های مختلف تنظیم نمود و همچنین نوع سیگنال PWM خروجی را مشخص کرد.

۳. با قرار دادن مقداری در ثبات OCR0 (شکل‌های (۵-۱۵) و (۵-۲۱)) می‌توان عرض پالس خروجی PWM در پایه OC0 را تنظیم نمود.

۴. ثبات فعال کردن وقفه TIMSK (شکل (۵-۲۵)) مخصوص فعال کردن وقفه‌های تایمرها است که در این مورد می‌توان وقفه سرریز و همچنین وقفه Compare Match را فعال کرد.

۵. ثبات پرچم TIFR (شکل (۵-۲۴)) مخصوص بیت‌های پرچم تایمرها است که در این مورد می‌توان بیت پرچم سرریز تایمر 0، (TOV0) و همچنین بیت پرچم Compare Match ، (OCF0) را بررسی نمود.

در ذیل قسمت‌های مختلف تایمر مذکور و کار آن‌ها را مورد بررسی قرار می‌دهیم. قسمت اصلی تایمر 0 ، کنتور هشت بیتی TCNT0 است که می‌تواند به‌صورت صعودی یا نزولی به شمارد. پالسی که

به تایمر TCNT0 می‌آید $clk_{I/O}$ نام دارد که فرکانس این پالس توسط بیت‌های CS00 تا CS02 در ثبات کنترل TCCR0 تنظیم می‌شود. *

## ◆ ۵-۸: حالت‌های مختلف تایمر 0 در میکروکنترلر AVR: ATmega16، ATmega32 و...

تایمر 0 در حالت‌های مختلف مانند حالت معمولی[1]، حالت Fast PWM[2]، حالت Phase Correct PWM[3] و حالت CTC[4] کار می‌کند که هر حالت را در ذیل مورد بحث قرار می‌دهیم.

### تایمر 0 در حالت معمولی

ساده‌ترین حالت تایمر 0 در میکروکنترلرهای AVR : ATmega16، ATmega32 ... حالت معمولی و پیش‌فرض آن است که تایمر 0، با هر پالس ساعت، از مقدار اولیه 0 به‌طور صعودی می‌شمارد تا به مقدار ماکزیمم (MAX یا TOP) یعنی 0xFF هگزا (یا 255) برسد و با پالس بعدی، تایمر از ماکزیمم به Bottom یعنی به 0 می‌رود که در این صورت بیت پرچم سرریز TOV0 در ثبات پرچم TIFR (شکل (۵-۲۴)) برابر 1 می‌گردد و تایمر مجدداً از 0 به‌طور صعودی می‌شمارد و بالا می‌رود.

البته می‌توان مقداری در ثبات تایمر TCNT0 به‌عنوان مقدار اولیه قرار داد تا تایمر از آن به بعد به‌طور صعودی به شمارد.

برای اینکه تایمر در حالت معمولی کار کند، کافیست در ثبات کنترل TCCR0 (شکل (۵-۲۳))، فقط بیت‌های CS00، CS01 و CS02 را برای تعیین پالس ساعت، مقدار اولیه داد و بقیه بیت‌های آن را در حالت اولیه 0 رها کرد.

**مثال**

دستورات زیر:

```
ldi r16 , (1<<CS00)          ; (1)
out TCCR0 , r16              ; (2)
        یا
ldi r16 , 0b00000001         ; (3)
out TCCR0 , r16              ; (4)
```

* مطابق جدول (۵-۴) که بعداً بحث می‌شود.

1- Normal Mode
2- Fast PWM Mode
3- Phase Correct PWM Mode
4- Clear Timer on Compare Match (CTC) Mode



◆ دستورات (۱) و (۲): معادل دستورات (۳) و (۴) هستند و باعث می‌شوند که، بیت CS00 در ثبات کنترل TCCR0برابر 1 شود، در نتیجه تایمر 0 در حالت معمولی کار کند و پالس ساعت آن مطابق جدول (۵-۴) برابر پالس ساعت میکروکنترلر AVR: ATmega16 و ATmega32 باشد.

**مثال**

دستورات زیر:

```
ldi r16 , (1<<CS00) 1 (1<<CS02)   ; (1)
out TCCR0 , r16                   ; (2)
        یا
ldi r16 , 0b00000101              ; (3)
out TCCR0 , r16                   ; (4)
```

◆ دستورات (۱) و (۲): معادل دستورات (۳) و (۴) هستند و باعث می‌شوند که تایمر 0 در حالت معمولی کار کند و پالس ساعت تایمر مطابق جدول (۵-۴) برابر 1/1024 پالس ساعت میکروکنترلر AVR: ATmega16، ATmega32 و... شود.

**نکته:** *بعد از انتخاب پالس ساعت در ثبات کنترل TCCR0 توسط دستورات فوق، تایمر شروع به شمارش می‌کند.*

**مثال**

برنامه‌ای در میکروکنترلر AVR: ATmega8 , ATmega16 ، ATmega32 بنویسید که هر بار تایمر 0 پُر شد با استفاده از وقفه سرریز، بیت 0 پورت B معکوس شود.

**حل**

برنامه آن مطابق شکل (۵-۱۶) می‌باشد که براساس ساختار برنامه شکل (۵-۹) تهیه شده است. در برنامه مذکور در:

**بخش A :**

نوع میکروکنترلر و نام فایل ذکر شده است.

**بخش B :**

در این بخش آدرس برنامه اصلی و آدرس روتین سرویس وقفه و دستورات مربوطه قرار دارد.

در این بخش فایل Header File میکروکنترلر AVR : ATmega16 به نام "m16def.inc" آورده شده است که اگر ATmega8 یا ATmega32 باشدفایل مذکور به ترتیب "m8def.inc" یا "m32def.inc" می‌باشد.

◆عبارت (۱): آدرس اصلی برنامه که 00 است را نشان می‌دهد و دستور پرش به بخش C که برچسب Reset دارا می‌باشد، است.

◆عبارت (۲): آدرس روتین سرویس وقفه سرریز که 0x012 در میکروکنترلر AVR:ATmega16 است که اگر AVR:ATmega8, ATmega32 باشد آدرس آن‌ها به ترتیب 0x016 یا 0x009 می‌باشد* و دستور پرش به آدرس روتین سرویس وقفه به بخش E آورده شده است.

**بخش C :**

در این بخش به ثبات اشاره‌گر پُشته (SP) ، مقدار اولیه داده می‌شود، وقفه سرریز تایمر 0 ، فعال می‌گردد و پورت B خروجی می‌شود.

◆دستورات (۳) تا (۶): به ثبات اشاره‌گر پُشته (SP) مقدار اولیه می‌دهد**.

◆دستورات (۷) و (۸): بیت 0 ثبات کنترل تایمر 0 ، TCCR0 ، یعنی CS00 را برابر 1 می‌کنند، در نتیجه مطابق جدول (۵-۴) پالس ساعت تایمر، همان پالس ساعت میکروکنترلر می‌شود و تایمر شروع به کار می‌کند.

◆دستورات (۹) و (۱۰): باعث می‌شوند که مقدار 1 در بیت فعال‌کننده وقفه تایمر 0 ، یعنی TOIE0 نوشته شود، در نتیجه وقفه سرریز تایمر 0 فعال شود.

◆دستورات (۱۱) و (۱۲): پورت B را خروجی می‌کنند.

◆دستور (۱۳): بیت وقفه کلی I در ثبات SREG را فعال می‌کند در نتیجه وقفه‌ها می‌توانند فعال شوند.

**بخش D :**

◆دستور (۱۴): میکروکنترلر منتظر می‌ماند تا تایمر 0 ، پُر شود و بیت پرچم سرریز تایمر TOV0 برابر 1 گردد. در این حالت برنامه به آدرس 012 روتین سرویس وقفه در بخش B پرش می‌کند. و در این آدرس دستور پرش باعث می‌شود که پرش به روتین سرویس وقفه در بخش E انجام شود.

**بخش E :**

روتین سرویس وقفه است که باعث می‌شود بیت 0 پورت B معکوس شود. در این صورت:

◆دستور (۱۵): بیت 0 ثبات r19 را برابر 1 می‌کند.

◆دستور (۱۶): پورت B را می‌خواند و در ثبات r16 قرار می‌دهد.

◆دستور (۱۷): محتوای ثبات r19 و r16 را XOR می‌کند و نتیجه را در ثبات r16 قرار می‌دهد و چون بیت 0 ثبات r19 برابر 1 است، پس بیت 0 ثبات r16 معکوس می‌شود***.

* به جدول آدرس روتین وقفه صفحه ۱۶۰ مراجعه فرمایید.
** در فصل ۳ بخش (۳-۱۴) بحث شده است.
*** اصول گیت XOR



◆ دستور (۱۸): محتوای ثبات r16 را که بیت 0 آن معکوس شده، مجدداً در پورت B قرار می‌دهد. به این ترتیب بیت 0 پورت B معکوس می‌شود.*

دستور (۱۹): برگشت از روتین سرویس وقفه است، یعنی به بخش D برمی‌گردد و توسط دستور (۱۴) منتظر وقفه بعدی می‌شود و عملیات تکرار می‌گردد.

این برنامه در سیمولاتور AVR Studio تست گردیده و همانطور که از شکل (۵-۱۶- ج) ملاحظه می‌شود، با اجرای دستورات میکروکنترلر به ثبات‌های تایمر 0 و پورت B مقدارهای موردنظر داده شده است. علاوه بر این فایل با پسوند HEX برنامه نیز توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

```
;          timer0-interrupt-atmega16.asm
;***************************************************
;A Section
;* Number          : atmega16
;* File Name       : "timer0-interrupt-atmega16.asm"
;* Title           : Setup and Use Timer0
;***************************************************
;B Section
.include "m16def.inc"
; Interrupt Vectore Table
.org    0x00                    ;1- Reset-Address
` rjmp Reset

.org    0x012                   ;2- Timer overflow vector
  rjmp  ISR_TOV0

;***************************************************
;C Section
; *** Begin of Program Execution ***
Reset:
; Initialization of the Stack Pointer
   ldi   r16,low(RAMEND)        ;3- initialize
   out   SPL,r16                ;4- SPL
   ldi   r16,high(RAMEND)       ;5- initialize
   out   SPH,r16                ;6- SPH
```

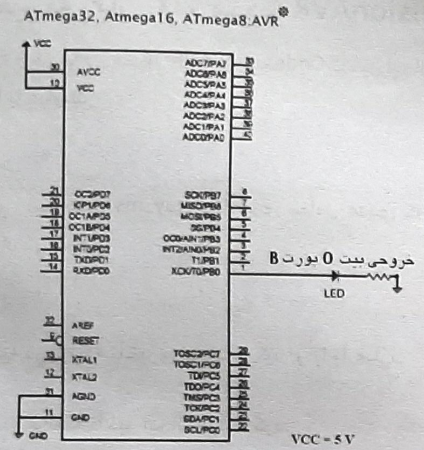
* خروجی این بیت میکروکنترلر را می‌توان با اُسیلوسکوپ مشاهده نمود.

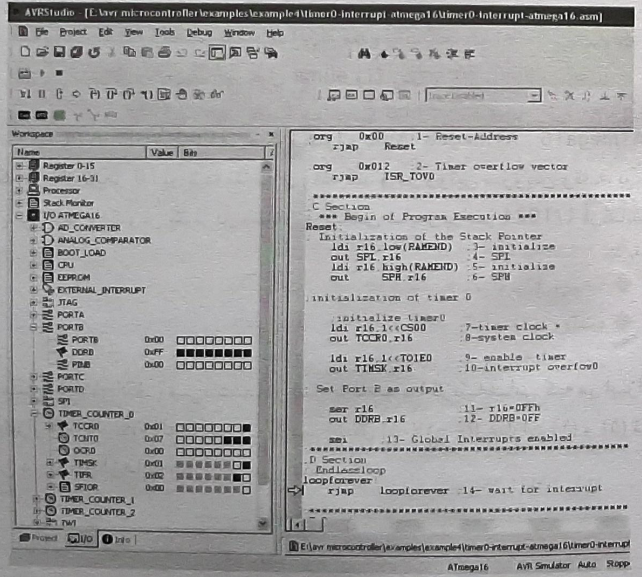
```
;initialization of timer 0

    ;initialize timer0
    ldi r16,1<<CS00          ;7-timer clock =
    out TCCR0,r16            ;8-system clock

    ldi r16,1<<TOIE0         ;9- enable  timer
    out TIMSK,r16            ;10-interrupt overfow0
; Set Port B as output
    ser r16                  ;11- r16=0FFh
    out DDRB,r16             ;12- DDRB=0FF
    sei                      ;13- Global Interrupts enabled
;*****************************************************
;D Section
; Endlessloop
loopforever:
    rjmp  loopforever        ;14- wait for interrupt
;*****************************************************
;E Section
;Interrupt service routine ISR_TOV0,
; is executed each time that overflow occured,
; that is TOV0 become 1, complement PB0 each
; time
ISR_TOV0:
    ldi r19,1                ;15- r19=1
    in  r16,PORTB            ;16- r16= PORTB
    eor r16,r19              ;17- complement
    ;change PB0 in 256 clock (for timer0 overflow)
;plus,time of executing of ISR .
    out PORTB,r16            ;18-PORTB=r16
    reti                     ;19- return from interrupt
```

*الف: برنامه*



*ب: شکل میکروکنترلر*




*ج: نتیجه تست برنامه timer0-interrupt-atmega16.asm در سیمولاتور AVR Studio*

*شکل (۵-۱۶) برنامه تایمر 0 در میکروکنترلر AVR : ATmega16, ATmaga8 و ATmega32*

*برای اینکه هر بار تایمر 0 پُر شد، با استفاده از وقفه سرریز، بیت 0 پورت B معکوس شود.*

* شماره پایه‌های آن مطابق شکل (۲-۱۰) فصل (۲) می‌باشد.

