به نام خدا

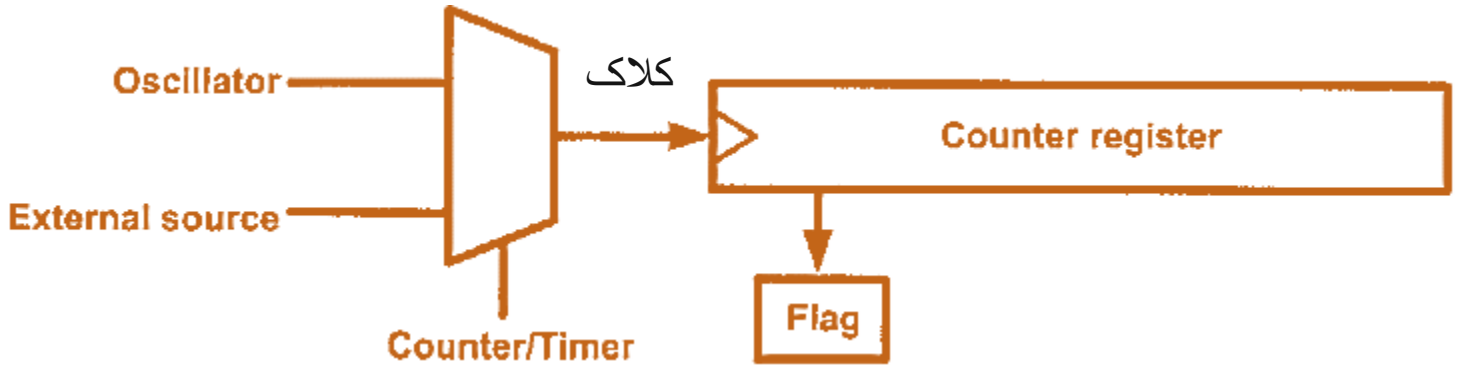
آشنایی با تایمرها

Timer in ATmega32

Dr. Aref Karimiafshar
A.karimiafshar@iut.ac.ir

AVR Timer

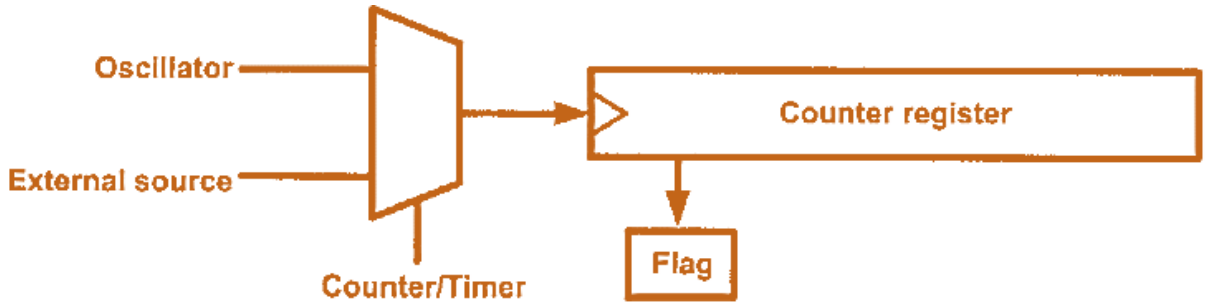
- Counter registers in µC
  - To count an event
  - Generate time delay



تایمر یک سیستم مبتنی بر شمارنده ها هستش
که ساختارش توی اسلاید روبه رو است: یک counter register داریم که توسط یک کلاک فعال میشه و با فعال شدن این پایه این شمارنده ما یک واحد شمارش میکنه
خود این کلاک هم می تونه یک عامل درونی مثل oscillator باشه یا یک عامل خارجی یا external source
و زمانی که این کانتر ما به انتهای خودش برسه یک پرچمی رو به نام flag فعال میکنه که به ما نشون میده این به حد ماکسیمم خودش رسیده
وابسته به این کلاک ما به چه منبعی وصل است می تونه استفاده متفاوتی از این ساختار داشته باشیم
به صورت کلی دو هدف از داشتن چنین سیستم هایی داریم:
1- تعداد یک سری رویدادهایی رو در سیستم یا خارج از سیستم خودمون شمارش بکنیم
2- توی سیستممون می خوایم یکسری تاخیرات ایجاد بکنیم که باز از طریق همین بحث تایمر و کانترهای توی شکل قابل پیاده سازی هستش

AVR Timer

When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred. When we want to generate time delays, we connect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter register represents how many ticks have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can calculate the tick period, and from the content of the counter register we will know how much time has elapsed.



زمانی که میخوایم یک رویدادی رو می خوایم شمارش بکنیم میایم اون پایه کلاک اون شمارنده رو به اون سیگنال مرتبط به اون رویداد وصل میکنیم و احتمالا این رویداد هم خارجی هستش
با هر باری که رویداد مورد نظر ما اتفاق می افته این شمارنده یک واحد زیاد میشه پس اون عددی رو که شمارنده ما داره نشون میده بیان میکنه که اون اتفاق چند بار رخ داده و از این طریق میایم اون رویداد خارجی رو شمارش میکنیم
ولی اگر بیایم پایه این شمارنده رو به oscillator وصل بکنیم اون موقع می تونیم یکسری تاخیراتی به صورت خیلی دقیق داشته باشیم و این کار به این صورت انجام میشه که:
وقتی که میخوایم یک تاخیر با اندازه مشخص ایجاد بکنیم میایم و این پایه ساعت رو به oscillator وصل میکنیم و وقتی که oscillator داره تیک می زنه این شمارنده ما افزایش پیدا میکنه با توجه به اینکه ما دقیقا دوره oscillator رو داریم می تونیم به صورت دقیق بیایم و اون تاخیری که مدنظرمون بوده یا ایجاد شده رو محاسبه بکنیم و در برنامه هامون استفاده بکنیم

Time Delay

So, one way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number. For example, consider a microcontroller with an oscillator with frequency of 1 MHz; in the microcontroller, the content of the counter register increments once per microsecond. So, if we want a time delay of 100 microseconds, we should clear the counter and wait until it becomes equal to 100.

In the microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software. The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set. For example, in a microcontroller with a frequency of 1 MHz, with an 8-bit counter register, if we want a time delay of 3 microseconds, we can load the counter register with $FD and wait until the flag is set after 3 ticks. After the first tick, the content of the register increments to $FE; after the second tick, it becomes $FF; and after the third tick, it overflows (the content of the register becomes $00) and the flag is set.

اما به صورت کلی برای اینکه بیایم و یک تاخیری رو ایجاد بکنیم دو روش داریم:
1- در ابتدای کار تایمر رو کلیر بکنیم و این از صفر شروع بکنه به شمارش تا به یک حد مشخصی برسه مثلا اون میکروکنترلر ما با یک فرکانس یک مگاهرتز داره کار میکنه و این به معنای این هستش که ما یک میکروثانیه تاخیر داریم و ما اگر بیایم به اندازه 100 پالس ساعت شمارنده ما شمارش انجام بده (خودمون اینو تنظیم کردیم که تا 100 برسه این تاخیر رو بده) اون موقع می تونیم به اندازه 100 میکروثانیه تاخیر ایجاد بکنیم
پس روش اول این شد که ما بیایم اون تایمر رو کلیر بکنیم و از صفر شروع بکنه این شمارش کردن و چک بکنیم به یک مقدار خاصی که رسید این متوقف بشه و اون تاخیری که ما میخوایم رو بهمون بده

2- یک حالت برعکس بالایی رو داشته باشیم ینی شمارنده رو روی یک مقدار خاصی تنظیم بکنیم و شمارنده رو فعال بکنیم و شروع بکنه به شمارش کردن و زمانی که به اون حد ماکس خودش رسید تاخیر مورد نظر ما رو بده مثلا میکروکنترلر ما یک مگاهرتز فرکانس داره و میخوایم 3 میکرو ثانیه تاخیر ایجاد بکنیم و اتفاقی که می افته اینه که ما میایم مقدار اون شمارنده رو به نحوی تنظیم می کنیم که فقط سه تا تیک ساعت برای ما بشماره مثلا اون رو روی FD تنظیم میکنیم و توی پالس اول که میاد یک واحد افزایش پیدا میکنه و میشه FE و توی پالس بعدی این میشه FF و در پالس سوم این که به حد خودش رسیده overflow میکنه و به صفر برمیگرده پس اینجا یک فلگی که معرفی کردیم فعال میشه و می فهمیم این به حد ماکس خودش رسیده و از این طریق می تونیم اون تاخیری که مدنظرمون هستش رو به دست بیاریم و توی برنامه هامون استفاده بکنیم

AVR Timers

- AVR has one to six timers
  - Depending on family member
  - Timers 0, 1, 2, 3, 4, and 5
  - Can be used as:
    - Timers: to generate time delay
    - Counters: count events happening outside the µC
  - Some T/C are 8-bit and some are 16-bit
  - For example: ATmega32
    - Three timers
      - Timer0: 8-bit
      - Timer1: 16-bit
      - Timer2: 8-bit

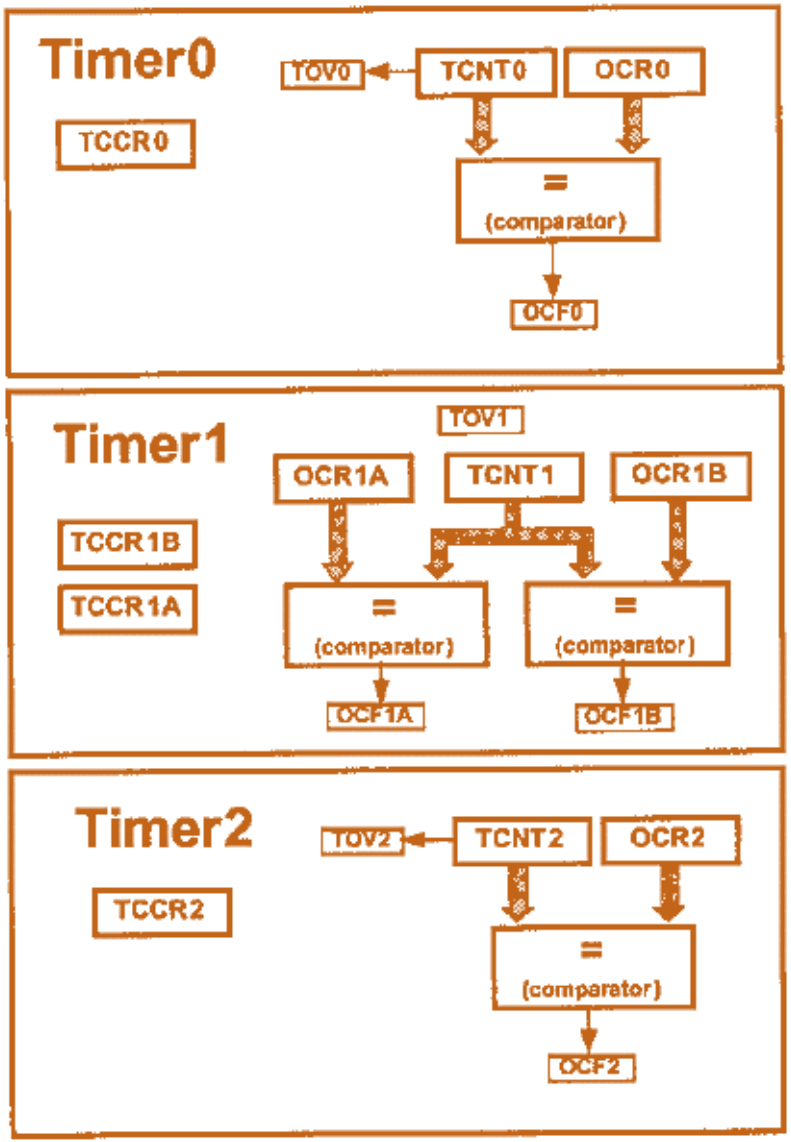
اعضای مختلف خانواده avr امکانات متنوع و متفاوتی در تایمرها دارند
به صورت کلی avr ها میتونن از 1 تا 6 تا تایمر داشته باشند که این بسته به نوع عضو اون خانواده و سری که توش هستش بستگی داره
شماره تایمرها از صفر شروع میشه مثلا اگر میکروکنترلی 6 تایمر داشته باشه ما اینو به صورت تایمرهای 0 و 1 و 2 و 3 و 4 و 5 نشون میدیم
تایمرها می تونن به منظور تولید یک تاخیر و یا شمارش یک رویدادی در خارج از اون میکروکنترلر استفاده بشن که خب ما این ها رو معمولا به عنوان تایمر یا کانتر می شماریم:
تایمر برای زمانی که می خوایم تاخیر ایجاد بکنیم و کانتر برای زمانی که میخوایم شمارش بکنیم
به هر حال اگر اون سیستم شمارشی ما چه به عنوان تایمر باشه و چه به عنوان کانتر باشه این ها می تونن در ساختارهای 8 بیتی یا 16 بیتی در این میکروکنترلرها در دسترس باشند که بسته به سری که داریم حرف می زنیم این ها متفاوت است
برای مثال: اگر ما atmega32 در نظر بگیریم این میکروکنترلر سه تایمر خواهد داشت:
تایمر صفر و تایمر 1 و تایمر 2
تایمر 0 و تایمر 2 ما 8 بیتی هستند و تایمر 1 ما 16 بیتی