** در توضیح برنامه اختلافات آن‌ها بحث شده است.

## ◆ ۵-۹: ایجاد تأخیر به زبان C در محیط CodeVisionAVR

برای اینکه در برنامه‌های به زبان C در محیط CodeVisionAVR تأخیر ایجاد کنیم، می‌توانیم در ابتدای برنامه فایل <delay.h> را با عبارت:

```
# include <delay.h>
```

قرار دهیم.

در این صورت با تابع : (یک عدد) delay_ms تأخیری معادل عددی که در داخل پرانتز قرار داده شده برحسب میلی‌ثانیه ایجاد می‌شود.

به‌عنوان مثال عبارت:

```
delay_us (100)
```

مقدار 100* میکروثانیه در برنامه تأخیری ایجاد می‌کند. و یا با عبارت:

```
delay_ms (10)
```

مقدار 10* میلی ثانیه در برنامه تأخیر ایجاد خواهد کرد.

از این توابع تأخیر در مثال‌های بعدی استفاده خواهد شد.

**مثال**

برنامه‌ای به زبان C در محیط CodeVisionAVR در میکروکنترلر AVR: ATmega8 , ATmega16 ، ATmega32 ... بنویسید که هر بار تایمر 0 ، پُر شد، تقاضای وقفه کند و در روتین سرویس وقفه، بیت 0 پورت B را هر 300 میلی‌ثانیه معکوس کند. در ضمن فرکانس تایمر برابر 1/1024 فرکانس میکروکنترلر است.

**حل**

برنامه مذکور مطابق شکل (۵-۱۷- الف) می‌باشد. در این برنامه با گزینه‌هایی که در ابزار CodeWizardAVR و نرم‌افزار CodeVisionAVR انتخاب کردیم (شکل (۵-۱۷- ج)) عبارات (۱) و (۵) تا (۱۱) تولید شده‌اند که معنی هر یک از آن‌ها به شرح زیر می‌باشد.

**بخش A :**

نوع میکروکنترلر، فرکانس آن و... مشخص شده است.

**بخش B :**

◆ عبارت (۱): فایل Header File میکروکنترلر ATmega16:AVR می‌باشد که اگر ATmega8:AVR ، ATmega32 باشد، فایل مذکور به ترتیب <mega8.h> ، <mega32.h> است.

* عدد داخل پرانتز حداکثر 32768 می‌باشد.



◆ عبارت (۲): تابع delay.h را در اختیار برنامه C قرار می‌دهد، به‌طوری که ما می‌توانیم این تأخیر را برحسب میلی‌ثانیه به صورت (عدد) delay_ms استفاده کنیم که عدد داخل پرانتز مقدار تأخیر است.

به‌عنوان مثال در بخش C در عبارت (۴) تابع (300) delay_ms مقدار 300 میلی‌ثانیه تأخیر ایجاد می‌کند.

**بخش D :**

برنامه اصلی میکروکنترلر است که در آن به پورت B و تایمر 0 ، مقدار اولیه داده شده است. لذا:

◆ دستورات (۵) و (۶): پورت B را خروجی می‌کنند.

◆ دستور (۷): بیت 0 و بیت 2 ثبات کنترل TCCR0 یعنی CS00 و CS02 را برابر 1 می‌کند، لذا فرکانس پالس تایمر 0 ، برابر 1/1024 فرکانس میکروکنترلر (یعنی برابر 0.977khz) می‌شود و در ضمن تایمر شروع به کار می‌کند.

◆ دستور (۸): مقدار اولیه تایمر TCNT0 را برابر 0 می‌کند.

◆ دستور (۹): بیت 0 یعنی بیت فعال‌ساز وقفه TOIE0 در ثبات TIMSK را برابر 1 می‌کند تا وقفه سرریز تایمر 0 فعال شود.

◆ دستور (۱۰): بیت فعال‌ساز وقفه کلی I در ثبات SREG را برابر 1 می‌کند تا وقفه بتواند فعال شود.

دستور (۱۱): حلقه داخلی (1) while را مرتباً تکرار می‌کند، یعنی میکروکنترلر منتظر می‌ماند تا تایمر پُر شود و تقاضای وقفه نماید. به محض پُر شدن تایمر 0 ، بیت پرچم وقفه TOV0 برابر 1 می‌گردد و میکروکنترلر روتین سرویس وقفه را در بخش C اجرا می‌کند.

**بخش C**

روتین سرویس وقفه مربوط به سرریز تایمر 0 قرار دارد که در آن:

◆ دستور (۳): باعث می‌شود که بیت 0 پورت B معکوس شود.

◆ دستور (۴): مقدار 300 میلی‌ثانیه تأخیر ایجاد می‌کند و سپس میکروکنترلر به برنامه اصلی و به حلقه while در عبارت (۱۱) برمی‌گردد و عملیات را تکرار می‌نماید.

به این ترتیب هر بار که تایمر 0 ، پُر شد، در روتین سرویس وقفه بیت 0 پورت B معکوس می‌شود و تأخیری معادل 300 میلی‌ثانیه ایجاد می‌شود* و عملیات تکرار می‌گردد.

فایل HEX این برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

نام فایل timer02_atmega16.c

```
/*****************************************************
A Section
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1.000000 MHz
```

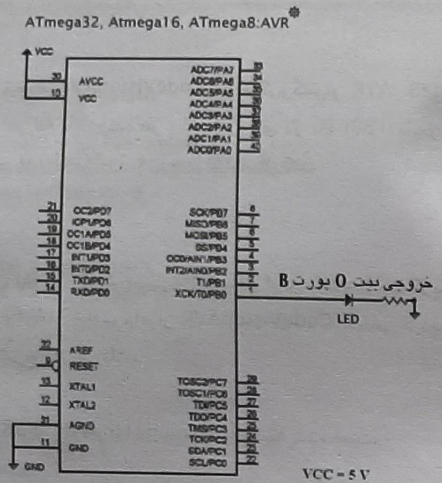
* اگر یک لامپ LED به بیت 0 پورت B متصل شود، خاموش و روشن شدن آن با حدود 300 میلی‌ثانیه با چشم دیده می‌شود.

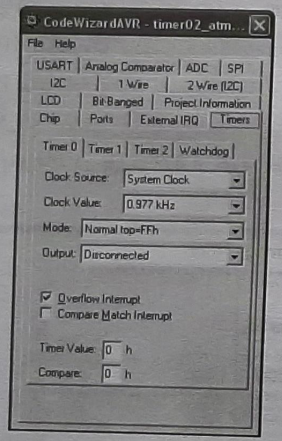
```
*****************************************************/
//B Section
#include <mega16.h>                    //(1)
#include <delay.h>                     //(2)
//*****************************************************
//C Section
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
PORTB.0=!PORTB.0;                      //(3)
delay_ms(300);                         //(4)
}
//*****************************************************
//D Section
// Declare your global variables here

void main(void)
{
// Declare your local variables here
// Port B initialization, as ouput
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out
// Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0
// State2=0 State1=0 State0=0
PORTB=0x00;                            //(5)
DDRB=0xFF;                             //(6)

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 0.977 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x05;                            //(7)
TCNT0=0x00;                            //(8)
// Timer0 interrupt enable
TIMSK=0x01;                            //(9)
// Global enable interrupts
#asm("sei")                            //(10)
while (1)                              //(11)
    {
    // Place your code here
    };
}
```

*الف: برنامه*

*ب: شکل میکروکنترلر*






*ج : تنظیمات ابزار CodeWizardAVR برای تایمر 0 و برنامه timer02_atmega16.c*

*شکل (۵-۱۷) برنامه به زبان C در محیط CodeVisionAVR با استفاده از وقفه در میکروکنترلر AVR: ATmega8 ،ATmega16 ،ATmega32 و... که بیت 0 پورت B را هر 300 میلی‌ثانیه معکوس می‌کند.*

فایل با پسوند HEX برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

* شماره پایه‌های آن مطابق شکل (۳-۱۰) فصل (۳) می‌باشد.

## مثال

برنامه‌ای به زبان C در محیط CodeVisionAVR در میکروکنترلر AVR: ATmega8 , ATmega16 ، ATmega32 بنویسید که در لبه بالارونده هر پالس ورودی در T0 (PB0) تایمر 0 ، یک شماره بیندازد و در هر لحظه خروجی تایمر TCNT0,0 را به پورت D ارسال کند.

## حل

برنامه آن مطابق شکل (۵-۱۸- الف) می‌باشد. در این برنامه با گزینه‌هایی که در ابزار CodewizardAVR (شکل (۵-۱۸- ج)) نرم‌افزار CodeVisionAVR انتخاب کردیم عبارات (۰) تا (۷) تولید شده‌اند که معنی آن‌ها به شرح زیر می‌باشد.

**بخش A :**

اطلاعات مربوط به میکروکنترلر AVR: ATmega16* نوشته شده است.

**بخش B :**

- فایل Header File با عبارت (0) مربوط به میکروکنترلر AVR: ATmega16 می‌باشد که اگر AVR:ATmega8 یا ATmega32 باشد، فایل مذکور <mega8.h> یا <mega32.h> است.
- چون ورودی T0 در پایه PB0 است، لذا توسط دستورات (۱) و (۲) پورت B ورودی شده است و مقاومت pull up در آن قرار گرفته است.
- چون در این برنامه اطلاعات تایمر باید در پورت D قرار گیرد. لذا توسط دستورات (۳) و (۴) پورت D خروجی شده است.

```
Counter00_atmega16.c
/*****************************************************
A Section
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1.000000 MHz
*****************************************************/
//B Section
#include <mega16.h>                    //(0)
// Declare your global variables here
void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
// Port B initialization, as input
// Func7=In Func6=In Func5=In Func4=In Func3=In
// Func2=In Func1=In Func0=In
// State7=P State6=P State5=P State4=P State3=P
```

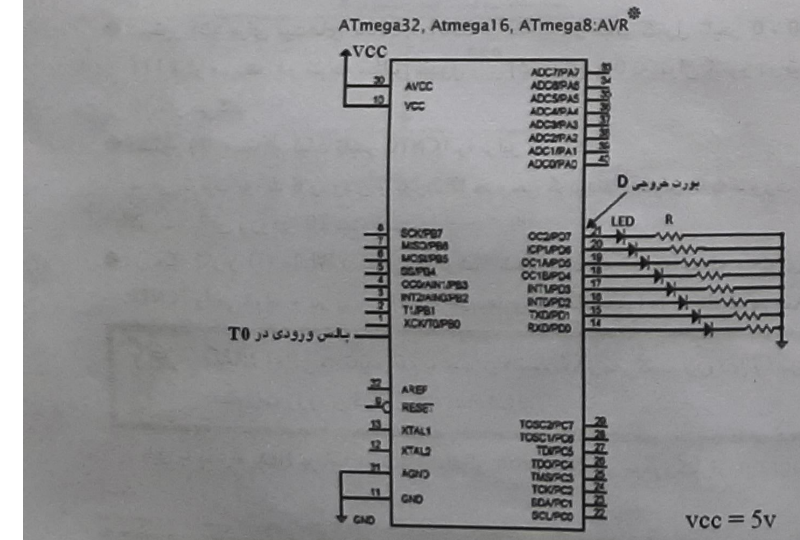
* اگر میکروکنترلر AVR: ATmega8 یا ATmega32 باشد، مشخصات آن نوشته می‌شود.



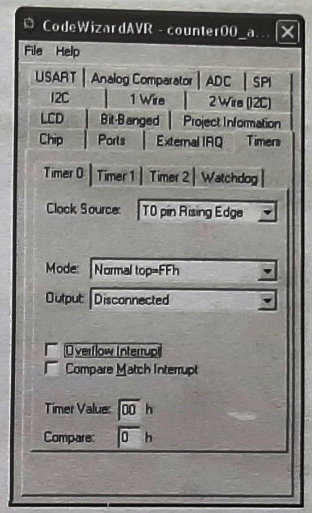
```
// State2=P State1=P State0=P
PORTB=0xFF;                    //(1)
DDRB=0x00;                     //(2)
// Port D initialization, as output
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out
// Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0
// State2=0 State1=0 State0=0
PORTD=0x00;                    //(3)
DDRD=0xFF;                     //(4)
// Timer/Counter 0 initialization
// Clock source: T0 pin Rising Edge
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x07;                    //(5)
TCNT0=0x00;                    //(6)
while (1)                      //(7)
    {
    // Place your code here
    PORTD=TCNT0;               //(8)
    };
}
```

*الف: برنامه*

*ب: شکل میکروکنترلر*




* شماره پایه‌های آن مطابق شکل (۳-۱۰) فصل (۳) می‌باشد.




*ج : تنظیمات ابزار CodeWizardAVR برای تایمر 0 به‌عنوان کنتور در برنامه counter00-atmwga16.c*

***شکل (۵-۱۸) برنامه به زبان C در محیط CodeVisionAVR کنتور 0 در میکروکنترلر AVR : ATmega8*، ATmega16، ATmega32 و... حساس به لبه بالارونده پالس در $T_0$***

◆ دستور (۵): برای بیت‌های CS00 ، CS01 و CS02 در ثبات کنترل تایمر 0 ، TCCR0** مقدار 111 قرار می‌دهد، در نتیجه مطابق جدول*** (۵-۴) تایمر 0 به‌عنوان کنتور در لبه بالارونده پالس T0 کار می‌کند.

◆ دستور (۶): محتوای ثبات تایمر TCNT0 را برابر 0 می‌کند.

به این ترتیب پورت B ورودی و پورت D خروجی گردیدند و تایمر 0 به‌صورت کنتور حساس به لبه بالارونده پالس ورودی T0 شده است.

● در حلقه تکرار While (1) دستور (۸) نوشته شده است که در هر لحظه محتوای ثبات کنتور 0 ، TCNT0 را می‌خواند و در پورت D قرار می‌دهد و این عملیات را مرتباً تکرار می‌نماید.

**نکته:** *زمانی که تایمر 0 ، به‌عنوان کنتور کار می‌کند، ورودی پالس T0 باید با دستورات ورودی گردد و Pull up شود.*

فایل با پسوند HEX برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

* اختلافات آنها در توضیح برنامه شرح داده شده است.

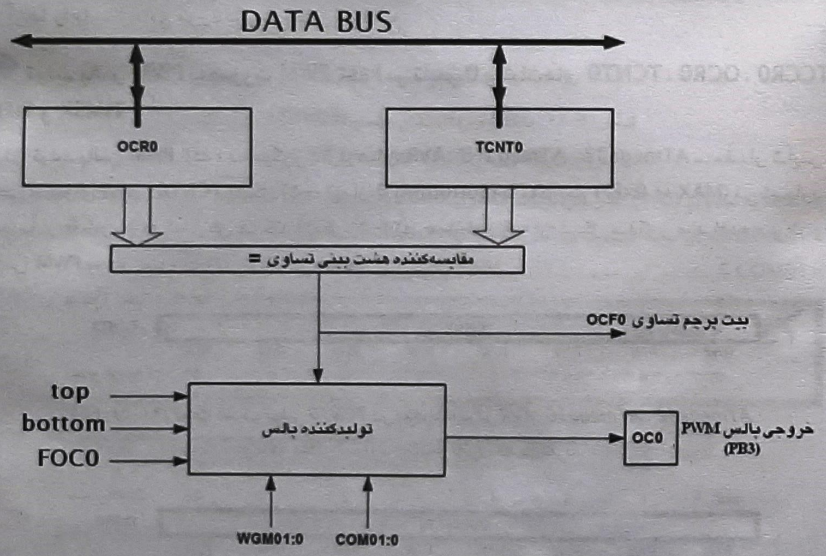
** در مورد AVR:ATmega8 شکل (۵-۶) می‌باشد.

*** در مورد AVR:ATmega8 جدول (۵-۲) می‌باشد.



## ◆ تولید پالس PWM* در تایمر 0

همانطور که قبلاً بحث شد، به‌وسیله پالس PWM می‌توان عرض پالس را تغییر داد. برای تولید پالس PWM در میکروکنترلرهای AVR: ATmega16 ، ATmega32 و... همانطور که از شکل‌های (۵-۱۵) و (۵-۱۹) مشاهده می‌شود، تایمر 0 دارای یک مقایسه‌کننده[1] هشت بیتی است که به‌طور دائم مقدار شمارشی که در ثبات تایمر 0 ، TCNT0 است را، با ثبات[2] OCR0 مقایسه می‌کند. اگر محتوای این دو ثبات با هم مساوی شوند، یک سیگنال تساوی[3] می‌فرستد که باعث می‌گردد بیت پرچم تساوی OCF0 در ثبات پرچم TIRF (شکل (۵-۲۴)) برابر 1 شود. قسمت تولیدکننده پالس[4] PWM ، با استفاده از این سیگنال و همچنین مقادیر WGM00 ، WGM01 ، COM00 و COM01 که کاربر در ثبات کنترل TCCR0 قرار می‌دهد، نوع پالس PWM را مشخص می‌کند.**




*شکل (۵-۱۹) طرز تولید پالس PWM در تایمر 0 در میکروکنترلر AVR: ATmega16 و ATmega32 و...*

در تایمر:

● مقدار Bottom برابر صفر یعنی 0x00 است.

● مقدار ماکزیمم MAX برابر 0xFF (یا 255) می‌باشد.

* قبل از مطالعه این بخش توصیه می‌شود بخش (۵-۶) را مطالعه فرمایید.

** بعداً بیشتر بحث می‌شود.

1- Comparator
2- Output Compare Register (OCR0)
3- Match
4- Waveform Generator

● حداکثر مقداری که تایمر 0 ، می‌تواند بشمارد، TOP نام دارد. مقدار TOP بستگی به طرز کار تایمر دارد. در برخی حالت‌های تایمر 0 ، مقدار TOP برابر MAX یعنی 0xFF می‌باشد. در برخی حالت‌ها مقدار TOP مساوی مقدار ثبات OCR0 است (جدول (۵-۶)).

## ◆ ۵-۱۰: حالت‌های مختلف تولید پالس PWM تایمر 0 در میکروکنترلرهای AVR: ATmega16 ، ATmega32 و...

همانطور که قبلاً متذکر شد، تایمر 0 ، پالس PWM در خروجی OC0 (پایه PB3) تولید می‌کند، که بستگی به مقداری که در ثبات کنترل TCCR0 گذاشته می‌شود، به‌صورت Fast PWM یا حالت Phase Correct PWM و... می‌باشد که به‌وسیله آن‌ها عرض پالس را می‌توان تغییر داد. در ذیل هر یک از آن‌ها را با مثال‌هایی مورد بررسی قرار می‌دهیم.

### ◆ تولید پالس PWM به‌صورت Fast PWM در تایمر 0 و ثبات‌های TCNT0 ، OCR0 ، TCCR0 ، TIFR و TIMSK

برای تولید پالس Fast PWM در میکروکنترلرهای AVR: ATmega16 ، ATmega32 ... مقدار تایمر 0 یعنی محتوای ثبات TCNT0[1] (شکل (۵-۲۰)) از 0 یا (Bottom) تا ماکزیمم (MAX = 0xFF) می‌شمارد و مجدداً از 0 شروع به شمارش می‌کند (شکل (۵-۲۲)). همانطور که از شکل مذکور مشاهده می‌شود، پالس PWM به دو صورت زیر تولید می‌شود:

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TCNT0 | TCNT0[7:0] | | | | | | | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۲۰) ثبات هشت بیتی TCNT0 در میکروکنترلر AVR: ATmega16 ، ATmega32 و... که مقدار شمارش تایمر 0 را نگه می‌دارد.*

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| OCR0 | OCR0[7:0] | | | | | | | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

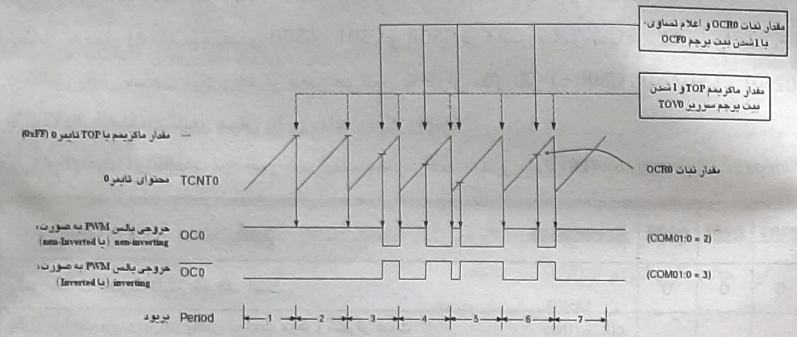
*شکل (۵-۲۱) ثبات هشت بیتی OCR0 در میکروکنترلر AVR: ATmwga16 ، ATmega32 و... که به‌طور مداوم با محتوای ثبات TCNT0 مقایسه می‌شود.*

● **در حالت Non-inverted یا Non-inverting:** موقعی که مقدار تایمر 0 یعنی TCNT0 برابر مقدار ثبات OCR0 شکل (۵-۲۲) شود، خروجی پالس PWM در پایه OC0 برابر 0 می‌شود و هنگامی که تایمر 0 ، مساوی 0 شود خروجی پالس PWM در پایه OC0 برابر 1 می‌گردد.

1- Timer/Counter Register (TCNT0)



● **در حالت Inverted یا Inverting:** هنگامی که مقدار تایمر 0 ، یعنی TCNT0 برابر مقدار ثبات OCR0 شود، خروجی OC0 برابر 1 می‌گردد و زمانی که مقدار تایمر 0 ، مساوی 0 شود خروجی پالس PWM در OC0 برابر 0 می‌گردد.




*شکل (۵-۲۲) دیاگرام زمانبندی پالس Fast PWM تایمر 0 ، در میکروکنترلر AVR: ATmega16 ، ATmega32 و...*

مطابق جدول (۵-۳) اگر در ثبات کنترل TCCR0 (شکل ۵-۲۳)، مقدار بیت‌های COM00=0 و COM01=1 قرار داده شود، پالس Non-inverting PWM تولید می‌شود و اگر COM00=1 و COM01=1 گردد، در این صورت پالس PWM به‌صورت Inverting تولید می‌شود.

**نکته:** مقدار ثبات OCR0 نباید برابر ماکزیمم (0xFF) یا 0 باشد، در غیر این صورت پالسی در خروجی PWM (OC0) نخواهد داشت یا پالس نازک[1] خواهیم داشت.

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TCCR0 | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
| | W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۲۳) ثبات کنترل TCCR0[2] تایمر 0 در میکروکنترلر AVR: ATmega16 ، ATmega32 و...*

*جدول (۵-۳) مقادیر بیت‌های COM01 و COM00 در ثبات کنترل TCCR0 در حالت Fast PWM در میکروکنترلر AVR: ATmega16 و ATmega32*

| COM01 | COM00 | شرح |
|---|---|---|
| 0 | 0 | تایمر در حالت معمولی کار می‌کند و OC2 قطع است. |
| 0 | 1 | تایمر در حالت معمولی کار می‌کند و OC0 قطع است. |
| 1 | 0 | هنگامی که تساوی پیش آید OC0 صفر می‌شود و در TOP برابر 1 می‌گردد. |
| 1 | 1 | هنگامی که تساوی پیش آید OC0 یک می‌شود و در TOP برابر 0 می‌گردد. |

1- Spik Pulse

2- Timer/Counter Control Register (TCCR0)

### ◆ در ثبات کنترل TCCR0

● مطابق جدول (۵-۶)، بیت‌های WGM00 و WGM01 نوع پالس PWM را مشخص می‌کنند که در حالت تولید پالس Fast PWM مقادیر WGM00 = 1 و WGM01 = 1 باید قرار داده شوند.

● مطابق جدول (۵-۴)، بیت‌های CS00 ، CS01 و CS02 فرکانس پالس ساعت تایمر را نسبت به فرکانس پالس ساعت میکروکنترلر معین می‌کنند. به‌عنوان مثال اگر CS00=1 ، CS01=0 و CS02=0 باشند، پالس ساعت تایمر همان پالس ساعت میکروکنترلر می‌باشد.

*جدول (۵-۴) بیت‌های انتخاب فرکانس پالس ساعت تایمر 0 در میکروکنترلر AVR: ATmega16 ، ATmega32*

| CS02 | CS01 | CS00 | Description | شرح |
|---|---|---|---|---|
| 0 | 0 | 0 | | پالس ساعت ندارد، تایمر متوقف است. |
| 0 | 0 | 1 | $clk_{I/O}$/(No prescaling) | پالس ساعت تایمر برابر پالس ساعت میکروکنترلر است. |
| 0 | 1 | 0 | $clk_{I/O}/8$ | پالس ساعت تایمر برابر 1/8 پالس ساعت میکروکنترلر است. |
| 0 | 1 | 1 | $clk_{I/O}/64$ | پالس ساعت تایمر برابر 1/64 پالس ساعت میکروکنترلر است. |
| 1 | 0 | 0 | $clk_{I/O}/256$ | پالس ساعت تایمر برابر 1/256 پالس ساعت میکروکنترلر است. |
| 1 | 0 | 1 | $clk_{I/O}/1024$ | پالس ساعت تایمر برابر 1/1024 پالس ساعت میکروکنترلر است. |
| 1 | 1 | 0 | | در لبه پایین رونده پالس در پایه T0 تایمر 0 می‌شمارد. |
| 1 | 1 | 1 | | در لبه بالا رونده پالس در پایه T0 تایمر 0 می‌شمارد. |

**نکته:** *پایه OC0 (یا PB3) با قرار دادن 1 در ثبات جهت DDRB باید به‌صورت خروجی تعریف شود.*

**مثال**

محتوای تایمر 0 را در پورت B نشان دهید:

```
in r26 , TCNT0      ; (1)
out PORTB , r26     ; (2)
```

◆ دستور (۱): محتوای تایمر 0 را در ثبات r26 قرار می‌دهد.

◆ دستور (۲): محتوای ثبات r26 را در پورت B قرار می‌دهد.



**مثال**

مقدار Compare Match را برابر 150 قرار دهید.

```
ldi r16 , 150       ; (1)
out OCR0 , r16      ; (2)
```

◆ دستورات (۱) و (۲): مقدار 150 را در ثبات OCR0 قرار می‌دهند لذا مقدار Compare Match مساوی 150 می‌شود.

به علت موج دندانه‌اره‌ای که در تایمر 0 ، ایجاد می‌شود، فرکانس پالس خروجی Fast PWM بالا و دو برابر فرکانس حالت Phase Correct است که موج تایمر به صورت مثلثی* است. لذا موج Fast PWM که فرکانس آن بالا است برای کاربردهایی نظیر رگولاتورها[1]، رکتیفایرها[2]، تبدیل دیجیتال به آنالوگ DAC[3] ... به کار برده می‌شود. علاوه بر این قطعات اضافی خارجی مانند کویل و خازن‌ها و... نیز در این حالت کوچکتر و ارزان‌تر می‌باشد.