Programming the Timers

- Every timer needs a clock pulse to tick
  - The clock source can be internal or external
- Internal clock source
  - The frequency of the crystal oscillator is fed into the timer
  - Called timer
- External clock source
  - Feed pulses through one of the AVR's pin
  - Called counter

برای اینکه از این تایمرها استفاده بکنیم نیاز داریم با ساختار این ها اشنا بشیم:
هر تایمری به یک منبع کلاک نیاز داره که اون شمارش خودش رو انجام بده
توی تمام این سری های Avr این ها می تونن کلاکشون به یک منبع داخلی یا خارجی متصل باشه
اگر اون منبع کلاک رو داخلی سیستم در نظر بگیریم: فرکانس oscillator ما میاد به عنوان feder اون شمارنده استفاده میشه و اینجا نقش همون تایمر رو داره
و اگر منبع کلاک ما خارجی باشه این می تونه برای شمارش یکسری رویدادها انتخاب بشه و اینجا نیازه که اون منبع کلاک رو از طریق یکی از پایه های Avr بیایم به feder کلاک شمارندمون متصل کنیم که اینجا سیستم شمارنده ما نقش یک کانتر رو خواهد داشت

Timers ATmega32



در atmega32 ما سه تایمر داریم:

Basic Registers of Timers

- TCNTn **ATmega32** → TCNT0, TCNT1 and TCNT2
- TCNT is counter register (Timer/CouNTer)
  - Counts up with each pulse
  - Upon reset, it contains zero
  - You can load a value into TCNT or read from it
- TOVn **ATmega32** → TOV0, TOV1 and TOV2
- TOV is a flag (Timer OVerflow)
  - When a timer overflows, its TOV flag will be set

در تمام این شمارنده ها و تایمرهایی که در atmega32 است یکسری از این رجیسترها و پرچم ها وجود داره:
اولین رجیستر، رجیستر TCNTn هستش که برای تایمر صفر به صورت TCNT0 و تایمر یک TCNT1 و تایمر دو هم TCNT2 هستش
این TCNT یک رجیستر شمارنده است که توسط اون feeder کلاکی که بهش وارد میشه عملیات شمارش رو انجام میده ینی هر پالسی که وارد میشه اینو تحریک میکنه و یک واحد شمارش میکنه و وقتی سیستم ریست میشه این رجیستر به خودش مقادیر رو صفر میگیره
برای رجیستر TCNT و اون مقداری که داریم شمارش میکنیم در این قرار میگیره این امکان وجود داره که ما بخوایم یک مقدار خاصی رو درون اون بارگذاری کنیم یا از اون بخونیم
پس TCNT میاد اون مقداری که داره شمارش می شه رو درون خودش نگه میداره

برای هر کدوم از اون تایمرها ما یک فلگی داریم که نشان دهنده overflow کردن تایمر یا شمارنده ما هستش که با TOVn شروع میشه و اندیس n نشون میده این پرچم مربوط به کدوم یکی از اون تایمرهاست مثلا TOV0 مال تایمر صفر هستش
وقتی که اون تایمر ما overflow میکنه این پرچم ست میشه

Basic Registers of Timers

- TCCRn **ATmega32** → TCCR0, TCCR1$_{A,B}$ and TCCR2
- TCCR is register (Timer/Counter Control Register)
  - For setting modes of operation
    - Specify a timer to work as a timer or counter
- OCRn **ATmega32** → OCR0, OCR1$_{A,B}$ and OCR2
- OCR is a register (Output Compare Register)
  - The content of OCR is compared with the content of TCNT
  - When they are equal the OCFn flag will be set
- OCFn (Output Compare Flag)

TCCRn این رجیستر میاد و مدهای عملیاتی تایمر رو تعیین میکنه و از طریق این می تونیم اون شمارنده یا اون تایمر رو کنترل بکنیم و اون عملکرد اصلی که اینجا مشخص میکنیم اینه که ما میخوایم این به عنوان یک تایمر عمل بکنه ینی feeder کلاک اون oscillator ما باشه یا اینکه یک منبع خارجی

OCRn این رجیستر یک مقدار رو در خودش ذخیره میکنه که ما می تونیم این رو با اون مقدار درون رجیستر TCNT مقایسه بکنیم و اگر جایی قراره تایمر رو متوقف بکنیم از طریق این رجیستر می تونیم این کارو انجام بدیم پس محتوای OCR با محتوای اون رجیستری که داره شمارش میشه مقایسه میشه و اگر به اون عدد رسید به ما خبر بده و این خبر دادنش از طریق یک فلگی به اسم OCFn انجام میشه

OCFn یک فلگ است و فلگی است که اگر زمانی شمارنده ما به یک عدد خاصی رسید این فلگ میاد و فعال میشه و متناسب با هر کدوم یکی از اون تایمرها هم ما اینجا فلگ داریم مثلا OCF1 مرتبط میشه با تایمر یک ما

Basic Registers of Timers

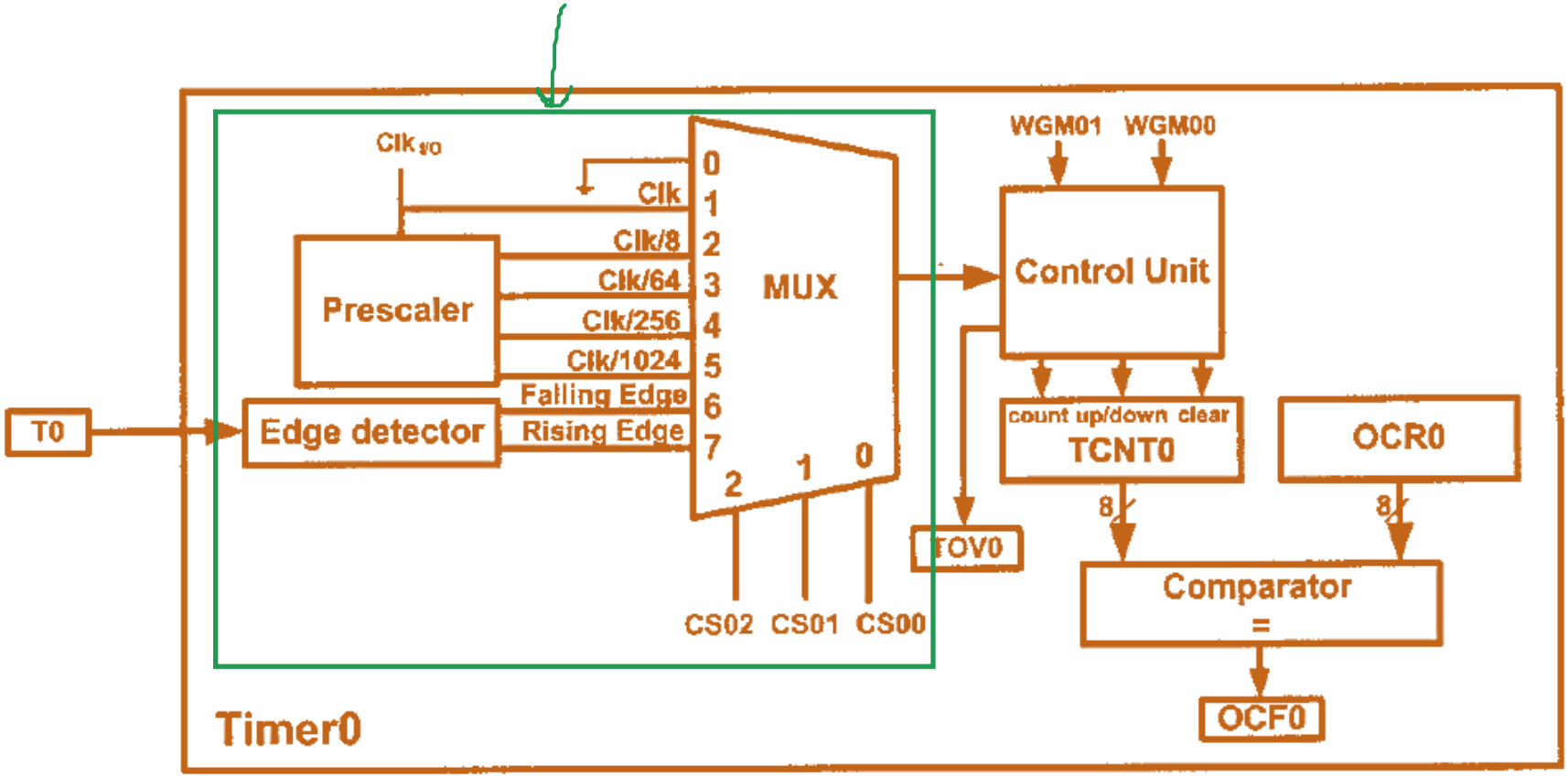
- Timer registers are located in I/O register memory
- You can read or write from timer registers using IN and OUT instructions, like other I/O registers
- For example, load TCNT0 with 25

```
LDI  R20,25        ;R20 = 25
OUT  TCNT0,R20     ;TCNT0 = R20
```

این رجیسترهایی که برای تایمر 0 و 1 و 2 گفتیم این ها همه در اون زیرفضای I/O رجیسترها در داخل فضای مموری قرار میگیرند
و ما میتونیم با دستورات IN , OUT که برای فضای I/O داشتیم بیایم و این هارو مقدار دهی بکنیم
مثلا اگر بخوایم TCNT0 رو با 25 پر بکنیم می تونیم از دستور OUT استفاده بکنیم
که پایینش نوشته : توی این مثال شمارش از 25 شروع میشه و ادامه پیدا میکنه

Timer 0 ATmega32

این feeder کلاک ما هستش



ساختار تایمر صفر:
TCNT ما یک رجیستر 8 بیتی هستش و بیشترین مقداری که میتونه داخل خودش بگیره 255 هستش و اگر از این مقدار بالاتر بره منجر به یک overflow میشه که از طریق پرچم TOV قابل تشخیص هستش
تمام اندیس ها توی این ساختار صفر هستش چون راجع به تایمر صفر داریم حرف می زنیم

Timer0 ATmega32

- Timer0 is 8-bit ⟶ TCNT0 is 8-bit

- TCCR0 (Timer/Counter Control Register)
  - An 8-bit register used for control of timer0

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

نکته:
این تایمر یک تایمر 8 بیتی هستش

رجیستر TCCR0 : این به منظور کنترل این شمارنده مورد استفاده قرار میگیره
این رجیستر هم 8 بیتی هستش و عملکرد تایمر صفر را برای ما کنترل میکنه

TCCR0 (Timer/Counter Control Register)

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force compare match: This is a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM01:00 D5 D4 Compare Output Mode:
These bits control the waveform generator

CS02:00

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

این رجیستر 8 بیت داره که 3 بیت پایین تر برای clock selector هست و دو بیت برای تعیین مد مقایسه ای یا compare outpur mode و بیت شماره 7 هم کمک میکنه یکسری کنترل هایی روی مقایسه ها داشته باشیم و بیت شماره 3 و 6 که مد عملیاتی تایمر را برای ما تعیین میکنه

TCCR0 (Timer/Counter Control Register)

CS02:CS00 (Timer0 clock source)

These bits in the TCCR0 register are used to choose the clock source. If CS02:CS00 = 000, then the counter is stopped. If CS02–CS00 have values between 001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation.

If CS02–CS00 are 110 or 111, the external clock source is used and it acts as a counter.

WGM01:00

Timer0 can work in four different modes: Normal, phase correct PWM, CTC, and Fast PWM. The WGM01 and WGM00 bits are used to choose one of them.