فرکانس پالس خروجی PWM برابر است با:

$$f_{PWM}^{**} = \frac{\text{فرکانس پالس ساعت تایمر میکروکنترلر}}{N \times 256} \qquad (۱)$$

که N مدار تقسیم فرکانس در Prescaler است که می‌تواند 1 ، 8 ، 64 ، 256 یا 1024 باشد.

**نکته:** *در حالت Fast PWM تایمر 0 ، چون تایمر از 0 تا 0xFF می‌شمارد، لذا پریود پالس PWM ثابت است و تابع پالس ساعت تایمر و مقدار ماکزیمم تایمر 0xFF می‌باشد، ولی عرض پالس با مقدار Compare Match یعنی مقدار ثبات OCR0 عوض می‌شود.*

### ◆ ثبات پرچم تایمر 0، TIFR[4] در میکروکنترلرهای AVR: ATmega16 ، ATmega32

ثبات پرچم TIFR (شکل (۵-۲۴)) دارای بیت‌های پرچم مربوط به تایمرها است که در مورد تایمر 0 دو بیت پرچم زیر را دارد.***

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| TIFR | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۲۴) ثبات پرچم TIFR تایمرها در میکروکنترلر AVR: ATmega16 ، ATmega32 و...*

* بعداً بحث می‌شود.

** مطابق شکل (۵-۲۲)، چون هر بار که کنتور 256 پالس می‌شمارد پالس PWM یکبار عوض می‌شود.

*** بقیه بیت‌ها مربوط به تایمر 0 و تایمر 1 است.

1- Regulator
2- Rectifier
3- Digital to Analog Converter (DAC)
4- Timer/Counter Interrupt Flag Register (TIFR)

● **بیت ۱- بیت پرچم تساوی Compare Match به نام OCF0**[1]

هنگامی که مقدار شمارش تایمر 0 ، (TCNT0) برابر مقدار ثبات مقایسه OCR0 شود آن وقت بیت پرچم تساوی OCF0 برابر 1 می‌شود.

زمانی که این بیت پرچم برابر 1 شد و بیت فعال کردن وقفه تایمر 0 برای Compare Match به نام OCIE0 در ثبات وقفه TIMSK نیز برابر 1 باشد و همچنین بیت I در ثبات SREG نیز برابر 1 باشد، آن وقت روتین سرویس وقفه مربوط به Compare Match اجرا می‌شود.

در صورتی که روتین سرویس وقفه مربوطه اجرا شود، بیت پرچم تساوی OCF0 به‌طور خودکار صفر می‌گردد. البته می‌توان با دستورات میکروکنترلر با قرار دادن 1 در بیت پرچم OCF0 آن را Clear کرد.

● **بیت ۰- بیت پرچم سرریز TOV0**

هنگامی که مقدار شمارش تایمر 0 در ثبات TCNT0 از مقدار ماکزیمم یعنی OxFF هگزادسیمال گذشت، و دوباره صفر شد، آن وقت این بیت 1 می‌شود.

زمانی که این بیت پرچم 1 شد و بیت فعال کردن وقفه تایمر 0 برای سرریز TOIE0 در ثبات وقفه TIMSK نیز برابر 1 باشد و همچنین بیت I در ثبات SREG نیز مساوی 1 باشد، آن وقت روتین سرویس وقفه مربوطه به سرریز اجرا می‌شود.

درصورتی که روتین سرویس وقفه مربوط به سرریز اجرا شود، بیت پرچم سرریز TOV0 به‌طور خودکار صفر می‌گردد. البته می‌توان با دستورات میکروکنترلر با قرار دادن 1 در بیت پرچم سرریز TOV0 آن را Clear نمود.

## ◆ ثبات فعال کردن وقفه در تایمرها TIMSK[2] در میکروکنترلر AVR: ATmega16، ATmega32 و...

این ثبات دارای بیت‌های مربوط به فعال کردن وقفه تایمرها است که در مورد تایمر 0 ، دو بیت فعال کردن وقفه زیر را دارد:

● **بیت ۱- فعال کردن وقفه در صورت اعلام تساوی Compare Match به نام (OCIE0)**[3]

هنگامی که مقدار این بیت را 1 کنیم و همچنین مقدار بیت I در ثبات SREG را نیز 1 کنیم، آن وقت وقفه مربوط به Compare Match فعال می‌شود. و درصورتی‌که مقدار شمارش تایمر 0 یعنی مقدار TCNT0 برابر مقدار ثبات OCR0 شد (هنگامی که بیت OCF0 در ثبات پرچم TIFR برابر 1 شد)، آن وقت روتین سرویس وقفه مربوطه اجرا می‌شود.

| شماره بیت | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ثبات TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| مقدار اولیه | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*شکل (۵-۲۵) ثبات فعال کردن وقفه تایمرها TIMSK*

1- Output Compare Flag 0 (**OCF0**)
2- Timer/Counter Interrupt Mask Register (**TIMSK**)
3- Timer/Counter Output Compare Match Interrupt Enable (**OCIE0**)



● **بیت ۰- فعال کردن وقفه در صورت سرریز تایمر 0 (TOIE0)**[1]

هنگامی که مقدار این بیت را 1 کنیم و همچنین مقدار بیت I در ثبات SREG را نیز 1 کنیم، آن وقت، وقفه مربوط به سرریز تایمر 0 فعال می‌شود. در صورتی که در تایمر 0 سرریز به وجود آید (موقعی که بیت پرچم سرریز TOV0 در ثبات TIFR مساوی 1 شود)، آن وقت روتین سرویس وقفه مربوط به سرریز اجرا می‌شود.

**مثال**

وقفه Compare Match در تایمر 0 را فعال کنید.

```
ldi r20 , 1<<OCIE0          ; (1)
Out TIMSK , r20             ; (2)
        یا
ldi r20 , 0b00000010        ; (3)
out TIMSK , r20             ; (4)
sei                         ; (5)
```

◆ دستورات (۱) و (۲): باعث می‌شوند که بیت OCIE0 در ثبات TIMSK برابر 1 شود.

◆ دستورات (۳) و (۴): معادل دستورات (۱) و (۲) هستند فقط در دستورات (۱) و (۲) از نام سمبولیک OCIE0 استفاده شده است.

◆ دستور (۵): باعث می‌شود که بیت فعال‌ساز کلی وقفه‌ها I در ثبات SREG برابر 1 شود.

**مثال**

تایمر 0 را در حالت:

الف: Fast PWM ، به‌صورت Non-inverted با فرکانس پالس ساعت تایمر برابر پالس ساعت میکروکنترلر AVR قرار دهید.

ب: مقدار Compare Match را برابر 75 قرار دهید.

**حل**

الف:

```
ldi r16 , (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00)   ; (1)
out TCCR0 , r16                                              ; (2)
        یا
ldi r16 , 0b01101001                                         ; (3)
out TCCR0 , r16                                              ; (4)
```

1- Timer/Counter Overflow Interrupt Enable (**TOIE2**)

◆ دستورات (۱) و (۲): معادل دستورات (۳) و (۴) هستند که مطابق جدول‌های (۵-۳) و (۵-۴) و (۵-۶) چون بیت‌های CS00 ، COM01 ، WGM01 و WGM00 ثبات کنترل TCCR0 را برابر 1 قرار می‌دهند، لذا تایمر در حالت Fast PWM ، Non-inverted ، با پالس ساعت برابر پالس ساعت میکروکنترلر AVR کار می‌کند.

ب:

```
ldi r16 , 75        ; (5)
out OCR0 , r16      ; (6)
```

◆ دستورات (۵) و (۶): مقدار 75 را در ثبات OCR0 قرار می‌دهد، لذا مقدار Compare Match برابر 75 می‌گردد.

در ضمن باید بیت OC0 یا PB3 خروجی شود، لذا کل پورت B را با دستورات زیر خروجی می‌کنیم:

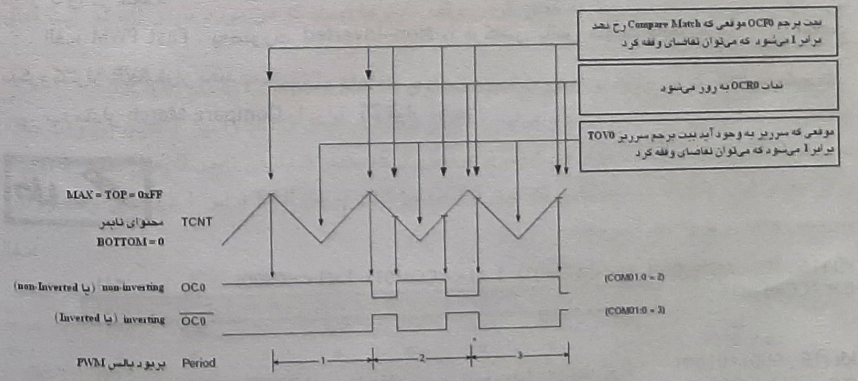
```
ser r17             ; (7)
out DDRB , r17      ; (8)
```

◆ دستور (۷): مقدار 11111111 را در ثبات r17 قرار می‌دهد.

◆ دستور (۸): مقدار 11111111 را در ثبات جهت پورت B یعنی DDRB قرار می‌دهد، لذا کل پورت B خروجی می‌شود.*

## ◆ تولید پالس PWM به صورت Phase Correct در تایمر 0

برای تولید پالس Phase Correct PWM در میکروکنترلرهای AVR: ATmega16 ، ATmega32 و... مقدار تایمر 0 یعنی محتوای ثبات (TCNT0) از 0 به‌طور صعودی تا ماکزیمم (MAX = 0xFF) می‌شمارد و سپس از ماکزیمم به صورت نزولی تا 0 می‌شمارد، به عبارت دیگر موج تولید شده توسط تایمر به صورت مثلثی مانند شکل (۵-۲۶) می‌باشد.




*شکل (۵-۲۶) دیاگرام زمانبندی پالس Phase Correct PWM*

* البته می‌توان فقط بیت 3 یعنی PB3 را خروجی کرد.



همانطور که از شکل (۵-۲۶) مشاهده می‌شود، پالس Phase Correct PWM به دو صورت تولید می‌شود:

● در حالت Non-inverted یا Non-inverting : هنگامی که تایمر 0 ، (TCNT0) در حال شمارش به‌صورت صعودی است، اگر مقدار تایمر 0 ، (TCNT0) برابر مقدار ثبات OCR0 شود، در این صورت پالس خروجی PWM در OC0 برابر 0 می‌شود و هنگامی که تایمر 0 ، به‌صورت نزولی می‌شمارد، زمانی که مقدار تایمر 0 ، (TCNT0) برابر مقدار ثبات OCR0 شد، پالس خروجی PWM برابر 1 می‌گردد (شکل (۵-۲۶)).

● در حالت Inverted یا Inverting : عملیات عکس حالت Non-inverting فوق می‌باشد (شکل (۵-۲۶)).

مطابق جدول (۵-۵) اگر در ثبات کنترل TCCR0 مقدار بیت‌های COM01 = 1 و COM00 = 0 قرار داده شود، در این صورت پالس PWM به‌صورت Non-inverting تولید می‌شود. و اگر COM01 = 1 و COM00=1 در ثبات کنترل TCCR0 قرار داده شود، آن وقت پالس PWM به‌صورت Inverting تولید می‌شود (شکل (۵-۲۶)).

***جدول (۵-۵) تعیین نوع پالس Phase Correct PWM به صورت Non-inverting یا Inverting در تایمر میکروکنترلر AVR: ATmega16 ، ATmega32 و...***

| COM01 | COM00 | توضیح |
|---|---|---|
| 0 | 0 | تایمر در حالت معمولی و OC0 قطع است. |
| 0 | 1 | تایمر در حالت معمولی و OC0 قطع است. |
| 1 | 0 | Non-inverting زمانی که تایمر 0 صعودی می‌شمارد اگر Compare Match رُخ دهد پالس PWM برابر 0 می‌شود و هنگامی که تایمر نزولی می‌شمارد اگر Compare Match رُخ دهد پالس PWM مساوی 1 می‌گردد. |
| 1 | 1 | Inverting زمانی که تایمر 1 صعودی می‌شمارد اگر Compare Match رُخ دهد پالس PWM برابر 1 می‌شود و هنگامی که تایمر نزولی می‌شمارد اگر Compare Match رُخ دهد پالس PWM مساوی 0 می‌گردد. |

## ◆ در ثبات کنترل TCCR0

● مطابق جدول (۵-۶) بیت‌های WGM00 ، WGM01 نوع پالس PWM را مشخص می‌کنند که در حالت تولید پالس Phase Correct PWM مقادیر WGM00=1 و WGM01=0 باید قرار داده شود.

***جدول (۵-۶) تعیین نوع پالس تولید شده PWM در OC0 در میکروکنترلر ATmega16 ، ATmega32 و...***

| حالت | WGM01 | WGM00 | حالت تایمر 0 | TOP | مقدار ثبات OCR0 در این موقع به‌روز می‌شود | بیت پرچم سرریز TOV0 برابر 1 می‌شود |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | معمولی | 0xFF | بلافاصله | ماکزیمم MAX |
| 1 | 0 | 1 | PWM, Phase Correct | 0xFF | در TOP | صفر BOTTOM |
| 2 | 1 | 0 | CTC | OCR0 | بلافاصله | ماکزیمم MAX |
| 3 | 1 | 1 | Fast PWM | 0xFF | در TOP | ماکزیمم MAX |

● فرکانس تایمر 0 : مطابق جدول (۵-۴) بیت‌های CS00 ، CS01 و CS02 فرکانس پالس ساعت تایمر 0 را نسبت به فرکانس پالس میکروکنترلر معین می‌کنند.

به‌عنوان مثال اگر CS00 = 1 ، CS01 = 0 و CS02 = 0 باشند، پالس ساعت تایمر 0 ، همان پالس ساعت میکروکنترلر می‌باشد.

**نکته:** *پایه OC0 (یا PB3) با قرار دادن 1 در ثبات جهت پورت (DDRB) باید به‌صورت خروجی تعریف شود.*

● در حالت Phase Correct PWM : چون تایمر به‌صورت صعودی و نزولی می‌شمارد، فرکانس پالس PWM در هر حالت نصف فرکانس حالت Fast PWM است، ولی چون پالس PWM به‌صورت قرینه می‌باشد، این حالت برای کنترل موتور مناسب‌تر است. فرکانس پالس PWM ((foc0(PWM))) در حالت Phase Corect PWM برابر است:

$$foc0_{(PWM)} = \frac{clk_{I/O}}{N \times 510} \qquad (۲)$$

که $clk_{I/O}$ فرکانس پالس ساعت میکروکنترلر، N مقدار Prescaler است که ممکن است 1 ، 8 ، 64 ، 256 یا 1024 مطابق جدول (۵-۴) انتخاب شود.

**نکته:** *در حالت Phase Correct PWM چون مقدار TOP برابر 0xFF است، پس فرکانس پالس PWM در خروجی OC0 ، تابع مقدار پالس ساعت تایمر (یعنی فرکانس پالس ساعت میکروکنترلر، تقسیم بر N) می‌باشد، ولی عرض پالس PWM ، تابع مقدار Compare Match یعنی تابع مقدار ثبات OCR0 است.*

● **دقت یا Resolution تایمر 0 در میکروکنترلر AVR: ATmega16 ، ATmega32 و...**

چون تایمر 0 ، هشت بیتی است پس دقت یا تفکیک‌پذیری Resolution تایمر هشت بیت است. به عبارت دیگر با تعداد $2^8 = 256$ قسمت کوچک می‌توانیم یک سیکل پالس PWM را بسازیم. و یا با تغییر یک بیت در ثبات OCR0 هشت بیتی، عرض پالس به اندازه 1/256 تغییر می‌کند.

● اگر مقدار ثبات OCR0 برابر 0 یا MAX (0xFF) انتخاب شود، در این صورت خروجی OC0 مقدار ثابت 1 یا 0 را خواهد داشت و پالس PWM نخواهیم داشت.

**نکته:** *زمانی که میکروکنترلر پالس PWM تولید می‌کند، ثبات OCR0 دارای بافر دوبل[1] است و تغییر مقدار ثبات OCR0 فقط در TOP انجام می‌شود تا خروجی پالس PWM متقارن گردد.*

1- Dubble Buffer



**مثال**

دستوراتی بنویسید که تایمر 0 :

الف: در حالت Phase Correct PWM و Non-inverted کار کند و پالس ساعت تایمر برابر پالس ساعت میکروکنترلر باشد.

ب: مقدار Compare Match یا ثبات OCR0 مساوی 75 باشد.

ج: پایه OC0 خروجی گردد.

**حل**

الف:

```
ldi r17 , (1<<WGM00) 1 (1<<COM01) 1 (1<<CS00)    ; (1)
out TCCR0 , r17                                  ; (2)
```

یا

```
ldi r18 , 0b01100001                             ; (3)
out TCCR0 , r18                                  ; (4)
```

◆ دستورات (۱) و (۲) معادل دستورات (۳) و (۴) هستند که باعث می‌شوند، بیت‌های WGM00 ، COM01 و CS00 در ثبات کنترل TCCR0 (شکل (۵-۲۲)) برابر 1 شوند که در نتیجه مطابق جدول‌های (۵-۴)، (۵-۵) و (۵-۶) تایمر 0 مطابق صورت مسئله کار می‌کند.

ب: دستورات (۵) و (۶):

```
ldi r19 , 75                                     ; (5)
out OCR0 , r19                                   ; (6)
```

باعث می‌شوند که مقدار 75 در ثبات OCR0 قرار گیرد یعنی Compare Match برابر 75 گردد.

ج: دستورات (۷) و (۸):

```
ser r19                                          ; (7)
out DDRB , r19                                   ; (8)
```

باعث می‌شوند که کل پورت B خروجی شود، در نتیجه بیت 3 یعنی PB3 یا پایه OC0 نیز خروجی می‌گردد.

**مثال**

به فرض اینکه میکروکنترلر در حالت Phase Correct PWM باشد، دستوراتی بنویسید که وقفه Compare Match را فعال کند.

**حل**

```
ldi r20 , (1<<OCIE0)                             ; (1)
out TIMSK , r20                                  ; (2)
```

یا

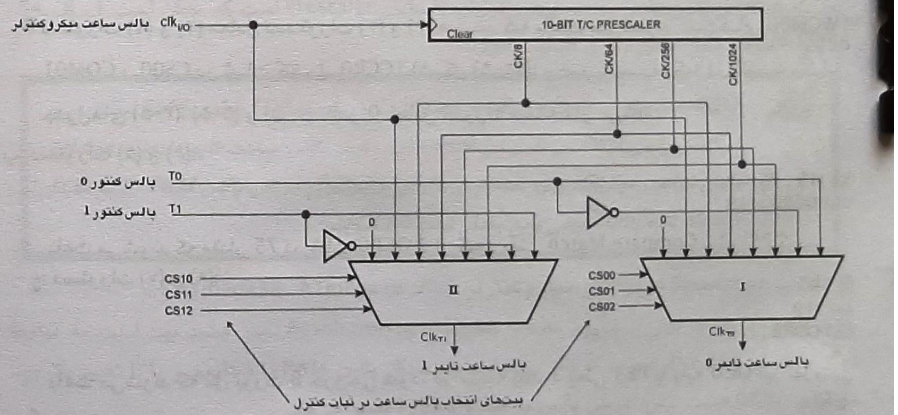
```
ldi r20 , 0b00000010      ; (3)
out TIMSK , r20           ; (4)
;
sei                       ; (5)
```

◆ دستورات (۱) و (۲): معادل دستورات (۳) و (۴) هستند و باعث می‌شوند که بیت فعال‌ساز وقفه Compare Match یعنی OCIE0 را در ثبات TIMSK (شکل (۵-۲۵)) برابر 1 کند.

◆ دستور (۵): بیت فعال‌ساز کلی وقفه ا در ثبات SREG را برابر 1 می‌کند تا وقفه‌ها بتوانند فعال شوند.

## ◆ ۵-۱۱: تولید پالس ساعت تایمر، از طریق تقسیم‌کننده فرکانس با Prescaler* در میکروکنترلرهای AVR

در میکروکنترلرهای AVR : ATmega8 ، ATmega16 ، ATmega32 و... تایمر 0 و تایمر** 1، دارای یک تقسیم‌کننده فرکانس با Prescaler مشترک هستند که پالس ساعت میکروکنترلر را، متناسب با مقادیر CS00 تا CS02 در ثبات کنترل TCCR0 ، بر عدد 8 ، 64 ، 256 یا 1024 تقسیم می‌کند (شکل (۵-۳۰)).




*شکل (۵-۲۷) تولید پالس ساعت تایمر 0 و 1 از طریق تقسیم‌کننده فرکانس، با Prescaler در میکروکنترلرهای AVR: ATmega8 ، ATmega16 ، ATmega32 و...*

در شکل مذکور مولتی پلکسر ا و II متناسب با بیت‌های CS00 ، CS01 و CS02 ثبات کنترل تایمر 0 (TCCR0) و بیت‌های CS10 ، CS11 و CS12 ثبات کنترل تایمر 1 (TCCR1) ، پالس ساعت موردنیاز تایمر 0 و تایمر 1 را با استفاده از Prescaler مطابق جدول (۵-۴) تولید می‌کنند.

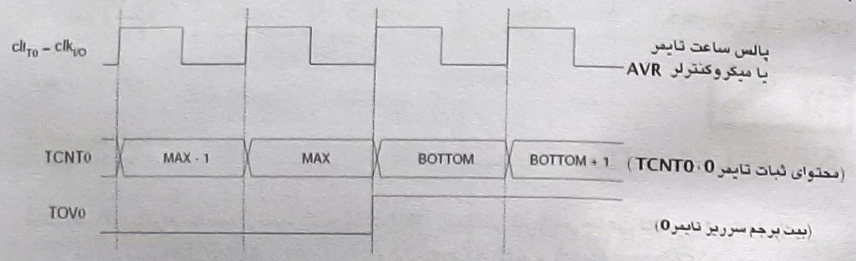
* ساختار Prescaler مشابه یک کنتور ده بیتی است که عمل تقسیم فرکانس را انجام می‌دهد.

** در فصل‌های بعدی بحث می‌شود.



## ◆ ۵-۱۲: دیاگرام زمانی تایمر 0 در میکروکنترلرهای AVR

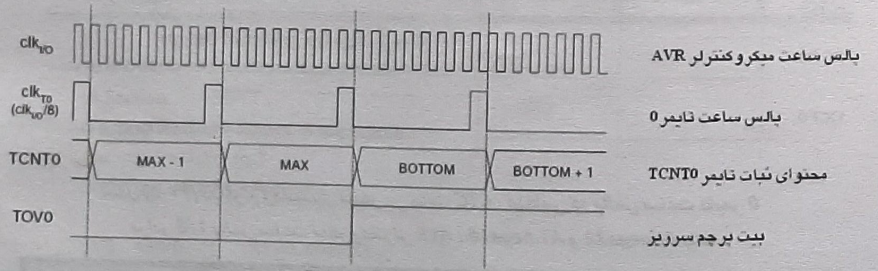
شکل (۵-۲۸) دیاگرام زمانبندی تایمر 0 موقعی که پالس ساعت تایمر برابر پالس ساعت میکروکنترلر باشد را، نشان می‌دهد.




*شکل (۵-۲۸) دیاگرام زمانبندی تایمر 0 ، هنگامی که پالس ساعت تایمر 0 برابر پالس ساعت میکروکنترلر AVR: ATmega8 ، ATmega16 ، ATmega32 و... باشد.*

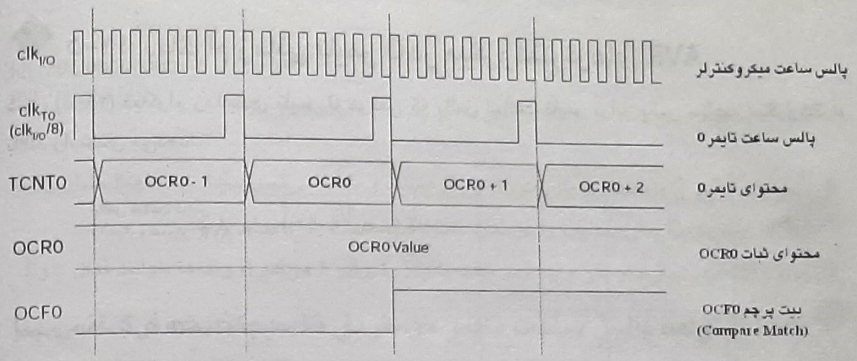
همانطور که از شکل مذکور ملاحظه می‌شود، در لبه بالارونده پالس ساعت، تایمر 0 ، یک شماره در ثبات TCNT0 می‌اندازد و زمانی که تایمر به ماکزیمم (MAX) رسید، با پالس بعدی مقدار تایمر 0 برابر BOTTOM یعنی مساوی 0 می‌شود و بیت پرچم سرریز TOV0 برابر 1 می‌گردد.

شکل (۵-۲۹) همان دیاگرام شکل (۵-۲۸) است فقط پالس ساعت تایمر $Clk_{I/O}$ توسط Prescaler برابر 1/8 پالس ساعت میکروکنترلر AVR شده است.



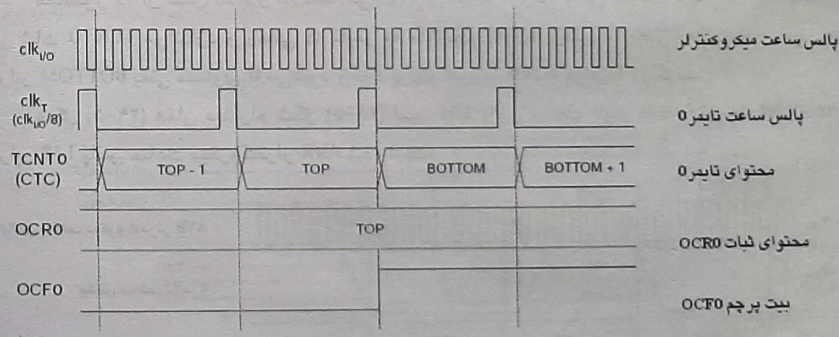

*شکل (۵-۲۹) دیاگرام زمانبندی تایمر 0 ، هنگامی که پالس ساعت تایمر 0 برابر 1/8 پالس ساعت میکروکنترلر AVR: ATmega8 ، ATmega16 ، ATmega32 و... باشد.*

شکل (۵-۳۰) دیاگرام زمانبندی تایمر 0 در حالت PWM ، زمانی که پالس ساعت تایمر، 1/8 پالس ساعت میکروکنترلر AVR باشد را نشان می‌دهد. در میکروکنترلرهای AVR : ATmega16 ، ATmega32 و... زمانی مقدار تایمر TCNT0 برابر مقدار ثبات OCR0 شد، یا Compare Match رُخ داد، بیت پرچم OCF0 در ثبات پرچم TIFR برابر 1 می‌شود.