TOV0 (Timer0 Overflow)

The flag is set when the counter overflows, going from $FF to $00. As we will see soon, when the timer rolls over from $FF to 00, the TOV0 flag is set to 1 and it remains set until the software clears it.

```
LDI    R20,0x01
OUT    TIFR,R20      ;TIFR = 0b00000001
```

بیت شماره 0 تا 2: امکان انتخاب منبع کلاک رو برای ما فراهم میکنه مثلا اگر این سه بیت همگی صفر باشند ینی هیچ منبع کلاکی به اون متصل نیست و مثل این است که تایمر ما متوقف باشه و اگر این سه بیت ما بین 1 تا 5 باشه فیدر کلاک ما از کلاک داخلی ینی oscillator تامین میشه که بین این ها هم گزینه های متفاوتی وجود داره مثلا اگر 1 باشه این بودن هیچ اسکیلی کلاک را مستقیما به فیدر کلاک شمارنده متصل میکنه ولی در غیر اینصورت ینی 2 تا 5 اگر بود اون موقع کلاک ما اسکیل باشه که این اسکیل می تونه بین 8 تا 1024 باشه و اگر این کلاک سلکت های ما بین 6 تا 7 باشه این متصل میشه به منابع کلاک خارجی و بسته به این که بیت D0 صفر باشه یا یک باشه به لبه پایین رونده و بالای رونده اون سیگنال احساسه: در صفحه قبل کل این نشون داده شده

از بیت های WGM01:00 : ینی بیت های شماره 3 و 6 ما برای انتخاب مد عملیاتی اون شمارنده استفاده میکنیم این مد میتونه شامل اینا باشه: normal , PWM , CTC , Fast PWM که توی صفحه قبل گفته هر کدوم از اینا برای کدوم است

فلگ TOV0: این فلگ مربوط به تایمر شماره صفر است و وظیفه اش این است که وقتی که کانتر می خواد overflow کنه و زمانی overflow میکنه که به FF رسیدیم و با پالس بعدی این 00 میشه و اینجا جایی است که این فلگ یک میشه و مشخص میکنه که یک overflow رخ داده و این در حالت ست باقی می مونه تا ما این رو به صورت نرم افزاری بخوایم کلیر بکنیم و کلیر این فلگ هم پایین صفحه نوشته : یک را در این فلگ می نویسه و این فلگ کلیر میشه

TIFR

TIFR (Timer/counter Interrupt Flag Register) register

The TIFR register contains the flags of different timers,

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit

OCF1B D3 Timer1 output compare B match flag

OCF1A D4 Timer1 output compare A match flag

ICF1 D5 Input Capture flag

TOV2 D6 Timer2 overflow flag

OCF2 D7 Timer2 output compare match flag

نکته ای که در مورد این فلگ در تایمر شماره 3 و در تایمرهای دیگه وجود داره اینه که: این ها به صورت مجتمع در یک رجیستری قرار دارند تحت عنوان TIFR و ما اگر بخوایم این مقدار رو بخونیم باید از رجیستر TIFR بخونیم و TIFR یک رجیستر 8 بیتی است که فلگ تایمر overflow و... برای تایمرهای متفاوتی توی خودش داره مثلا بیت شماره 0 و 1 مرتبط میشه با تایمر صفر و بیت های 3 تا 5 مرتبط میشن با تایمر شماره 1 و بیت شماره 6 و 7 مرتبط میشه با تایمر شماره 2

پس اون فلگی که مشخص میکرد که چه موقع overflow رخ داده این ها به صورت مجتمع در کنار فلگ های مرتبط با سایر تایمرها در یک رجیستر مجزایی قرار دارند تحت عنوان TIFR
بیت شماره 0: مرتبط با تایمر صفر هستش و هر زمانی که overflow رخ بده این یک است و در غیر این صورت صفر است
بیت شماره 1 : این زمانی یک است که مچ بشه به اون عددی که از قبل مشخص کردیم

Example

Find the value for TCCR0 if we want to program Timer0 in Normal mode, no prescaler. Use AVR's crystal oscillator for the clock source.

TCCR0 =

0	0	0	0	0	0	0	1
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that no prescaler is used.

(a) 10 MHz (b) 8 MHz (c) 1 MHz

(a) F = 10 MHz and T = 1/10 MHz = 0.1 µs
(b) F = 8 MHz and T = 1/8 MHz = 0.125 µs
(c) F = 1 MHz and T = 1/1 MHz = 1 µs

مثال: از به کار گیری رجیستر TCCR و TIFR :
می خوایم رجیستر کنترل را به نحوی برای تایمر صفر تنظیم بکنیم که در حالت normal بدون هیچ گونه اسکیلی در کلاک عمل بکنه
و توی این مثال می خوایم منبع کلاک ما oscillator باشه

کاری که باید بکنیم اینه که: فقط CS00 باید یک باشه

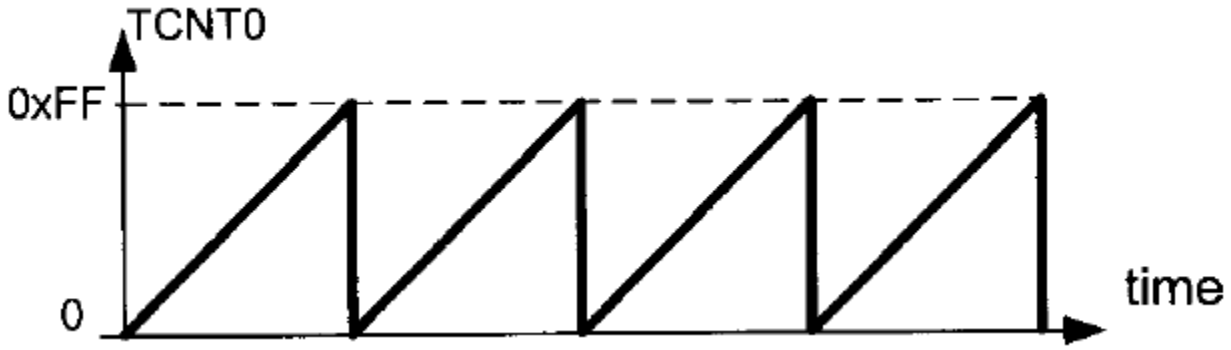
مثال بعدی: میخوایم فرکانس کلاک تایمر را پیدا بکنیم در یکسری از میکروکنترلرهایی که فرکانس oscillator اون ها به صورت گزینه های a , b , c به ما داده شده
و حالتی که اینجا مدنظر داریم اینه که هیچ اسکیلی در کلاک ما وجود نداشته باشه
اگر فرض کنیم oscillator ما با 10 مگاهرتز کار بکنه و بخوایم دوره مرتبط با شمارنده رو پیدا بکنیم معکوس فرکانس رو حساب میکنیم و با توجه به اینکه هیچ اسکیلی هم وجود نداره این دقیقا اون دوره کلاک تایمر رو به ما میده

$$T(\mu s) = \frac{1}{f(MHz)}$$

Mode of Operation

Normal mode

In this mode, the content of the timer/counter increments with each clock. It counts up until it reaches its max of 0xFF. When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0 (Timer Overflow). This timer flag can be monitored.



در مدهای متفاوت عملکردی این شمارنده یا تایمرهای ما اتفاقات متفاوتی می افته:
بررسی مد نورمال:
شمارنده از یک عددی شروع میکنه به شمارش و تا حد خودش می ره و دوباره ریست میشه ینی صفر میشه و دوباره شمارش خودش رو انحام میده تا با سقف خودش برسه و توی این سقف ینی FF زمانی است که overflow رخ میده و اون فلگ ما یک میشه
و این FF هم که اینجا می بینیم بخاطر 8 بیت بودن اون است ینی 8 بیت به صورت FF هگز تفسیر میشه و این شمارنده می تونه از صفر تا FF هگز را شمارش بکنه

Steps to program Timer0 in Normal Mode

1. Load the TCNT0 register with the initial count value.
2. Load the value into the TCCR0 register, indicating which mode (8-bit or 16-bit) is to be used and the prescaler option. When you select the clock source, the timer/counter starts to count, and each tick causes the content of the timer/counter to increment by 1.
3. Keep monitoring the timer overflow flag (TOV0) to see if it is raised. Get out of the loop when TOV0 becomes high.
4. Stop the timer by disconnecting the clock source, using the following instructions:

```
LDI    R20,0x00
OUT    TCCR0,R20    ;timer stopped, mode=Normal
```

5. Clear the TOV0 flag for the next round.
6. Go back to Step 1 to load TCNT0 again.

برای اینکه ما بیایم و تایمر صفر را توی حالت نورمال تنظیم بکنیم باید یکسری گام را طی بکنیم:
1- مقدار تایمر TCNT0 با اون مقدار اولیه که قصد داریم از اون شمارش رو شروع بکنیم پر میکنیم
2- بعد از اینکه عدد اولیه را در رجیستر TCNT0 بارگذاری کردیم بیایم و رجیستر TCCR0 را به نحوی تنظیم بکنیم که مد عملیاتیمون نورمال باشه و اگر میخوایم اسکیلی داشته باشیم در کلاک این را هم باید اینجا تنظیم بکنیم و بعد مرتب چک میکنیم که چه موقع overflow رخ میده و اینجا اتفاقی که داره می افته اینه که شمارنده ما از یک عددی شمارش رو شروع میکنه و می ره که به FF هگز برسه و وقتی که به FF هگز رسید این صفر می شه و اون پرچم TOV0 ما رو یک میکنه
3- این پرچم TOV0 رو پایش میکنیم و هر زمانی که این ست شد متوجه می شیم به اون تاخیری که خواستیم رسیدیم
4- بعد از اینکه تاخیر رو به دست اوردیم می تونیم تایمر رو متوقف بکنیم که این کار با انتخاب 00 برای اون پایه های کلاک سلکت هستش که دستورش توی اسلاید هست
5- بعد از اون میایم پرچم TOV0 را کلیر میکنیم
6- و این باعث میشه که اماده بشه برای دور بعدی این تایمر

Example

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay.

```
.INCLUDE "M32DEF.INC"
.MACRO      INITSTACK          ;set up stack
      LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)
      OUT   SPL,R20
.ENDMACRO
      INITSTACK
      LDI   R16,1<<5     ;R16 = 0x20 (0010 0000 for PB5)
      SBI   DDRB,5       ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17    ;clear PORTB
BEGIN:RCALL DELAY        ;call timer delay
      EOR   R17,R16      ;toggle D5 of R17 by Ex-Oring with 1
      OUT   PORTB,R17    ;toggle PB5
      RJMP BEGIN
;-------------------Time0 delay
DELAY:LDI   R20,0xF2     ;R20 = 0xF2
      OUT   TCNT0,R20    ;load timer0
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20     ;clear TOV0 flag by writing a 1 to TIFR
      RET
```

مثال: یک نمونه از به کارگیری این شمارنده هستش توی همین مدی که الان گفتیم
در ابتدای کد چون میخوایم ساب روتین رو فراخوانی بکنیم میایم استک رو تنظیم میکنیم به نحوی که از بالای حافظه شروع بکنه به پر شدن که این ماکرویی که تعریف کردیم برای ما این کارو میکنه

پورت شماره 5 را به نحوی تنظیم میکنیم که به صورت خروجی باشه و بعد مقدار R17 را صفر میکنیم

و بعد از اون یک ساب روتین داریم که تاخیر ما رو فراهم میکنه: این تاخیر به این صورت انجام میشه که ما یک مقدار اولیه به اون میدیم و این را داخل رجیستر TCNT0 قرار میدیم و بعد اونو در حالت نورمال تنظیم میکنیم و بعد مرتب این حلقه تکرار میشه تا جایی که تایمر ما به حالت ماکسش برسه و بعد با چک کردن اون فلگی که overflow را به ما نشون میده متوجه میشیم که به مقدار نهایی خودمون رسیدیم

و وقتی این کار انجام شد میایم مقدار اون پایه رو با xor کردنش با یک رجیستری که از قبل توی اون یک ریختیم معکوس میکنیم و این کا رو مرتبا تکرار میکنیم

توی این مثال سوال این بود که روی پین شماره 5 پورت B یک موج مربعی 50 درصد ایجاد بکنیم

Example cnt.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay.

In the above program notice the following steps:

1. 0xF2 is loaded into TCNT0.
2. TCCR0 is loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of F3, F4, F5, F6, F7, F8, F9, FA, FB, and so on until it reaches 0xFF. One more clock rolls it to 0, raising the Timer0 flag (TOV0 = 1). At that point, the "`SBRS R20, TOV0`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The TOV0 flag is cleared.


```
.INCLUDE "M32DEF.INC"
.MACRO      INITSTACK         ;set up stack
      LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)
      OUT   SPL,R20
.ENDMACRO
      INITSTACK
      LDI   R16,1<<5    ;R16 = 0x20 (0010 0000 for PB5)
      SBI   DDRB,5      ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17   ;clear PORTB
BEGIN:RCALL DELAY       ;call timer delay
      EOR   R17,R16     ;toggle D5 of R17 by Ex-Oring with 1
      OUT   PORTB,R17   ;toggle PB5
      RJMP BEGIN
;-------------------Time0 delay
DELAY:LDI   R20,0xF2    ;R20 = 0xF2
      OUT   TCNT0,R20   ;load timer0
      LDI   R20,0x01
      OUT   TCCR0,R20   ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR    ;read TIFR
      SBRS  R20,TOV0    ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20   ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20    ;clear TOV0 flag by writing a 1 to TIFR
      RET
```

شرح برنامه صفحه قبل:

Example

In the previous **Example** , calculate the amount of time delay generated by the timer. Assume that XTAL = 8 MHz.

We have 8 MHz as the timer frequency. As a result, each clock has a period of T = 1 /8 MHz = 0.125 μs. In other words, Timer0 counts up each 0.125 μs resulting in delay = number of counts × 0.125 μs.
The number of counts for the rollover is 0xFF – 0xF2 = 0x0D (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FF to 0 and raises the TOV0 flag. This gives 14 × 0.125 μs = 1.75 μs for half the pulse.

مثال:
می خوایم توی مثال قبل ببینیم چه تاخیری خواهیم داشت اگر فرکانس کاری oscillator ما 8 مگاهرتز باشه

اگر این فرکانس برابر با 8 مگاهرتز باشه زمان دوره یک کلاک برای این شمارنده به صورت 1/8 است
ما برای اینکه بخوایم تاخیر دقیق اون حلقه رو محاسبه بکنیم کافیه که تعداد شمارش هارو ضرب در تاخیر یک دوره این کلاک بکنیم
چون در مثال قبلی از F2 هگز شمارش رو شروع می کردیم پس FF - F2 هگز این میاد شمارش میکنه پس میشه 13 گام + یک گام هم از FF به 00 داریم که پرچم ما فعال بشه پس در کل 14 سیکل ساعت داریم پس تاخیر دقیق میشه 0.125 * 14

Example

In the last Example, calculate the frequency of the square wave generated on pin PORTB.5. Assume that XTAL = 8 MHz.

To get a more accurate timing, we need to add clock cycles due to the instructions.

```
                                    Cycles
      LDI   R16,0x20
      SBI   DDRB,5
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY                     3
      EOR   R17,R16                   1
      OUT   PORTB,R17                 1
      RJMP  BEGIN                     2
DELAY:LDI   R20,0xF2                  1
      OUT   TCNT0,R20                 1
      LDI   R20,0x01                  1
      OUT   TCCR0,R20                 1
AGAIN:IN    R20,TIFR                  1
      SBRS  R20,0                 1 / 2
      RJMP  AGAIN                     2
      LDI   R20,0x0                   1
      OUT   TCCR0,R20                 1
      LDI   R20,0x01                  1
      OUT   TIFR,R20                  1
      RET                             4
                                     24
```

$T = 2 \times (14 + 24) \times 0.125\ \mu s = 9.5\ \mu s$ and $F = 1 / T = 105.263$ kHz.

توی این قسمت قصد داریم که یک مقدار دقیق تر بیایم فرکانس اون موج مربعی که ایجاد میکنیم روی پین شماره 5 پورت B محاسبه بکنیم:

Finding the value to be load into timer

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TCNT0 register. To calculate the values to be loaded into the TCNT0 registers, we can use the following steps:

1. Calculate the period of the timer clock using the following formula:
   $T_{clock} = 1/F_{Timer}$
   where F_{Timer} is the frequency of the clock used for the timer. For example, in no prescaler mode, $F_{Timer} = F_{oscillator}$. T_{clock} gives the period at which the timer increments.
2. Divide the desired time delay by T_{clock}. This says how many clocks we need.
3. Perform 256 – *n*, where *n* is the decimal value we got in Step 2.
4. Convert the result of Step 3 to hex, where *xx* is the initial hex value to be loaded into the timer's register.
5. Set TCNT0 = *xx*.

برای اینکه توی کارهای متعدد بیایم و یک تاخیر مشخصی رو ایجاد بکنیم نیاز داریم که رجیستر رو با یک مقدار مناسبی توی حالت نورمال پر بکنیم ولی این که این مقدار چه باشد نیاز به محاسبه داره گام هایی رو برای اینکه اون مقداری که باید در رجیستر TCNT لود بشه تا اون تاخیر مورد نظر رو به ما بده اوردیم:

1- میایم دوره مرتبط با اون تایمر کلاک رو حساب می کنیم. ما اگر معکوس دوره کلاک رو بگیریم اون فرکانس رو به ما میده

در حالتی که از هیچ اسکیلی استفاده نکنیم فرکانس تایمر ما میشه همون فرکانس oscillator در زمان کلاک = این دقیقا یک دوره از کلاک رو به ما میده

2- حالا اگر اون تاخیری که مد نظر داریم بر این دوره تقسیم بکنیم تعداد شمارش هایی که باید انجام بشه رو به ما میده

3- اگر ما این عدد رو از 256 کم بکنیم هموم مقداری رو به ما میده که باید در رجیستر TCNT بارگذاری بکنیم و ما اینو میایم به صورت هگز تبدیل میکنیم و در رجیستر TCNT بارگذاری میکنیم

4- بارگذاری کردن

Example

Assuming that XTAL = 8 MHz, write a program to generate a square wave with a period of 12.5 μs on pin PORTB.3.

For a square wave with T = 12.5 μs we must have a time delay of 6.25 μs. Because XTAL = 8 MHz, the counter counts up every 0.125 μs. This means that we need 6.25 μs / 0.125 μs = 50 clocks. 256 – 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x08
      SBI   DDRB,3       ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D3 of R17
      OUT   PORTB,R17    ;toggle PB3
      RJMP BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0xCE
      OUT   TCNT0,R20    ;load Timer0
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20     ;clear TOV0 flag
      RET
```

مثال:
فرض میکنیم oscillator ما با 8 مگاهرتز کارمیکنه و میخوایم یک برنامه بنویسیم که یک موج مربعی با دوره 12.5 میکروثانیه روی پورت B پین شماره 3 به ما بده

برای اینکه یک موج مربعی با دوره 12.5 داشته باشیم ما نیاز داریم که یک تاخیی با اندازه 6.25 میکروثانیه داشته باشیم که این میشه نیم سیکل ما و کافیه که این نیم سیکل رو ینی 6.25 رو تقسیم بر دوره کلاک ینی 0.125 بکنیم تا تعداد شمارش هارو به دست بیاریم که اینجا میشه 50 ینی ما نیاز داریم که 50 سیکل ساعت بشماریم و بعد از اون 50 رو از 256 کم میکنیم که میشه 206 یا CE هگز که اون مقداری هست که باید در رجیستر TCNT0 قرار بگیره و تایمر کانتر از این عدد شروع میکنه به شمارش تا به FF هگز برسه و بعد به صفر تغییر وضعیت بده و یک فلگی اینجا ست بشه و ما بتونیم براساس چک کردن اون یک دور از اجرا شدن این تاخیر رو متوجه بشیم تا بتونیم دوباره ادامه برنامه رو داشته باشیم و ....

Example

Modify TCNT0 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

To get the largest delay we make TCNT0 zero. This will count up from 00 to 0xFF and then roll over to zero.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x08
      SBI   DDRB,3       ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D3 of R17
      OUT   PORTB,R17    ;toggle PB3
      RJMP  BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0x00
      OUT   TCNT0,R20    ;load Timer0 with zero
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20     ;clear TOV0 flag
      RET
```

می خوایم مثال قبلی رو در حالتی بگیم که بیشترین تاخیر ممکن رو به ما بده:

برای اینکه بیشترین تاخیر رو داشته باشیم شمارش ما باید از صفر شروع بشه ینی از صفر شمارش میکنیم تا به FF برسیم و بعد از FF به صفر می ریم تا اون فلگ یک بشه پس توی تنظیم اون ساب روتین مقدار اولیه برابر با صفر قرار میگیره و رجیستر کنترل هم به نحوی تنظیم می شه که توی حالت نورمال و بدون هیچ اسکیلی کار خودشو انجام بده

Example cnt.

Modify TCNT0 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then will roll over to raise the TCNT0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 0.125 μs = 32 μs. That gives us the smallest frequency of 1 / (2 × 32 μs) = 1 / (64 μs) = 15.625 kHz.

برای مثال صفحه قبل میخوایم این تاخیر رو حساب بکنیم:

اگر از صفر شروع بکنیم به شمارش ینی 256 سیکل ساعت داریم می شماریم
و اگر فرکانس ما 8 مگاهرتز باشه یک دوره اون میشه 0.125 میکروثانیه

تاخیری که اینجا به دست اوردیم بیشترین تاخیری بود که می تونستیم توی مد نورمال بدون هیچ اسکیلی داشته باشیم از یه طرفی این متناظر خواهد بود با کوچکترین فرکانسی که می تونیم توی این حالت به دست بیاریم ینی کوچکترین فرکانس ما اینجا میشه 15.625 کیلوهرتز

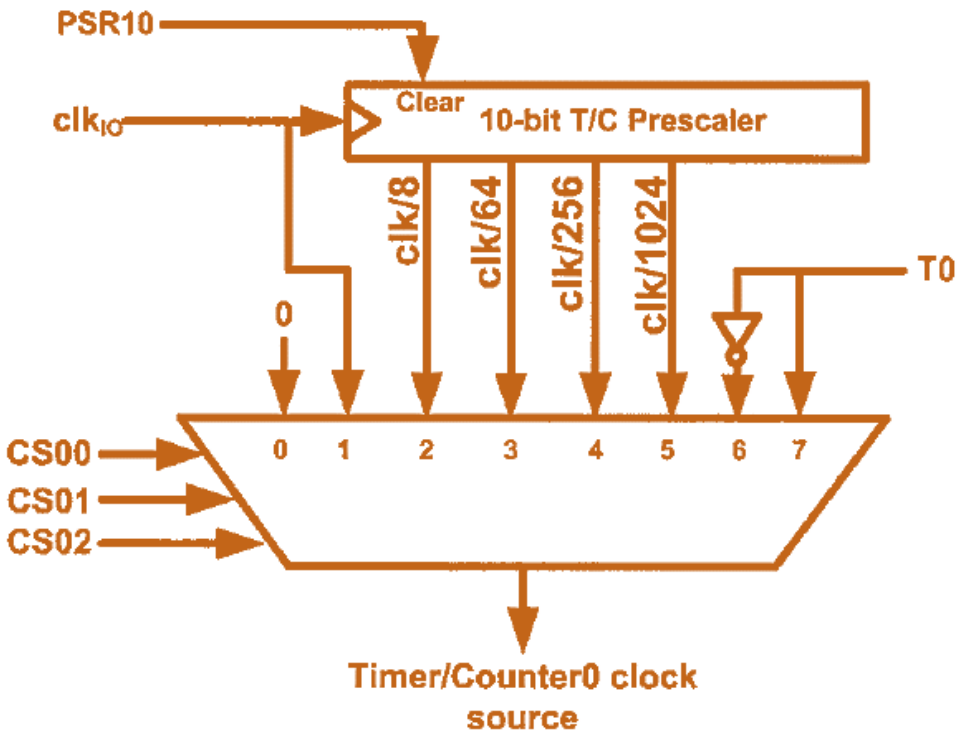
Prescaler and generating a large time delay

- The size of the time of delay depends on
  - Crystal frequency
  - Timer's register
- These factors are beyond the control of AVR programmer
- The largest time delay achieved by making TCNT0 zero
- What if that is not enough?
  - We can use the prescaler option
    - Increase delay by reducing the period

ویژگی های تایمرها در خانواده avr:
یک خاصیت دیگری که میکروکنترلرهای avr در رابطه با تایمرهاشون در اختیار قرار میدند می تونه کلاکی که برای تحریک شمارنده استفاده میشه اسکیل بشه و علتش به این خاطره که اندازه تاخیر و یا میزان شمارشی که یک تایمر و شمارنده می تونه داشته باشه وابسته به دو چیز است: فرکانس کاری اون ینی oscillator که توسط اون شمارنده تحریک میشه و یکی هم اندازه اون رجیسترها وابسته است
این دوتا پارامتر خارج از کنترل برنامه نویس هستش و اگر برنامه نویس بخواد تاخیر بیشتری از اونی که با فرکانس معمولی و اندازه رجیستری که در اختیار داره تولید بکنه دچار مشکل خواهد شد در اینجا حداکثر تاخیری که در حالت استفاده نرمال و با کلاک معمولی می تونه به دست بیاره اینه که مقدار رجیستر تایمر کانتر رو صفر بکنه و این از صفر شروع بکنه به شمارش تا به FF هگز برسه و بعد کلیر بشه (این بیشترین حدی است که ما در حالت عادی با استفاده از این امکانات می تونیم داشته باشیم)
اما اگر این میزان از تاخیر برای ما کافی نباشه اون موقع نیاز به امکانات بیشتری داریم و avr مکانی که برای ما اینجا اورده اینه که می تونیم یک کلاک اسکیل شده رو در اختیار تایمر قرار بدیم در واقع با اسکیل کردن کلاک ما میایم دور ه اون کلاک رو تغییر میدیم و این باعث میشه که تاخیرهای متفاوتی رو بتونیم از تایمرمون بگیریم

Prescaler Option

- Prescaler option of TCCR0 allows us to divide the instruction clock by a factor of 8 to 1024

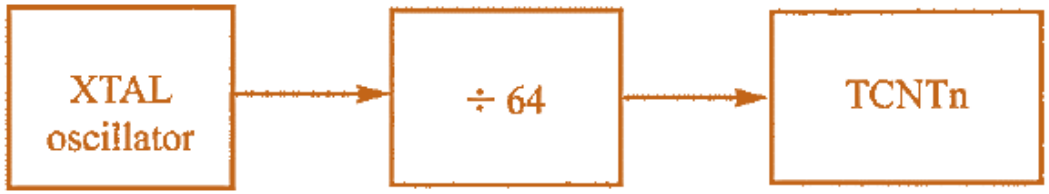


وقتی راجع به تایمر صفر حرف زدیم یک همچنین ساختاری رو در تایمر صفر Avr فراهم اورده ینی علاوه بر خود کلاکی که ما داریم 4 مقیاس از کلاک هم در اختیار ما قرار داده شده که این ها از طریق بیت های کلاک سلکت ینی CS00 , CS01 , CS02 قابل انتخاب اند
مقیاس هایی که تایمر صفر در اختیار ما قرار میده: 8 و 64 و 256 و 1024 هستش اگر ما تنظیم بکنیم سه بیت رو به نحوی که پایه شماره 2 برای ما انتخاب بشه کلاک ما 1/8 میشه ینی شمارش اون تایمر و شمارنده ما به ازای هر 8 کلاکی که توسط اون oscillator انجام میشه یک کلاک به تایمر منتقل خواهد شد و همین طور در رابطه با پایه شماره 3 توی این قسمت 1/64 و توی 4 میشه 1/256 و توی پایه 5 میشه 1/1024 ام است : که این باعث تولید تاخیرهای بیشتری میشه

Example

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that a prescaler of 1:64 is used.