*شکل (۵-۳۰) دیاگرام زمانبندی تایمر 0 در حالت PWM ، هنگامی که پالس ساعت تایمر برابر 1/8 پالس ساعت میکروکنترلر AVR : ATmega16 و ATmega32 باشد و Compare Match رُخ دهد.*

شکل (۵-۳۱) دیاگرام زمانبندی تایمر 0 در حالت CTC* می‌باشد و موقعی که مقدار تایمر 0 یعنی TCNT0 برابر TOPباشد، بیت پرچم OCF0 برابر 1 می‌شود.




*شکل (۵-۳۱) دیاگرام زمانبندی تایمر در حالت CTC ، هنگامی که پالس ساعت تایمر 0 برابر 1/8 پالس ساعت میکروکنترلر AVR : ATmega16 و Atmega32 باشد.*

**نکته:** *قبل از شروع کار تایمر 0 ، باید به ثبات‌های آن مقدار اولیه داد. مقدار اولیه دادن به ثبات‌ها معمولاً شامل مقدار اولیه دادن به ثبات تایمر TCNT0 ، ثبات OCR0 ، انتخاب پالس ساعت برای تایمر 0 و انتخاب نوع پالس PWM در ثبات کنترل TCCR0 ، فعال کردن یا غیرفعال کردن وقفه در ثبات TIMSK ... می‌باشد.*

* در بخش بعدی بحث می‌شود.



## مثال

برنامه‌ای بنویسید که با تایمر 0 ، پالس PWM در خروجی OC0 میکروکنترلر AVR : ATmega16 یا ATmega32 و... با روش Fast PWM تولید کند. مقدار اولیه 0x80 = OCR0 و فرکانس تایمر 0 ، برابر فرکانس میکروکنترلر است.

## حل

برنامه آن مطابق شکل (۵-۳۲) می‌باشد. در برنامه مذکور در:

**بخش A :**

نوع میکروکنترلر و فایل Header File آن به‌صورت m16def.inc برای میکروکنترلر AVR: ATmega16 مشخص شده که اگر میکروکنترلر AVR: ATmega32 باشد فایل مربوطه به‌صورت m32def.inc می‌باشد.

◆ دستور (۱): پرش به بخش B است.

**بخش B :**

مقدار اولیه‌دهی به تایمر 0 به صورت Fast PWM و پورت B می‌باشد.

◆ دستورات (۲) و (۳): در ثبات کنترلر تایمر 0 ، TCCR0 :

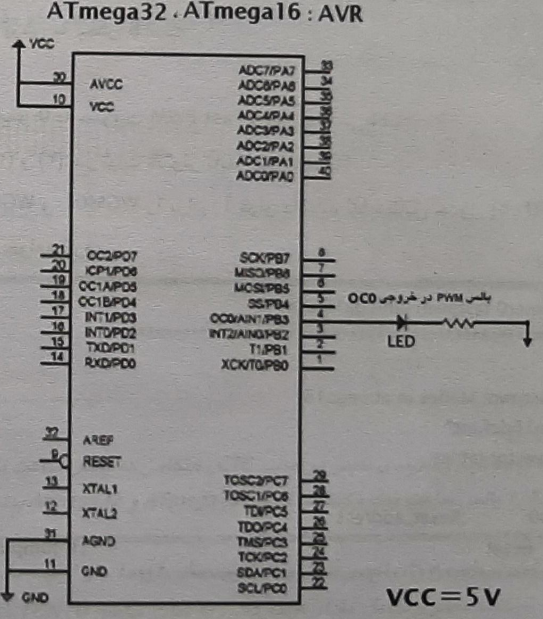
● مقدارهای WGM00 و WGM01 را برابر 1 قرار داده‌اند لذا مطابق جدول (۵-۶) تایمر 0 به‌صورت Fast PWM کار خواهد کرد.

```
;          timer0 fastpwm atmega16.asm
;*********************************************************
;A Section
;Testing fastpwm Modes in atmega16
.include "m16def.inc"
;Interrupt vector tables

.org   0x00          ;Reset_address
   rjmp     Reset                                  ;1-jump reset
;*********************************************************
;B Section
; begin of main program
reset:
;initialize fast pwm, non_inverted,clock = system clock
ldi r16,(1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS00) ;2-fast
   out TCCR0,r16                                   ;3-pwm

;set compare value or duty cycle ratio of output pulse OC0
```

```
    ldi r16,0x80                    ;4-OCR0=
    out OCR0,r16                    ;5-0x80

;set pB3 or OC0 as output
    ser r16                         ;6-DDRB=
    out DDRB,r16                    ;7-0xFF
;*****************************************************
;C Section
;endlessloop for generating output pulse on OC0
loopforever:
    rjmp loopforever                ;8-do it again
```

*الف: برنامه*

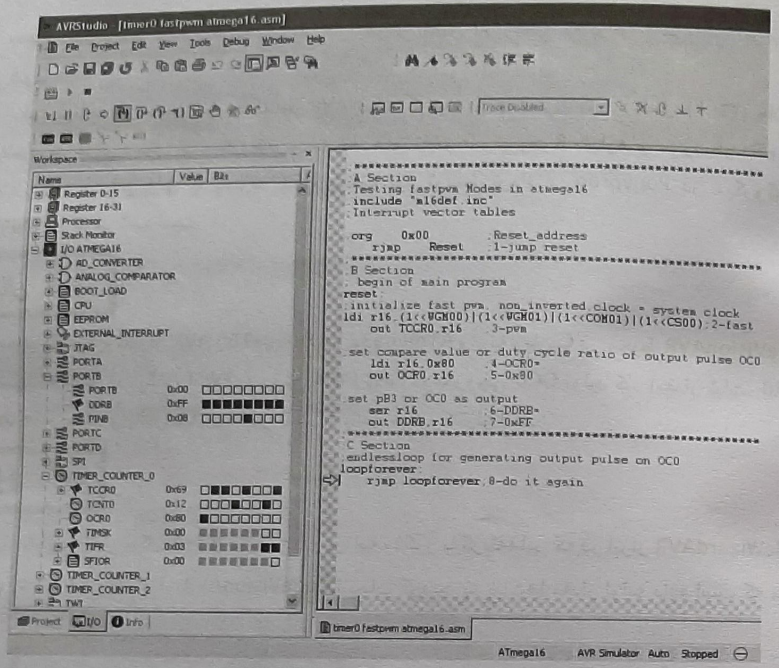
*ب: شکل میکروکنترلر*




*شکل (۵-۳۲) برنامه تولید پالس PWM به روش Fast PWM در خروجی OC0 (PB3) در میکروکنترلر AVR: ATmega16 ، ATmega32* * *و...*

* اختلافات آنها در توضیح برنامه شرح داده شده است.






*(۵-۳۳) نتیجه تست برنامه timer0 fastpwm atmega16.asm در سیمولاتور AVR Studio*

- مقدار COM01 را برابر 1 قرار داده‌اند، لذا مطابق جدول (۵-۵) پالس PWM به‌صورت Non-inverted می‌باشد.
- مقدار CS00 را برابر 1 قرار داده‌اند، لذا مطابق جدول (۵-۴) پالس ساعت تایمر 0 ، همان پالس ساعت میکروکنترلر می‌باشد.
- دستورات (۴) و (۵): در ثبات Compare Match یعنی در ثبات OCR0 مقدار 0x80 را قرار می‌دهند.
- دستورات (۶) و (۷): پورت B را خروجی می‌کنند، در نتیجه بیت PB3 یا OC0 نیز خروجی می‌شود.

بخش C :

- دستور (۸): قرار دارد که باعث می‌شود میکروکنترلر عملیات را تکرار کند، یعنی عملیات تولید پالس PWM را به‌صورت Fast PWM مطابق شکل (۵-۲۲) تولید کند.

فرکانس پالس PWM مطابق فرمول (۱) برابر است با:

$$f_{PWM} = \frac{\text{فرکانس پالس میکروکنترلر}}{N \times 256} = \frac{1000\,KHz}{1 \times 256} \approx 4\,KHz$$

زمانی که پالس PWM برابر 1 می‌باشد $\simeq \frac{OCR0}{255} = \frac{128}{255} \simeq \%50$

هر چقدر مقدار ثبات OCR0 را زیاد کنیم عرض پالس در زمانی که برابر 1 می‌باشد نیز بیشتر می‌شود.

این برنامه در سیمولاتور AVR Studio تست گردیده و همانطور که از شکل (۵-۳۳) ملاحظه می‌شود، با اجرای دستورات میکروکنترلر، به ثبات‌های تایمر 0 و پورت B مقدارهای موردنظر داده شده است. علاوه بر این فایل با پسوند HEX برنامه نیز توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

**مثال**

برنامه‌ای در میکروکنترلر AVR : ATmega16، ATmega32 و... به زبان C در محیط CodeVisionAVR بنویسید که تایمر 0، پالس PWM به روش Fast PWM در خروجی OC0 تولید کند (مقدار ثبات OCR0 = 0x80 می‌باشد و پالس ساعت تایمر 0 همان پالس میکروکنترلر است).

**حل**

برنامه آن مطابق شکل (۵-۳۴) می‌باشد. در برنامه مذکور با گزینه‌هایی که در ابزار CodeWizardAVR (شکل (۵-۳۵)) در نرم‌افزار CodeVisionAVR انتخاب کردیم، به ثبات‌ها مقدار اولیه داده است که معنی آن‌ها به شرح زیر می‌باشد. در:

**بخش A:**

نوع میکروکنترلر AVR : ATmega16 و فرکانس آن نوشته شده است.

**بخش B:**

به ثبات‌های تایمر 0 مقدار اولیه داده شده تا به صورت Fast PWM کار کند. در بخش مذکور:

- عبارت (0): فایل Header File میکروکنترلر AVR: ATmega16 است که اگر میکروکنترلر AVR: ATmega32 ... باشد فایل مذکور <mega32.h> می‌باشد.
- عبارات (1) و (2): بیت 3 پورت B یعنی PB3 یا OC0 را خروجی می‌کنند.

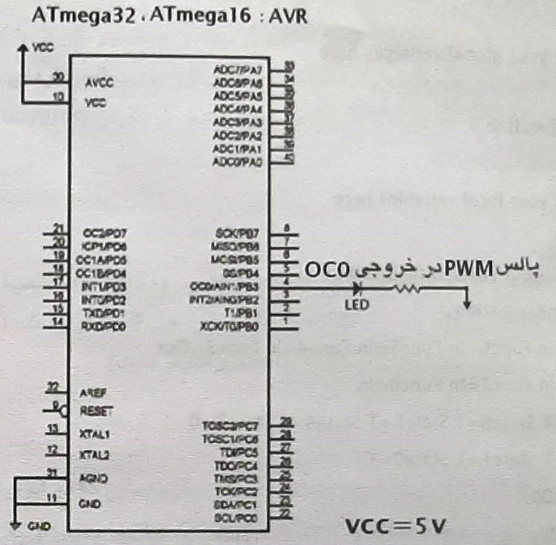
```
timer0 fastpwm.c
/*****************************************************
A Section
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1.000000 MHz
*****************************************************/
//B Section
```



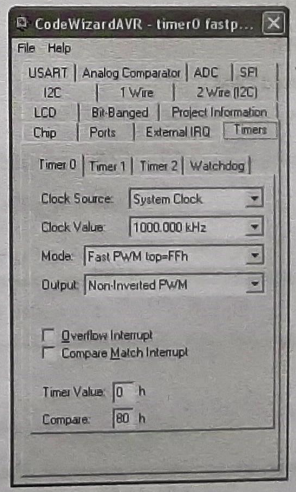
```
#include <mega16.h>                    //(0)

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out
// Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0
// State2=T State1=T State0=T
PORTB=0x00;                            //(1)
DDRB=0x08;                             //(2)
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1000.000 kHz
// Mode: Fast PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x69;                            //(3)
TCNT0=0x00;                            //(4)
OCR0=0x80;                             //(5)

while (1)                              //(6)
    {
    // Place your code here

    };
}
```

الف: برنامه

ب: شکل میکروکنترلر




شکل (۵-۳۴) برنامه تولید پالس Fast PWM به زبان C در محیط CodeVisionAVR در تایمر 0 میکروکنترلر AVR: ATmega16 و ATmega32 *




شکل (۵-۳۵) تنظیمات ابزار CodeWizardAVR برای برنامه timer0 fastpwm.c

* اختلاف آنها در توضیح برنامه بحث شده است.



◆ عبارت (۳): در ثبات کنترل تایمر 0 ، TCCR0 (شکل (۵-۲۳)) مقدار 69 هگزادسیمال یا 01101001 باینری را قرار می‌دهد. در این صورت:

● مقدارهای WGM00 و WGM01 را برابر 1 قرار داده، لذا مطابق جدول (۵-۶) تایمر 0 به‌صورت Fast PWM کار خواهد کرد.

● مقدار COM01 را برابر 1 قرار داده، لذا مطابق جدول (۵-۳) پالس PWM به‌صورت Non-inverted می‌باشد.

● مقدار CS00 را برابر 1 قرار داده، لذا مطابق جدول (۵-۴) پالس تایمر همان پالس ساعت میکروکنترلر می‌باشد.

◆ عبارت (۵): در ثبات OCR0 مقدار 0x80 را قرار داده است.