(a) 8 MHz (b) 16 MHz (c) 10 MHz



(a) 1/64 × 8 MHz = 125 kHz due to 1:64 prescaler and T = 1/125 kHz = 8 µs
(b) 1/64 × 16 MHz = 250 kHz due to prescaler and T = 1/250 kHz = 4 µs
(c) 1/64 × 10 MHz = 156.2 kHz due to prescaler and T = 1/156 kHz = 6.4 µs

Find the value for TCCR0 if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0 =	0	0	0	0	0	0	1	1
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

مثال:
ما تایمری که در اختیار داریم در میکروکنترلر با یکی از این سه تا فرکانس داره کار میکنه:
اگر با مقیاس 1/64 ام بخوایم از این تایمر شماره صفر استفاده بکنیم اون دوره مرتبط با این مقیاس چه خواهد بود؟
اگر کلاک ما oscillator ما 8 مگاهرتز باشه و از اسکیل 1/64 استفاده بکنیم این فرکانس 125 کیلوهرتز رو در اختیار ما قرار میده که اگر این فرکانس رو داشته باشیم دوره ما برابر با 8 میکروثانیه خواهد بود و ...

مثال: از کنترل کردن تایمر شماره صفر برای اینکه بتونیم در یکی از این مقیاس هایی که گفتیم استفاده بکنیم و سه بیتی که برای کلاک سلکت داریم به ما میگه کدوم یکی از این پایه ها تحریک کننده ما خواهد بود
اگر حالتی رو بخوایم در نظر بگیریم که در مد نرمال بخوایم شمارش انجام بدیم و اسکیل 1/64 داشته باشیم اون موقع بیت تایمر کنترلر رجیستر رو باید به چه نحوی پر بکنیم؟
چون میخواد در حالت نرمال عمل بکنه این رو تنظیم میکنیم براساس چیزی که قبلا گفتیم و با توجه به اینکه میخوایم 1/64 مقیاس بکنیم کلاکمون رو و چون 64 توی پایه شماره 3 است پس CS00 , CS01 رو یک می ذاریم

Example

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop. Assume XTAL = 8 MHz.

```
.INCLUDE "M32DEF.INC"
       INITSTACK
       LDI    R16,0x08
       SBI    DDRB,3        ;PB3 as an output
       LDI    R17,0
       OUT    PORTB,R17
BEGIN:RCALL DELAY
       EOR    R17,R16       ;toggle D3 of R17
       OUT    PORTB,R17     ;toggle PB3
       RJMP   BEGIN
;---------------- Timer0 Delay
DELAY:LDI    R20,0x10
       OUT    TCNT0,R20     ;load Timer0
       LDI    R20,0x03
       OUT    TCCR0,R20     ;Timer0, Normal mode, int clk, prescaler 64
AGAIN:IN     R20,TIFR      ;read TIFR
       SBRS   R20,TOV0      ;if TOV0 is set skip next instruction
       RJMP   AGAIN
       LDI    R20,0x0
       OUT    TCCR0,R20   ;stop Timer0
       LDI    R20,1<<TOV0
       OUT    TIFR,R20    ;clear TOV0 flag
       RET
```

TCNT0 = 0x10 = 16 in decimal and 256 – 16 = 240. Now 240 × 64 × 0.125 µs = 1920 µs, or we have 240 × 8 µs = 1920 µs.

مثال:
میخوایم تاخیر برنامه رو حساب بکنیم:
10 هگز میشه 16 دسیمال و اگر از 256 تا اینو کم بکنیم میشه 240 تا کلاک که باید طی بکنیم
که شمارنده ما به حد حداکثری خودش برسه و فلگ مرتبط با اونو ست بکنه
کل تاخیر میشه 1920 میکروثانیه
پس با مقیاس 1/64 به تاخیر بیشتری دست پیدا میکنیم

Example

Assume XTAL = 8 MHz. (a) Find the clock period fed into Timer0 if a prescaler option of 1024 is chosen. (b) Show what is the largest time delay we can get using this prescaler option and Timer0.

(a) 8 MHz × 1/1024 = 7812.5 Hz due to 1:1024 prescaler and T = 1/7812.5 Hz = 128 ms = 0.128 ms

(b) To get the largest delay, we make TCNT0 zero. Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then roll over to raise the TOV0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 128 μs = 32,768 μs = 0.032768 seconds.

مثال:
توی این مثال یک میکروکنترلی داریم که با 8 مگاهرتز داره کارمیکنه و اینو به صورت مقیاس 1/1024 میخوایم به شمارنده بدیم و با این مقیاس می خوایم ببینیم دوره تایمر ما چقدر خواهد بود؟
دوره میشه 0.128 میلی ثانیه

b) اگر بخوایم بیشترین تاخیری که با این مقیاس به ما میده رو حساب کنیم میشه:
زمانی این بیشترین تاخیر رو میده که تایمر کانتر ما با صفر پر بشه ینی از صفر شروع بکنه به شمارش ینی 256 تا کلاک باید بگذره که اون فلگ ما ست بشه پس 256 دوره که هر دوره هم شد 128 میکروثانیه و وقتی درهم ضرب میشن میشه تاخیر ما

Example

Assuming XTAL = 8 MHz, write a program to generate a square wave of 125 Hz frequency on pin PORTB.3. Use Timer0, Normal mode, with prescaler = 256.

Look at the following steps:

(a) T = 1 / 125 Hz = 8 ms, the period of the square wave.

(b) 1/2 of it for the high and low portions of the pulse = 4 ms

(c) (4 ms / 0.125 µs) / 256 = 125 and 256 – 125 = 131 in decimal, and in hex it is 0x83.

(d) TCNT0 = 83 (hex)

مثال:
فرکانس oscillator ما 8 مگاهرتز هستش و میخوایم یک برنامه ای بنویسیم که یک موج مربعی با فرکانس 125 هرتز روی پورت B پین شماره 3 تولید بکنه و میخوایم از تایمر صفر توی مد نرمال و مقیاس 256 استفاده بکنیم:
برای درست کردن همچین موجی که فرکانسش 125 هرتز هستش میایم دوره اونو حساب میکنیم که میشه 8 میلی ثانیه = دوره اش که نصفش در حالت low و نصف دیگه هم در حالت high هستش که این رو میگه که ما باید یک تاخیر 4 میلی ثانیه ای رو تولید بکنیم
با توجه به اینکه قصد داریم از مقیاس 256 استفاده بکنیم و فرکانس ما هم 8 مگاهرتز است ما به 125 کلاک اسکیل شده نیاز داریم و در نهایت 131 کلاک نیاز داریم که بتونیم یک همچین تاخیری ینی 4 میلی ثانیه رو تولید بکنیم پس کافیه که 83 هگز در تایمر صفر بارگذاری کنیم
ادامش صفحه بعدی..

Example cnt.

Assuming XTAL = 8 MHz, write a program to generate a square wave of 125 Hz frequency on pin PORTB.3. Use Timer0, Normal mode, with prescaler = 256.

```
        INITSTACK
        LDI    R16,0x08
        SBI    DDRB,3       ;PB3 as an output
        LDI    R17,0
BEGIN:  OUT    PORTB,R17    ;PORTB = R17
        CALL   DELAY
        EOR    R17,R16      ;toggle D3 of R17
        RJMP   BEGIN

;---------------- Timer0 Delay
DELAY:  LDI    R20,0x83
        OUT    TCNT0,R20    ;load Timer0
        LDI    R20,0x04
        OUT    TCCR0,R20    ;Timer0, Normal mode, int clk, prescaler 256

AGAIN:  IN     R20,TIFR     ;read TIFR
        SBRS   R20,TOV0     ;if TOV0 is set skip next instruction
        RJMP   AGAIN

        LDI    R20,0x0
        OUT    TCCR0,R20    ;stop Timer0
        LDI    R20,1<<TOV0
        OUT    TIFR,R20     ;clear TOV0 flag
        RET
```

```
.INCLUDE "M32DEF.INC"
        .MACRO INITSTACK          ;set up stack
        LDI    R20,HIGH(RAMEND)
        OUT    SPH,R20
        LDI    R20,LOW(RAMEND)
        OUT    SPL,R20
.ENDMACRO
```

برنامشو اینجا می بینیم:

Example

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave. Assume XTAL = 8 MHz.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,HIGH(RAMEND)
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16           ;initialize stack pointer
      LDI   R16,0x20
      SBI   DDRB,5            ;PB5 as an output
      LDI   R18,-150
BEGIN:SBI   PORTB,5           ;PB5 = 1
      OUT   TCNT0,R18         ;load Timer0 byte
      CALL  DELAY
      OUT   TCNT0,R18         ;reload Timer0 byte
      CALL  DELAY
      CBI   PORTB,5           ;PB5 = 0
      OUT   TCNT0,R18         ;reload Timer0 byte
      CALL  DELAY
      RJMP BEGIN

;----- Delay using Timer0
DELAY:LDI   R20,0x01
      OUT   TCCR0,R20    ;start Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;monitor TOV0 flag and skip if high
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,1<<TOV0
      OUT   TIFR,R20     ;clear TOV0 flag bit
      RET
```

مثال:
میخوایم فرکانس موج مربعی که توسط این کدی که اینجا اومده محاسبه بکنیم:
اینجا از هیچ اسکیلی استفاده نشده
ادامش صفحه بعدی..

Example cnt.

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave. Assume XTAL = 8 MHz.

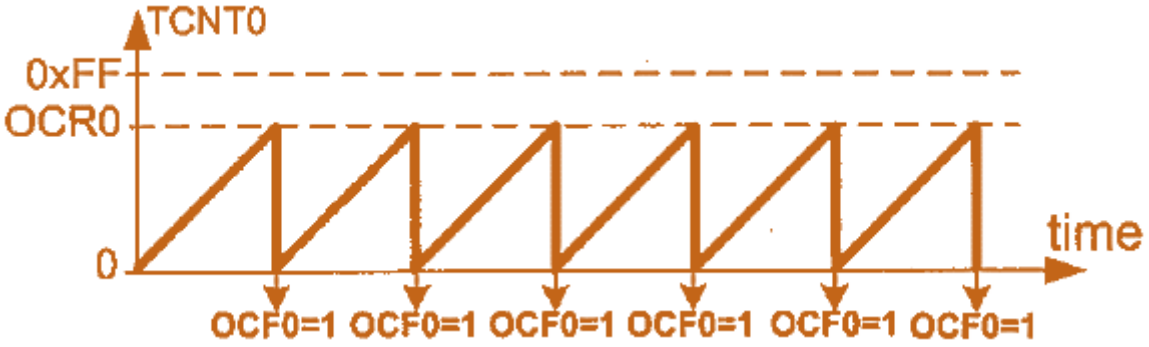
For the TCNT0 value in 8-bit mode, the conversion is done by the assembler as long as we enter a negative number. This also makes the calculation easy. Because we are using 150 clocks, we have time for the DELAY subroutine = 150 × 0.125 μs = 18.75 μs. The high portion of the pulse is twice the size of the low portion (66% duty cycle). Therefore, we have: T = high portion + low portion = 2 × 18.75 μs + 18.75 μs = 56.25 μs and frequency = 1 / 56.25 μs = 17.777 kHz.

به جای اینکه بیایم تعداد سیکل هارو از 256 کم بکنیم از منفی اون تعداد سیکل استفاده بکنیم
ینی به جای اینکه 150-256 بکنیم می تونیم بگیم 150- که این مثل این می مونه که ما داریم 150 رو از 256 کم می کنیم

توی قسمت b میخوایم duty cycle موج رو حساب بکنیم:
می تونیم موج هایی با duty cycle های متفاوت طراحی بکنیم که حتما 50 نباشه و بسته به نیاز می تونیم duty cycle های متفاوتی رو طراحی بکنیم

Clear Timer0 on Compare Match (CTC)

- OCR0 register is used with CTC mode
- In the CTC mode
  - As with the normal mode, the timer is incremented with a clock
  - But it counts up until the content of TCNT0 becomes equal to the content of OCR0
  - Then the timer will be cleared and the OCF0 flag will be set with the next clock



مد CTC:
توی این حالت همون اتفاقاتی می افته که در حالت نرمال هستش منتها یکسری مقایسه های بیشتری را هم داریم که میگیم اینجا:
رجیستر OCR0 : این رجیستر میاد و در مد CTC مورد استفاده قرار میگیره
تایمر با اون چیزی که در حالت نرمال داشتیم با هر پالس ساعت یک واحد افزایش پیدا میکنه و وقتی که محتوای توی این حالت CTC به اون عددی رسید که در داخل OCR هستش اونجا شمارش متوقف میشه و فلگ OCF فعال میشه و در سیکل ساعت بعدی این فلگ ست میشه

اتفاقی که می افته اینه که:
توی حالت نرمال ما از یک عددی شروع میکردیم و این می رفت و به حد ماکزیمم خودش می رسید و اونجا کلیر میشد و این فرایند هی تکرار میشد ولی در CTC اتفاقی که می افته اینه که ما شروع میکنیم به شمارش تا برسیم به اون مقداری که در رجیستر OCR قرار داره و وقتی که به اون رجیستر رسید این کلیر میشه و در این زمان اون پرچم OCF هم فعال میشه و دوباره این اتفاق ادامه پیدا میکنه
پس مد CTC میاد و یک عددی رو به عنوان مبنا قرار میده و شمارنده شروع میکنه به شمارش کردن تا به این عدد برسه و وقتی به این عدد رسید کلیر میکنه و فلگ OCF رو فعال میکنه

Example

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x08
      SBI   DDRB,3            ;PB3 as an output
      LDI   R17,0
BEGIN:OUT   PORTB,R17         ;PORTB = R17
      RCALL DELAY
      EOR   R17,R16           ;toggle D3 of R17
      RJMP  BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0
      OUT   TCNT0,R20
      LDI   R20,9
      OUT   OCR0,R20          ;load OCR0
      LDI   R20,0x09
      OUT   TCCR0,R20         ;Timer0, CTC mode, int clk
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,OCF0          ;if OCF0 is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20         ;stop Timer0
      LDI   R20,1<<OCF0
      OUT   TIFR,R20          ;clear OCF0 flag
      RET
```

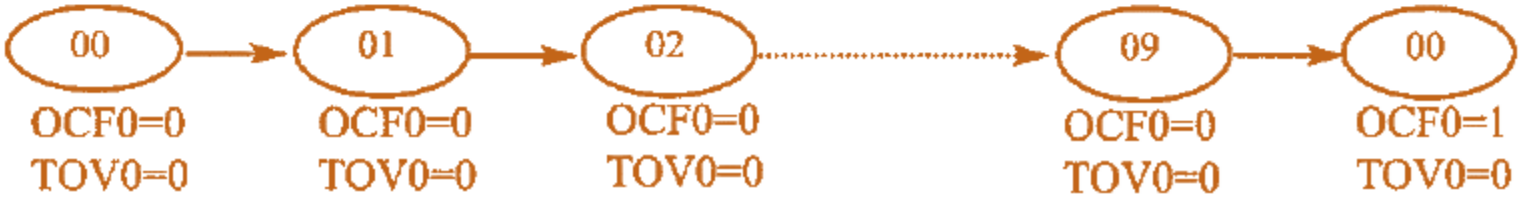
مثال:
می خوایم یک موج 50 درصد رو روی پورت B پین شماره 5 ایجاد بکنیم و می خوایم از تایمر صفر استفاده بکنیم:
اتفاقی که می افته اینجا اینه که ما داریم تایمر صفر رو روی حالتی تنظیم می کنیم که روی مد CTC باشه

Example cnt.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

In the above program notice the following steps:

1. 9 is loaded into OCR0.
2. TCCR0 is loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of 00, 01, 02, 03, and so on until it reaches 9. One more clock rolls it to 0, raising the Timer0 compare match flag (OCF0 = 1). At that point, the "`SBRS R20,OCF0`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The OCF0 flag is cleared.

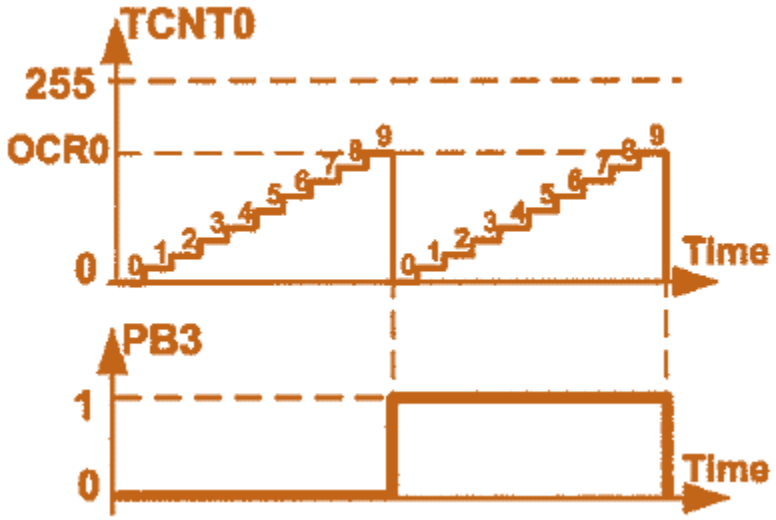


ادامش:
توی این مثال اتفاقی که داره می افته اینه که:
ما 9 رو در درون OCR قرار دادیم
پس شمارنده ما شروع میکنه به شمارش تا به 9 برسه
و وقتی به 9 رسید در سیکل بعدی پرچم OCF رو فعال میکنه و اون شمارش رو خاتمه میده
و به صورت نمادین شکلشو داره پایین نشون میده

Example

Find the delay generated by Timer0 in the last Example. Do not include the overhead due to instructions. (XTAL = 8 MHz)

OCR0 is loaded with 9 and TCNT0 is cleared; Thus, after 9 clocks TCNT0 becomes equal to OCR0. On the next clock, the OCF0 flag is set and the reset occurs. That means the TCNT0 is cleared after 9 + 1 = 10 clocks. Because XTAL = 8 MHz, the counter counts up every 0.125 μs. Therefore, we have 10 × 0.125 μs = 1.25 μs.

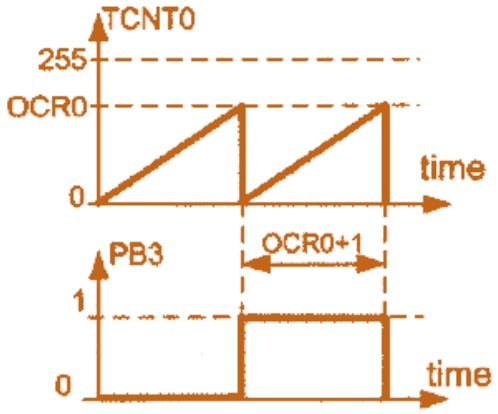


مثال:
می خوایم تاخیری رو که توی برنامه قبلی داشتیم محاسبه بکنیم
OCR روی 9 تنظیم شده بود : 9 کلاک طول میکشه که شمارنده ما به عدد 9 برسه و در سیکل بعدی هم این کلیر میشه و پرچم OCF یک میشه پس 10 کلاک سپری میشه
و توی فرکانس 8 مگاهرتز داریم کار میکنیم و تاخیری که به ما میده 1.25 میکروثانیه است و یک موجی به شکل پایینی برای ما تولید میکنه

Example

Find the delay generated by Timer0 in the following program. Do not include the overhead due to instructions. (XTAL = 8 MHz)

```
.INCLUDE "M32DEF.INC"
      LDI   R16,0x08
      SBI   DDRB,3        ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
      LDI   R20,89
      OUT   OCR0,R20      ;load Timer0
BEGIN:LDI   R20,0x0B
      OUT   TCCR0,R20     ;Timer0, CTC mode, prescaler = 64
AGAIN:IN    R20,TIFR      ;read TIFR
      SBRS  R20,OCF0      ;if OCF0 flag is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20     ;stop Timer0 (This line can be omitted)
      LDI   R20,1<<OCF0
      OUT   TIFR,R20      ;clear OCF0 flag
      EOR   R17,R16       ;toggle D3 of R17
      OUT   PORTB,R17     ;toggle PB3
      RJMP  BEGIN
```



Due to prescaler = 64 each timer clock lasts 64 × 0.125 μs = 8 μs. OCR0 is loaded with 89; thus, after 90 clocks OCF0 is set. Therefore we have 90 × 8 μs = 720 μs.

مثال:
میخوایم تاخیری که توسط تایمر صفر توی این برنامه محاسبه بکنیم و over head مرتبط با دستوراتی که توی حلقه ها است رو هم در نظر نمی گیریم
می خوایم توی مد CTC باشیم و زمانی به حد خودش می رسه که به 89 برسه
اینجا مقیاس 64 هم داریم

Example

Assuming XTAL = 8 MHz, write a program to generate a delay of 25.6 ms. Use Timer0, CTC mode, with prescaler = 1024.

Due to prescaler = 1024 each timer clock lasts 1024 × 0.125 µs = 128 µs. Thus, in order to generate a delay of 25.6 ms we should wait 25.6 ms / 128 µs = 200 clocks. Therefore the OCR0 register should be loaded with 200 – 1 = 199.

```
DELAY:LDI   R20,0
      OUT   TCNT0,R20
      LDI   R20,199
      OUT   OCR0,R20          ;load OCR0
      LDI   R20,0x0D
      OUT   TCCR0,R20         ;Timer0, CTC mode, prescaler = 1024
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,OCF0          ;if OCF0 is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20         ;stop Timer0
      LDI   R20,1<<OCF0
      OUT   TIFR,R20          ;clear OCF0 flag
      RET
```

مثال:
اگر فرکانس ما 8 مگاهرتز باشه می خوایم یک برنامه ای بنویسیم که تاخیر 25.6 میلی ثانیه رو با استفاده از تایمر صفر توی مد CTC با اسکیل 1024 به ما بده

جواب: به 200 کلاک نیاز داریم تا اون تاخیر رو ایجاد بکنیم

Example

Assuming XTAL = 8 MHz, write a program to generate a delay of 1 ms.

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	1 ms/0.125 µs = 8000
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	1 ms/1 µs = 1000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	1 ms/8 µs = **125**
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	1 ms/32 µs = **31.25**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 µs	1 ms/128 µs = **7.8125**

From the above calculation we can only use the options Prescaler = 64, Prescaler = 256, or Prescaler = 1024. We should use the option Prescaler = 64 since we cannot use a decimal point. To wait 125 clocks we should load OCR0 with 125 – 1 = 124.

```
DELAY:LDI   R20,0
      OUT   TCNT0,R20          ;TCNT0 = 0
      LDI   R20,124
      OUT   OCR0,R20           ;OCR0 = 124
      LDI   R20,0x0B
      OUT   TCCR0,R20          ;Timer0, CTC mode, prescaler = 64
AGAIN:IN    R20,TIFR           ;read TIFR
      SBRS  R20,OCF0           ;if OCF0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20          ;stop Timer0
      LDI   R20,1<<OCF0
      OUT   TIFR,R20           ;clear OCF0 flag
      RET
```

مثال:
یک برنامه ای بنویسید که یک تاخیری به اندازه 1 میلی ثانیه با فرض اینکه oscillator ما 8 مگاهرتز فرکانس داره داشته باشیم

اگر فرکانس ما 8 مگاهرتز داشته باشه توی مقیاس های متفاوت میخوایم ببینیم به چند کلاک نیاز داریم:
اگر هیچ مقیاسی نداشته باشیم و کلاک oscillator ما 8 مگاهرتز باشه دوره کلاک میشه 0.125 میکروثانیه میشه و اگر بخوایم تاخیر یک میلی ثانیه داشته باشیم کافیه اینو تقسیم بکنیم که میشه 8000 کلاک که تاخیر یک میلی ثانیه رو تولید بکنیم : توی این حالت هیچ اسکیلی نداریم
حالا اگر اسکیل 8 رو بخوایم استفاده بکنیم نیاز به 1000 تا کلاک داریم که بتونه اون تاخیر رو تولید بکنه
و برای بقیه هم به همین صورت می ریم جلو... که نوشته اونجا
نکته: 8000 و 1000 خارج از اندازه رجیستر تایمر کانتر ما است پس نمی تونیم ازشون استفاده بکنیم و سه تای پایینی هم دوتاش اعشاریه پس بهترین حالتی که می تونیم داشته باشیم همون 125 است
پس مقیاسی که مدنظرمون است 1/64 است پس با این مقیاس ما 125 کلاک نیاز داریم
اینجا مد CTC است

Notice!

Notice that the comparator checks for equality; thus, if we load the OCR0 register with a value that is smaller than TCNT0's value, the counter will miss the compare match and will count up until it reaches the maximum value of $FF and rolls over. This causes a big delay and is not desirable in many cases.