◆ عبارت (۶): باعث می‌شود که عملیات مرتباً تکرار گردد.

به این ترتیب پالس PWM به‌صورت Fast PWM مطابق شکل (۵-۲۲) در خروجی OC0 تولید می‌شود.

فرکانس پالس PWM مطابق فرمول (۱) برابر است با:

$$f_{PWM} = \frac{\text{فرکانس پالس میکروکنترلر}}{N \times 256} = \frac{1000\,KHz}{1 \times 256} \approx 4\,KHz$$

$$\text{زمانی که پالس PWM برابر 1 می‌باشد} \simeq \frac{OCR0}{255} \times 100 = \frac{0x80}{255} \times 100 = \frac{128}{255} \times 100 = \%50$$

هر چقدر مقدار ثبات Compare Match یعنی مقدار ثبات OCR0 را زیادتر کنیم، عرض پالس در زمانی که برابر 1 می‌باشد بیشتر می‌شود. *

فایل HEX این برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

**مثال**

برنامه‌ای به زبان C در محیط CodeVisionAVR بنویسید که با تایمر 0 ، پالس PWM در خروجی OC0 میکروکنترلر ATmega16 ، ATmega32 و... تولید کند (به روش Phase Correct) مقدار اولیه OCR0 = 0x80 است (فرکانس پالس تایمر همان فرکانس میکروکنترلر است).

* درصورتی‌که بخواهیم عرض پالس را عوض کنیم، می‌توانیم مُشابه شکل (۵-۳۹) با تغییر مقدار ثبات Compare Match در روتین وقفه انجام دهیم.

## حل

برنامه آن مطابق شکل (۵-۳۶) می‌باشد. در برنامه مذکور با گزینه‌هایی که در ابزار CodeWizardAVR (شکل (۵-۳۷)) در نرم‌افزار CodeVisionAVR انتخاب کردیم، عبارات (۰) تا (۶) برای پالس PWM به‌صورت Phase Correct PWM تولید شده‌اند، که معنی آن‌ها به شرح زیر می‌باشد. در:

**بخش A :**

نوع میکروکنترلر ATmega16 و فرکانس آن نوشته شده است.

**بخش B :**

به ثبات‌های تایمر 0 مقدار اولیه داده شده تا به‌صورت Phase Correct PWM کار کند. در بخش مذکور:

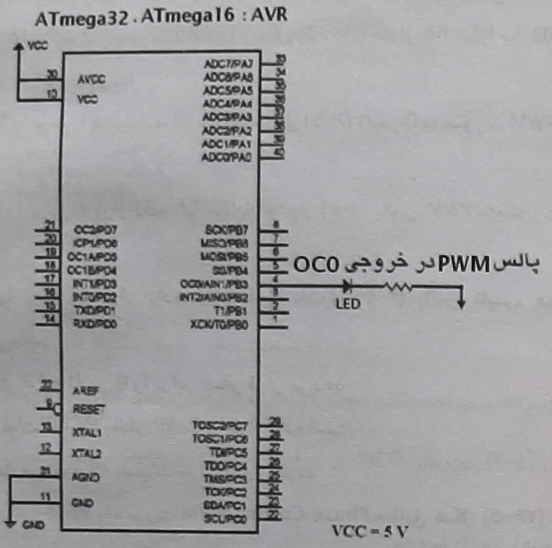
◆ عبارت (0): فایل Header File میکروکنترلر AVR : ATmega16 است که اگر میکروکنترلر AVR : ATmega32 باشد فایل مذکور به‌صورت <mega32.h> می‌باشد.

```
                phase correct atmega16.c
/*****************************************************
A Section
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1.000000 MHz
*****************************************************/
//B section
#include <mega16.h>                                   //(0)
// Declare your global variables here
void main(void)
{
// Declare your local variables here
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out
// Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0
//State2=T State1=T State0=T
PORTB=0x00;                                           //(1)
DDRB=0x08;                                            //(2)
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1000.000 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x61;                                           //(3)
TCNT0=0x00;                                           //(4)
OCR0=0x80;                                            //(5)
while (1)                                             //(6)
    {
    // Place your code here
    };
}
```

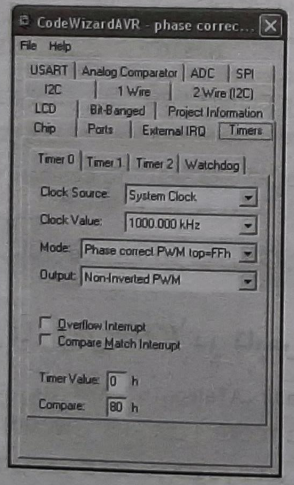
*الف: برنامه*



*ب: شکل میکروکنترلر*




*شکل (۵-۳۶) برنامه به زبان C در محیط CodeVisionAVR برای تولید پالس PWM به روش Phase Correct در میکروکنترلر AVR : ATmega16 و ATmega32* *




*شکل (۵-۳۷) تنظیمات ابزار CodeWizardAVR برای برنامه phase correct atmega16.c*

* اختلاف آن در توضیح برنامه بحث شده است.

◆عبارات (۱) و (۲): بیت 3 پورت B را خروجی می‌کند، یعنی OC0 خروجی می‌شود.

◆عبارت (۳): در ثبات کنترل تایمر 0، TCCR0 (شکل (۵-۲۳)) مقدار 61 هگزا یا 01100001 باینری را قرار می‌دهد. در این صورت:

● مقدار WGM00 را برابر 1 قرار داده، لذا مطابق جدول (۵-۶) تایمر 0 به‌صورت Phase Correct PWM کار خواهد کرد.

● مقدار COM01 را برابر 1 قرار داده، لذا مطابق جدول (۵-۵) پالس PWM به‌صورت Non-inverted می‌باشد.

● مقدار CS00 را برابر 1 قرار داده، لذا مطابق جدول (۵-۴) پالس تایمر، همان پالس ساعت میکروکنترلر می‌باشد.

◆عبارت (۴): مقدار اولیه تایمر 0 را برابر صفر قرار می‌دهد.

◆عبارت (۵): در ثبات OCR0 مقدار 0x80 را قرار داده است.

◆عبارت (۶): باعث می‌شود که عملیات مرتباً تکرار گردد.

به این ترتیب پالس PWM به‌صورت Phase Correct PWM مطابق شکل (۵-۲۶) در خروجی OC0 تولید می‌شود.

فرکانس پالس PWM مطالب فرمول (۲) برابر است با:

$$f_{OC0(PWM)} = \frac{\text{فرکانس پالس میکروکنترلر}}{N \times 510} = \frac{1000KHz}{1 \times 510} \approx 2KHz$$

همانطور که ملاحظه می‌شود فرکانس پالس Phase Correct PWM در این مثال برابر نصف فرکانس Fast PWM شکل (۵-۳۴) می‌باشد.

هر چقدر مقدار ثبات Compare Match یعنی مقدار ثبات OCR0 را زیادتر کنیم، عرض پالس در زمانی که برابر 1 می‌باشد بیشتر می‌شود.*

فایل HEX این برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

## ◆ ۵-۱۳: تولید پالس با روش CTC[1] در تایمر 0

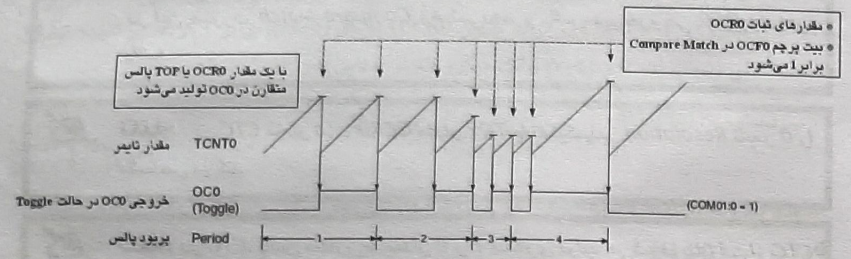
برای تولید پالس CTC در میکروکنترلرهای AVR : ATmega16، ATmega32 ... مطابق جدول (۵-۶) باید مقادیر WGM00 = 0 و WGM01 = 1 در ثبات کنترل TCCR0 قرار داده شود. در این صورت محتوای ثبات OCR0 به‌عنوان TOP در نظر گرفته می‌شود و تایمر 0 یعنی محتوای ثبات TCNT0

* در صورتی که بخواهیم عرض پالس را عوض کنیم، می‌توانیم مُشابه شکل (۵-۳۹) با تغییر مقدار ثبات Compare Match در روتین وقفه انجام دهیم.

1- Clear Timer on Compare Match (CTC)



از 0 تا زمانی که محتوای ثبات مذکور برابر مقدار ثبات OCR0 شود می‌شمارد، تا Compare Match رُخ دهد، در این صورت مقدار تایمر TCNT0 مجدداً مساوی 0 می‌شود و عملیات تکرار می‌گردد (شکل(۵-۳۸)).




*شکل (۵-۳۸) دیاگرام زمانبندی تایمر 0 در حالت CTC در میکروکنترلرهای AVR : ATmega16 ، ATmega32 و...*

مطابق جدول (۵-۷) خروجی OC0 در موقع Compare Match می‌تواند 0 ، 1 یا تغییر (Toggle) کند.

*جدول (۵-۷) خروجی OC0 در حالت CTC برای مقادیر COM00 و COM01*

| COM01 | COM00 | توضیح |
|---|---|---|
| 0 | 0 | تایمر در حالت معمولی و OC0 قطع است |
| 0 | 1 | در Compare Match مقدار OC0 عوض می‌شود (Toggle) |
| 1 | 0 | در Compare Match مقدار OC0 صفر می‌شود |
| 1 | 1 | در Compare Match مقدار OC0 یک می‌شود |

برای تولید پالس در خروجی OC0 می‌توان مقادیر COM01=0 و COM00=1 را در ثبات کنترل TCCR0 قرار داد تا مطابق جدول (۵-۷) هر بار که تساوی یا Compare Match رُخ دهد، خروجی عوض می‌شود. در این صورت فرکانس پالس خروجی $f_{OC0}$ در پایه OC0 برابر است با:

$$f_{OC0} = \frac{f_{I/O}}{2N(1+OCR0)} \qquad (۳)$$

که $f_{I/O}$ فرکانس پالس ساعت میکروکنترلر، OCR0 مقدار ثبات OCR0 و N مقدار Prescaler است که می‌تواند مطابق جدول (۵-۴) مساوی 1 ، 8 ، 64 ، 256 یا 1024 باشد.

هر بار که مقدار تایمر 0 برابر TOP یا Compare Match می‌شود، بیت پرچم OCF0 در ثبات پرچم TIFR (شکل (۵-۲۴)) مساوی 1 می‌گردد، که اگر وقفه فعال باشد، روتین سرویس وقفه اجرا می‌شود. در روتین سرویس وقفه می‌توان مقدار جدیدی برای ثبات OCR0 قرار داد که در نتیجه فرکانس پالس خروجی OC0 را مطابق فرمول فوق عوض نمود.

**نکته:** چون در حالت CTC بافر دوبل برای ثبات OCR0 وجود ندارد، لذا برای تغییر مقدار OCR0، باید موقعی که Compare Match رُخ می‌دهد، در روتین سرویس وقفه این تغییر انجام شود. چون اگر مقدار جدید OCR0 کمتر از مقدار تایمر TCNT0 باشد، در این صورت Compare Match رُخ نمی‌دهد و پالس در خروجی OC0 نخواهیم داشت.

**نکته:** در CTC مقدار ثبات OCR0 مقدار TOP یا تفکیک‌پذیر Resolution تایمر 0 را مشخص می‌کند.

**نکته:** در CTC، پالس متقارن با مقدار 0 و 1 مساوی تولید می‌شود، بنابراین از CTC می‌توان به‌عنوان تولیدکننده پالس متقارن استفاده نمود.

**نکته:** از حالت CTC می‌توان به‌عنوان تایمر استفاده نمود. به‌عنوان مثال اگر به تعداد 250 پالس، زمان لازم داشته باشیم کافیست مقدار OCR0 را برابر 250 قرار دهیم. دراین‌صورت تایمر 0 از 0 تا OCR0 یعنی از 0 تا 250 که شمرد، Compare Match رُخ می‌دهد و می‌تواند وقفه‌ای رخ دهد. به این ترتیب در زمان ثابت یعنی هر 250 پالس روتین سرویس وقفه اجرا می‌شود که می‌توان در روتین سرویس وقفه مذکور هر عملیاتی مانند معکوس کردن یک بیت پورت B و یا... را انجام داد.

**نکته:** پایه OC0 باید به‌وسیله ثبات جهت DDRB به‌عنوان خروجی تعریف شود.

**مثال**

دستوراتی در میکروکنترلر AVR: ATmega16 ، ATmega32 بنویسید که:

الف: میکروکنترلر را در حالت CTC با خروجی OC0 به صورت Toggle و پالس تایمر 0 برابر پالس میکروکنترلر نماید.

ب: مقدار ثبات OCR0 یعنی Compare Match را برابر 150 نماید.

ج: پایه OC0 را به‌عنوان خروجی تعریف نماید.

**حل**

الف:

```
ldi r16 , (1<<COM00) | (1<<WGM01) | (1<<CS00)      ; (1)
out TCCR0 , r16                                    ; (2)
```



یا

```
ldi r16 , 0b00011001                               ; (3)
out TCCR0 , r16                                    ; (4)
```

◆دستورات (۱) و (۲) معادل دستورات (۳) و (۴) هستند و باعث می‌شوند که مقدار WGM01 = 1 ، COM00 = 1 و CS00 = 1 در ثبات کنترل TCCR0 (شکل (۵-۲۳)) برابر 1 گردد. در نتیجه مطابق جدول‌های (۵-۴)، (۵-۶) و (۵-۷) حالت میکروکنترلر مطابق بند "الف" مسئله می‌شود.