Example:

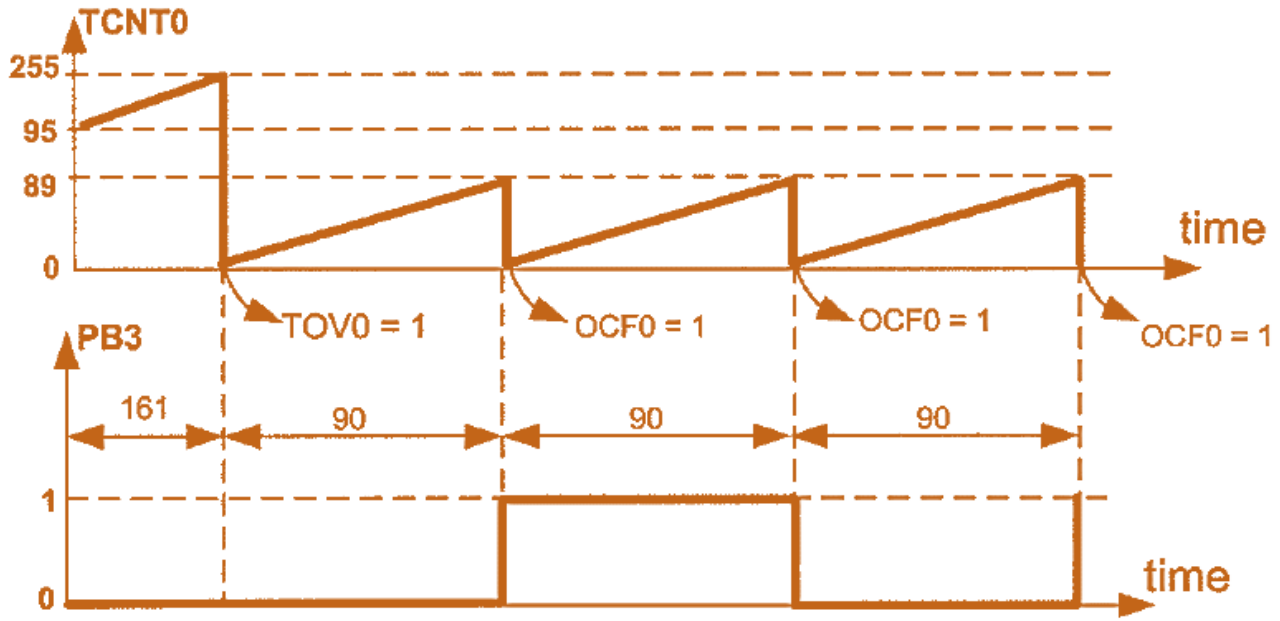
In the following program, how long does it take for the PB3 to become one? Do not include the overhead due to instructions. (XTAL = 8 MHz)

```
.INCLUDE "M32DEF.INC"
        SBI    DDRB,3              ;PB3 as an output
        CBI    PORTB,3             ;PB3 = 0
        LDI    R20,89
        OUT    OCR0,R20            ;OCR0 = 89
        LDI    R20,95
        OUT    TCNT0,R20           ;TCNT0 = 95
BEGIN:  LDI    R20,0x09
        OUT    TCCR0,R20           ;Timer0, CTC mode, prescaler = 1
AGAIN:  IN     R20,TIFR            ;read TIFR
        SBRS   R20,OCF0            ;if OCF0 flag is set skip next inst.
        RJMP   AGAIN
        LDI    R20,0x0
        OUT    TCCR0,R20           ;stop Timer0 (This line can be omitted)
        LDI    R20,1<<OCF0
        OUT    TIFR,R20     ;clear OCF0 flag
        EOR    R17,R16      ;toggle D3 of R17
        OUT    PORTB,R17    ;toggle PB3
        RJMP   BEGIN
```

نکته ای در رابطه با استفاده از تایمر صفر وجود داره و اون هم توی مد CTC:
زمانی که در مد CTC کار میکنیم و قراره اون مقداری که توی شمارنده TCNT ما هست و بیاد با OCR چک بشه اگر احیانا اون عددی که داخل OCR قرار دادیم از عددی که به صورت اولیه در TCNT بارگذاری شده باشه کوچکتر باشه این یکسری مشکلاتی به وجود میاره و توی دور اول یک تاخیر بیشتری رو به ما میده

مثال درباره همین نکته:
ما با OCR رو با 89 پر کردیم و در ادامه میایم رجیستر TCNT رو با 95 پر میکنیم و تنظیم میکنیم که تایمر صفر ما توی مد CTC و با اسکیل یک میخواد کار بکنه
الان مقداری که توی TCNT است بیشتر از OCR است ینی هیچ وقت این مچ نمیشه با این OCR که بخواد اونجا متوقف بشه
پس اتفاقی که می افته اینجا به این صورت است که..... ادامه صفحه بعدی...

Example cnt.



Since the value of TCNT0 (95) is bigger than the content of OCR0 (89), the timer counts up until it gets to $FF and rolls over to zero. The TOV0 flag will be set as a result of the overflow. Then, the timer counts up until it becomes equal to 89 and compare match occurs. Thus, the first compare match occurs after 161 + 90 = 251 clocks, which means after 251 × 0.125 µs = 31.375 µs. The next compare matches occur after 90 clocks, which means after 90 × 0.125 µs = 11.25 µs.

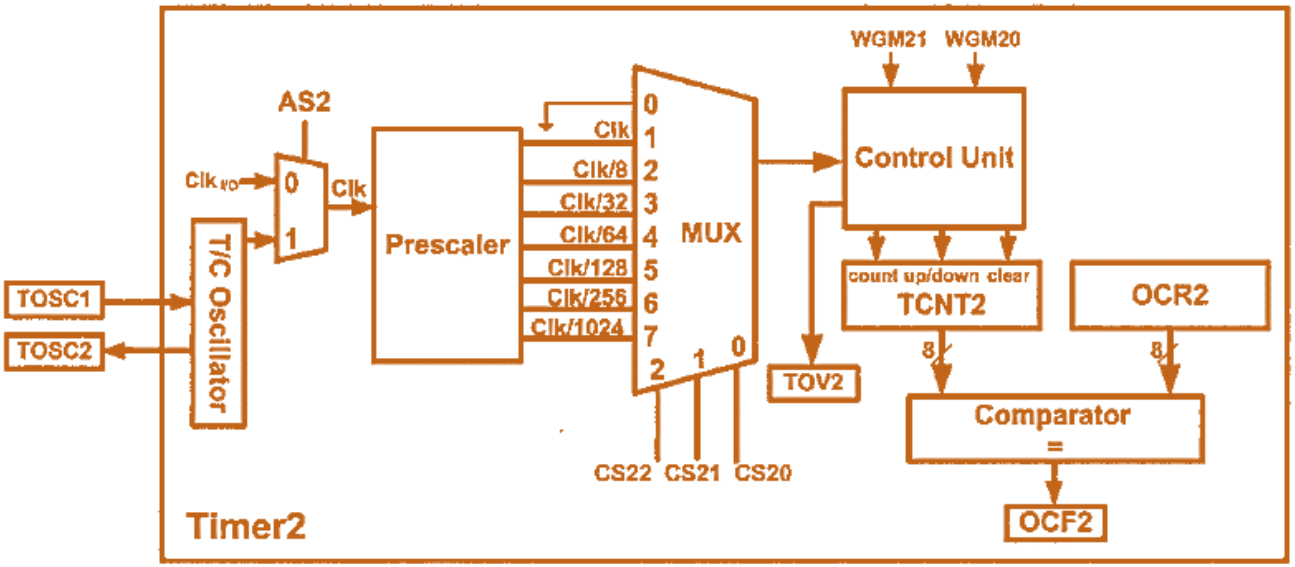
تایمر از 95 شروع میکنه به شمارش به حد ماکس که رسید ینی به 255 که رسید overflow میکنه و توی دوره های بعدی می تونه مچ بشه به اون عددی که توی OCR هست

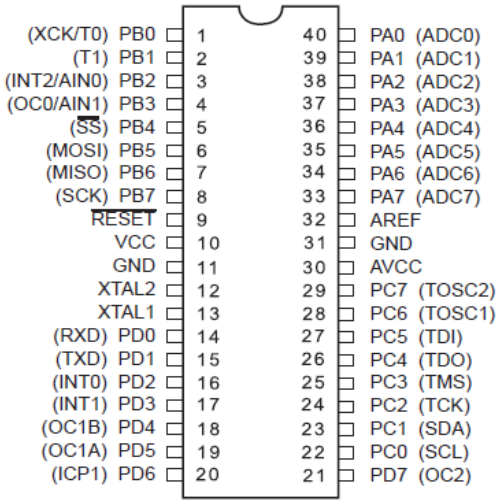
Timer2 ATmega32

Timer2 is an 8-bit timer. Therefore it works the same way as Timer0.
But there are two differences between Timer0 and Timer2:

1. Timer2 can be used as a real time counter. To do so, we should connect a crystal of 32.768 kHz to the TOSC1 and TOSC2 pins of AVR and set the AS2 bit.

2. In Timer0, when CS02–CS00 have values 110 or 111, Timer0 counts the external events. But in Timer2, the multiplexer selects between the different scales of the clock. In other words, the same values of the CS bits can have different meanings for Timer0 and Timer2.





تایمر شماره دو هم یک تایمر 8 بیتی است و عملکرد مشابهی با تایمر صفر داره
اما تفاوت هایی هم وجود داره از جمله تفاوت ها:
گزینه های متفاوت تر اسکیل کلاک هست و امکانات متفاوت تری در خصوص فیدر کلاک
اولین تفاوت: تایمر دو می تونه یک منبع کلاک خارجی ناهمزمان رو هم بپذیره و این از طریق اتصال یک کریستال به پایه های شماره 29 و 28
برای اینکه این تایمر بتونه عملکرد مناسبی داشته باشه بهترین گزینه اینه که یک کریستال با فرکانس 32.768 کیلوهرتز به این پایه ها متصل بکنیم تا تحریک بکنه این شمارنده رو و بتونیم به صورت همزمان یک تایمری هم اینجا داشته باشیم
تفاوت دوم: در تفسیر بیت های کلاک سلکت هستش

Timer2 ATmega32

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC2 D7 Force compare match: a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM20, WGM21

D6	D3	Timer2 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM21:20 D5 D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

CS22:20

D2	D1	D0	Timer2 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

تایمر شماره دو و رجیستر کنترل ان:
اینجا سه بیت داریم برای کلاک سلکت
دو بیت برای انتخاب مد ینی WGM
بیت های مربوط به کنترلر حالتی که می خواد compare انجام بشه
بیت های WGM21 , WGM20 باعث میشه که ما بتونیم بین 4 مد عملیاتی که این تایمر در اختیار ما قرار میده انتخاب داشته باشیم
کلاک سلکت: ..CS که سه بیت داریم:
اگر همش صفر باشه ینی هیچ منبع کلاکی به تایمر ما متصل نیست و اصلا کار نمی کنه تایمر
اگر یک باشه هیچ اسکیلی نداره و کلاک اصلی مستقیما به شمارنده متصل خواهد شد
اگر دو باشه کلاک ما 1/8 خواهد شد
و بقیش هم مثل اسلاید.... توی اسلاید رو بخون

Asynchronous status register (ASSR)

AS2 When it is zero, Timer2 is clocked from $clk_{I/O}$. When it is set, Timer2 works as RTC.

Example (Timer2 Programming)

Find the value for TCCR2 if we want to program Timer2 in normal mode with a prescaler of 64 using internal clock for the clock source.

we have TCCR2 = 0000 0100; XTAL clock source, prescaler of 64.

در ساختار تایمر شماره دو دیدیم که می تونیم منبع کلاک تغذیه کننده این تایمر رو از یک منبع داخلی یا یک منبع ناهمزمان خارجی انتخاب بکنیم و انتخاب بین این دو منبع کلاک از طریق یک بیت کنترلی تحت عنوان AS2 انجام میشه
AS2 چی هست؟ یک رجیستر داریم تحت عنوان ASSR که یک رجیستر 8 بیتی هستش و AS2 یک بیت از این رجیستر است ینی بیت شماره 3 و که این یکسری کنترل هایی در رابطه با تایمر شماره دو برای ما فراهم میکنه:
اگر این بیت شماره 3 رو ست بکنیم این باعث میشه که اون تایمر ما بره و از اون کلاک ناهمزمان خارجی تغذیه بشه

مثال:
نحوی تنظیم تایمر شماره دو:
می خوایم رجیستر کنترلی تایمر دو رو به نحوی برنامه ریزی بکنیم که توی مد نرمال با اسکیل 64 کار بکنه
شکلش پایین است از روش ببین

Example

Using a prescaler of 64, write a program to generate a delay of 1920 µs. Assume XTAL = 8 MHz.

Timer clock = 8 MHz/64 = 125 kHz ➔ Timer Period = 1 / 125 kHz = 8 µs ➔
Timer Value = 1920 µs / 8 µs = 240

```
;----------------- Timer2 Delay
DELAY:LDI   R20,-240     ;R20 = 0x10
      OUT   TCNT2,R20    ;load Timer2
      LDI   R20,0x04
      OUT   TCCR2,R20    ;Timer2, Normal mode, int clk, prescaler 64
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV2     ;if TOV2 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR2,R20    ;stop Timer2
      LDI   R20,1<<TOV2
      OUT   TIFR,R20     ;clear TOV2 flag
      RET
```

مثال:
می خوایم در اسکیل 1/64 ام یک برنامه ای بنویسیم با تایمر شماره دو که تاخیری به میزان 1920 میکروثانیه به ما بده و فرض میکنیم کلاک کنترلر ما 8 مگاهرتز هستش

توضیحاتش توی اسلاید است
نیاز داریم 240 کلاک رو بشماریم

Example

Using CTC mode, write a program to generate a delay of 8 ms. Assume XTAL = 8 MHz.