ب: دستورات (۵) و (۶):

```
ldi r17 , 150          ; (5)
out OCR0 , r17         ; (6)
```

عدد 150 یا Compare Match را در ثبات OCR0 قرار می‌دهند.

◆دستورات (۷) و (۸):

```
ser r18                ; (7)
out DDRB , r18         ; (8)
```

پورت B را خروجی می‌کنند، در نتیجه بیت 3 پورت B یعنی پایه OC0 (PB3) را نیز خروجی می‌کنند.

**مثال**

برنامه‌ای به زبان C در محیط CodeVisionAVR برای میکروکنترلر AVR: ATmega16 و ATmega32 بنویسید که تایمر 0 در حالت CTC ، پالس مربعی در خروجی OC0 تولید کند. مقدار اولیه ثبات OCR0 برابر 80 هگزادسیمال می‌باشد. با استفاده از وقفه Compare Match ، مقدار ثبات OCR0 را برابر مقدار پورت D قرار دهید تا فرکانس پالس خروجی متناسب با آن تغییر نماید.

**حل**

برنامه آن مطابق شکل (۵-۳۹- ب) می‌باشد. با گزینه‌هایی که در ابزار CodeWizardAVR (شکل (۵-۳۹- الف)) و نرم‌افزار CodeVisionAVR انتخاب کردیم این نرم‌افزار مقادیر زیر را در بخش‌های مختلف برنامه قرار داده است که به شرح زیر می‌باشند:

**بخش A :**

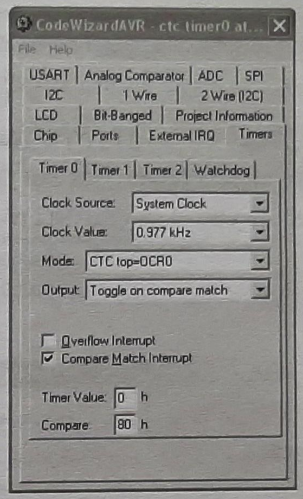
نوع میکروکنترلر و اطلاعات عمومی راجع به برنامه نوشته شده است.

**بخش C* :**

به پورت‌های B و D و تایمر 0 مقدار اولیه داده شده است. به این ترتیب:

◆دستور (۱): فایل Header File میکروکنترلر ATmega16 است که اگر میکروکنترلر AVR : ATmega32 باشد، فایل مذکور به‌صورت <mega32.h> می‌باشد.

* بخش B بعداً بحث می‌شود.




*الف : تنظیمات ابزار CodeWizardAVR برای تولید پالس CTC*
*توسط تایمر در برنامه ctc timer0 atmega16.c*

نام فایل ctc timer0 atmega16.c

```
/*****************************************************
A Section
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 1.000000 MHz
*****************************************************/
//B Section
#include <mega16.h>                    //(1)
unsigned char temp;                    //(2)
// Timer 0 output compare interrupt service routine
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
// Place your code here
OCR0=temp;                             //(3)
}
```



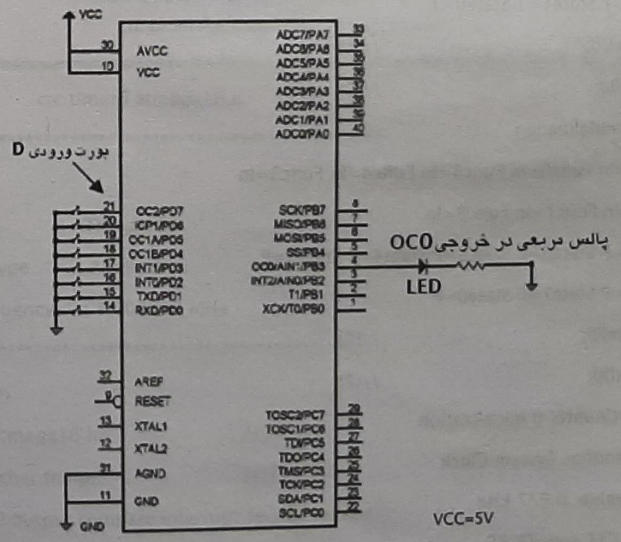
```
//*****************************************************
//C Section
// Declare your global variables here
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out
// Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0
// State2=T State1=T State0=T
PORTB=0x00;                            //(4)
DDRB=0x08;                             //(5)
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In
//Func2=In Func1=In Func0=In
// State7=P State6=P State5=P State4=P State3=P
// State2=P State1=P State0=P
PORTD=0xFF;                            //(6)
DDRD=0x00;                             //(7)
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 0.977 kHz
// Mode: CTC top=OCR0
// OC0 output: Toggle on compare match
TCCR0=0x1D;                            //(8)
TCNT0=0x00;                            //(9)
OCR0=0x80;                             //(10)
// Timer(s)/Counter(s) Interrupt(s) initialization
```

```
TIMSK=0x02;                    //(11)
// Global enable interrupts
#asm("sei")                    //(12)
while (1)
    {
    // Place your code here
    temp=PIND;                 //(13)
    };
}
```

ب: برنامه

ج: شکل میکروکنترلر

ATmega32 ، ATmega16 : AVR




شکل (۵-۳۹) برنامه تولید پالس مربعی توسط تایمر 0 با روش CTC در میکروکنترلر AVR: ATmega16 ، ATmega32* به زبان C در محیط CodeVisionAVR

◆عبارت (۲): متغیر temp را یک کاراکتر یک بایتی تعریف می‌کند.

* اختلاف آنها در توضیح برنامه بحث شده است.



◆دستورات* (۴) و (۵): بیت 3 پورت B (OC0 یا PB3) را خروجی می‌کنند.

◆دستورات (۶) و (۷): پورت D را ورودی می‌کنند و مقاومت Pull up برای آن قرار می‌دهند.

◆دستور (۸): در ثبات کنترل تایمر 0 ، TCCR0 مقدار 1D هگزادسیمال یا 00011101 باینری را قرار می‌دهد که باعث می‌شود، مطابق شکل (۵-۲۳):

■ پالس ساعت تایمر برابر 1/1024 پالس ساعت میکروکنترلر باشد (مطابق جدول (۵-۴)).

■ بیت WGM01 برابر 1 و WGM00 مساوی 0 است، پس مطابق جدول (۵-۶) تایمر به صورت CTC کار می‌کند.

■ بیت COM00 مساوی 1 است پس مطابق جدول (۵-۷) پالس مربعی در خروجی OC0 تولید می‌شود.

◆دستور (۹): مقدار اولیه تایمر 0 یعنی TNCT0 را برابر 0 قرار داده است.

◆دستور (۱۰): مقدار اولیه ثبات OCR0 را برابر 80 هگزادسیمال قرار می‌دهد.

◆دستور (۱۱): بیت فعال‌ساز وقفه Compare Match یعنی OCIE0 در ثبات TIMSK (شکل (۵-۲۵)) را برابر 1 می‌کند تا وقفه مربوط فعال شود.

◆دستور (۱۲): بیت فعال‌ساز کلی وقفه ا در ثبات SREG را برابر 1 می‌کند تا وقفه فعال گردد.

◆دستور (۱۳): از پورت D می‌خواند و مقدار آن را در متغیر temp قرار می‌دهد.

به این ترتیب به پورت‌های B و D و تایمر 0 مقدار اولیه داده شده و در هر لحظه مقدار پورت D خوانده و در متغیر temp قرار می‌گیرد. همچنین تایمر شروع به شمردن می‌کند تا به مقدار ثبات Compare Match یعنی به مقدار ثبات OCR0 برسد که در آن موقع پالس مربعی در خروجی OC0 عوض می‌شود (شکل (۵-۳۸)). در لحظه‌ای که مقدار تایمر 0 یعنی TCNT0 مساوی مقدار ثبات OCR0 شد وقفه مربوطه فعال می‌شود و میکروکنترلر به روتین سرویس وقفه در بخش B می‌رود.

**بخش B :**

توسط دستور (۳): مقدار ثبات OCR0 برابر متغیر temp یا مساوی پورت D می‌شود، لذا مطابق فرمول (۳) فرکانس پالس مربعی تغییر می‌کند.

بنابراین پالس مربعی در پایه خروجی OC0 (یا PB3) با فرکانس دلخواه توسط پورت D تولید می‌شود.

هر چقدر مقدار پورت D یعنی PIND بیشتر شود مقدار ثبات OCR0 نیز زیادتر می‌گردد، لذا عرض پالس مربعی بیشتر می‌شود و فرکانس پالس کمتر می‌گردد. فایل HEX این برنامه توسط نرم‌افزار Ponyprog در میکروکنترلر Program و تست شده است.

## ◆ ۵-۱۵: خلاصه

در این فصل ساختار اصولی تایمر و وقفه در میکروکنترلرها، آدرس یا وکتور روتین سرویس وقفه و اولویت وقفه دستگاه‌های جانبی میکروکنترلرهای AVR مورد بحث قرار گرفته است. علاوه بر این طرز کار تایمر 0 در میکروکنترلرهای AVR: ATmega8 ، ATmega16 ، ATmega32 و... ثبات‌های آن‌ها،

* دستور (۳) در بخش B بحث می‌شود.

انتخاب پالس ساعت و طریقه راه‌اندازی تایمر 0 و... مورد بررسی قرار گرفته است. همچنین ساختار برنامه اسمبلی، طرز تولید پالس PWM به‌صورت Fast PWM ، Phase Correct PWM ، CTC با تایمر 0 در میکروکنترلرهای AVR: ATmega16 ، ATmega32 و... مورد بحث قرار گرفته است و مثال‌های متنوعی با دستورات اسمبلی میکروکنترلرهای AVR و زبان C آورده شده است.

## ۵-۱۶ : تمرین

۵-۱- دستگاه‌هایی که تقاضای وقفه از میکروکنترلرهای AVR می‌توانند بکنند، کدامند؟

۵-۲- آدرس یا وکتور وقفه چیست و آدرس وقفه سرریز تایمر 0 در میکروکنترلر AVR: ATmega16 چقدر است؟

۵-۳- اولویت وقفه دستگاه‌های جانبی نسبت به هم چگونه است؟

۵-۴- اگر تایمر 0 ، پُر شود، چه پیش می‌آید؟

۵-۵- دستوراتی بنویسید که پالس ساعت تایمر 0 برابر 1/8 پالس ساعت میکروکنترلرAVR: ATmega8 شود و وقفه نیز فعال گردد.

۵-۶- برنامه‌ای در میکروکنترلر AVR : ATmega8 ، ATmega16 یا ATmega32 بنویسید که هر بار تایمر 0 ، پُر شد، تقاضای وقفه کند و روتین سرویس وقفه، پورت B را معکوس کند.

۵-۷- برنامه‌ای در میکروکنترلرهای AVR : ATmega8 ، ATmega16 یا ATmega32 به زبان C در محیط CodeVisionAVR بنویسید که هر بار تایمر 0 ، پُر شد، تقاضای وقفه کند و روتین سرویس وقفه، پورت B را هر 500 میلی‌ثانیه معکوس کند، در ضمن فرکانس تایمر 0 برابر 1/1024 فرکانس میکروکنترلر باشد.

۵-۸- فرکانس پالس PWM در روش Fast PWM چگونه است و برای تغییر فرکانس، چه پارامترهایی را باید تغییر داد؟

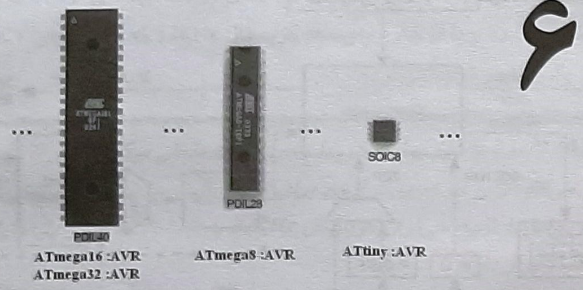
۵-۹- عرض پالس PWM در روش Fast PWM یا Phase Correct PWM چگونه تغییر می‌یابد؟

۵-۱۰- در شکل (۵-۳۴) اگر فرکانس پالس تایمر 0 را برابر 1/8 پالس ساعت میکروکنترلر و مقدار OCR0 = 64 باشد، فرکانس پالس PWM و عرض پالس چقدر است؟

۵-۱۱- برنامه‌ای به زبان C در محیط CodeVisionAVR بنویسید که با تایمر 0 ، پالس PWM در خروجی OC0 میکروکنترلر ATmega16 یا ATmega32 به روش Phase Correct و Non-inverted تولید کند. مقدار اولیه ثبات OCR0 = 0x80 است و با تغییر مقدار پورت D عرض پالس PWM عوض شود.

*** *** ***

# ۶




# تایمر 2

**هدف: آشنایی با امکانات تایمر 2 هشت بیتی**

**و حالت‌های تولید پالس PWM در میکروکنترلرهای AVR**

*چشم‌انداز این فصل:*

- مقدمه
- حالت‌های مختلف کار تایمر
- تولید پالس PWM به‌صورت Fast PWM و Phase Correct PWM و CTC...
- تولید پالس ساعت مبنای زمان ۱ ثانیه برای ساعت RTC
- مثال‌ها و برنامه‌های متنوع با دستورات اسمبلی میکروکنترلرها و زبان C بامیکروکنترلرهای AVR: ATmega32, ATmega16, ATmega8 ...
- خلاصه