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	8 ms / 0.125 µs = 64 k
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	8 ms / 1 µs = 8000
32	8 MHz/32 = 250 kHz	1/250 kHz = 4 µs	8 ms / 4 µs = 2000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	8 ms / 8 µs = 1000
128	8 MHz/128 = 62.5 kHz	1/62.5 kHz = 16 µs	8 ms / 16 µs = 500
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	8 ms / 32 µs = **250**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 µs	8 ms / 128 µs = **62.5**

From the above calculation we can only use options Prescaler = 256 or Prescaler = 1024. We should use the option Prescaler = 256 since we cannot use a decimal point. To wait 250 clocks we should load OCR2 with 250 - 1 = 249.

مثال:
از تایمر شماره 2 می خوایم توی مد CTC استفاده بکنیم وفرض میکنیم فرکانس کاری 8 مگاهرتز است و میخوایم یک تاخیری به اندازه 8 میلی ثانیه داشته باشیم:
گزینه هایی که برای انتخاب کلاک داریم:
اینه که هیچ اسکیلی نداشته باشیم یا 8 یا 32 یا ... داشته باشیم (توی اسلاید نوشته)
64K و 8000 و 2000 و 1000 و 500 توی اندازه رجیستر نیست چون رجیستر ما 8 بیتی است
پس از بین دو گزینه 250 و 62.5 (رقم اعشار چندان امکان پذیر نخواهد بود) پس ما 250 رو انتخاب می کنیم

در کل ما باید 250 سیکل ساعت رو بشماریم پس OCR2 با 249 پر میکنیم

Example cnt.

Using CTC mode, write a program to generate a delay of 8 ms. Assume XTAL = 8 MHz.

TCCR2 =

0	0	0	0	1	1	1	0
FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

```
;----------------- Timer2 Delay
DELAY:LDI   R20,0
      OUT   TCNT2,R20          ;TCNT2 = 0
      LDI   R20,249
      OUT   OCR2,R20           ;OCR2 = 249
      LDI   R20,0x0E
      OUT   TCCR2,R20          ;Timer2,CTC mode,prescaler = 256
AGAIN:IN    R20,TIFR           ;read TIFR
      SBRS  R20,OCF2           ;if OCF2 is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR2,R20          ;stop Timer2
      LDI   R20,1<<OCF2
      OUT   TIFR,R20           ;clear OCF2 flag
      RET
```

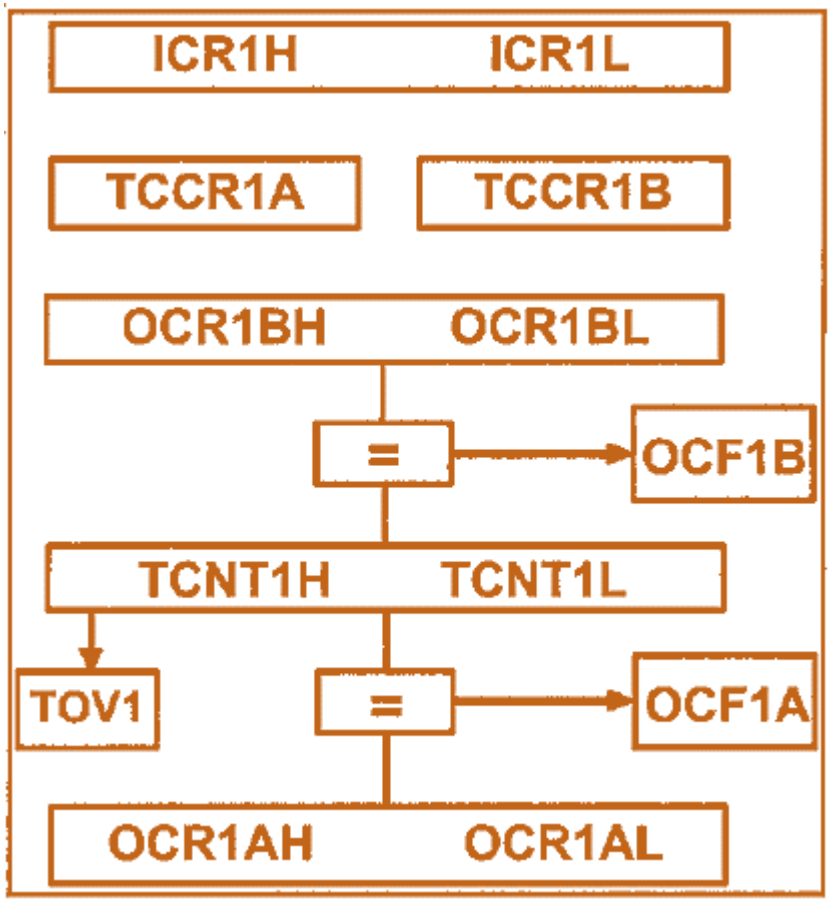
کد مرتبط با مثال قبلی:

Timer1 ATmega32

Since Timer1 is a 16-bit timer its 16-bit register is split into two bytes. These are referred to as TCNT1L (Timer1 low byte) and TCNT1H (Timer1 high byte).

Timer1 also has two control registers named TCCR1A (Timer/counter 1 control register) and TCCR1B. The TOV1 (timer overflow) flag bit goes HIGH when overflow occurs. Timer1 also has the prescaler options of 1:1, 1:8, 1:64, 1:256, and 1:1024.

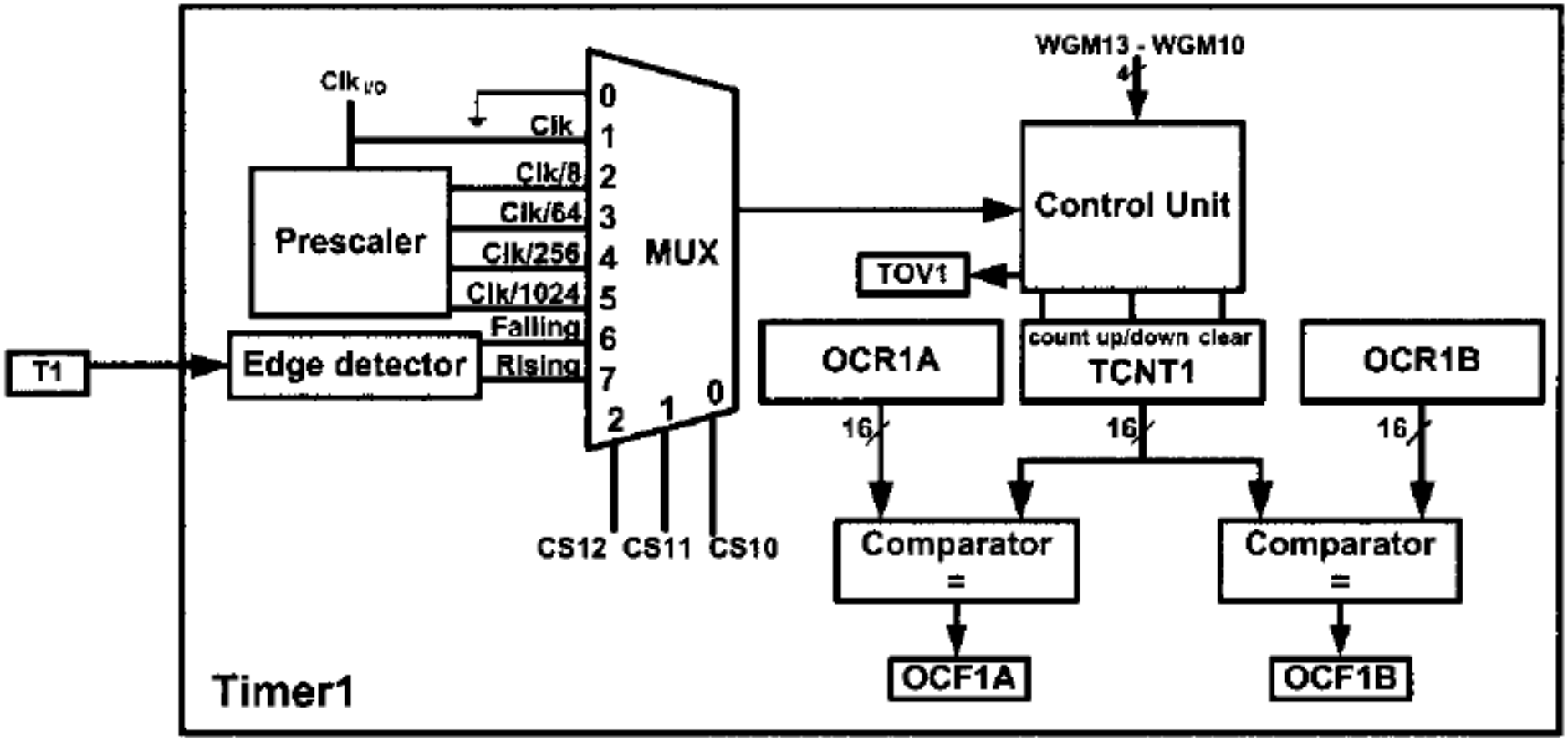
There are two OCR registers in Timer1: OCR1A and OCR1B. There are two separate flags for each of the OCR registers, which act independently of each other.



تایمر شماره یک:
این تایمر یک تایمر 16 بیتی هستش و از اونجایی که میکروکنترلر AVR یک میکروکنترلر 8 بیتی است پس ما 16 بیت رو به دو قسمت 8 بیتی تقسیم می کنیم
اینجا تایمر TCNT رو به دو قسمت high , low تقسیم میکنیم: TCNT1L , TCNT1H
اینجا هم دوتا رجیستر TCCR داریم که با نام A , B شناخته میشه ینی TCCR1A , TCCR1B
و یک TOV1 هم داریم = و زمانی که تایمر ما overflow میکنه این پرچم یک خواهد شد
از امکاناتی که توی تایمر شماره یک هستش اینه که ما اسکیل های متفاوتی داریم : 1 و 1/8 و 1/64 و 1/256 و 1/1024 ام هستش
و دوتا رجیستر OCR داره که با OCR1A , OCR1B شناخته میشه متناظر با هر کدوم از این رجیسترهای OCR ما یک فلگ داریم که به نام OCF1A , OCF1B که هر کدوم متناظر با OCR ها دیده شده

شکلش هم توی صفحه است:
توی این تایمر یک رجیستر اضافی هم داریم تحت عنوان ICR که برای کپچر کردن ورودی مورد استفاده قرار میگیره

Timer1 ATmega32



ساختار کلی اش:

Timer1 ATmega32

The TIFR register contains the TOV1, OCF1A, and OCF1B flags.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit

OCF1B D3 Timer1 output compare B match flag

OCF1A D4 Timer1 output compare A match flag

ICF1 D5 Input Capture flag

TOV2 D6 Timer2 overflow flag

OCF2 D7 Timer2 output compare match flag

پرچم های OCF:
ما اینجا دو تا پرچم OCF داریم برای تایمر شماره یک
یک فلگ TOV داریم و یک فلگ ICF داریم پس برای این تایمر ما در کل 4 تا فلگ در رجیستر TIFR مون داریم
OCF1A , OCF1B برای اون حالت compare هستش که برای OCR های A , B وجود داره
یک دونه هم TOV داریم که به صورت مشترک داره استفاده میشه

TCCR

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

COM1A1:COM1A0 D7 D6 Compare Output Mode for Channel A (discussed in Section 9-3)

COM1B1:COM1B0 D5 D4 Compare Output Mode for Channel B (discussed in Section 9-3)

FOC1A D3 Force Output Compare for Channel A (discussed in Section 9-3)

FOC1B D2 Force Output Compare for Channel B

WGM11:10 D1 D0 Timer1 mode

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	**TCCR1B**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICNC1 D7 Input Capture Noise Canceler
0 = Input Capture is disabled.
1 = Input Capture is enabled.

ICES1 D6 Input Capture Edge Select
0 = Capture on the falling (negative) edge
1 = Capture on the rising (positive) edge

D5 Not used

WGM13:WGM12 D4 D3 Timer1 mode

رجیستر TCCR:
این رجیستر 16 بیت داره و در دو قسمت 8 بیتی تقسیم شده
و در قسمت پایین تر اون ما بیت های کنترلی و حالتی که میخوایم compare داشته باشیم و بیت شماره 0 و 1 اش برای تنظیم مد است

توی قسمت بالاتر دو بیت WGM میاد در کنار سه بیت کلاک سلکت قرار می گیره
کلاک سلکت گزینه های متفاوتی داره

در حالت کلی ما اینجا 4 بیت برای انتخاب مد داریم و داشتن 4 بیت برای انتخاب مد به معنای این هستش که ما 16 مد عملیاتی می تونیم داشته باشیم (که در صفحه بعدی است...)

TCCR

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

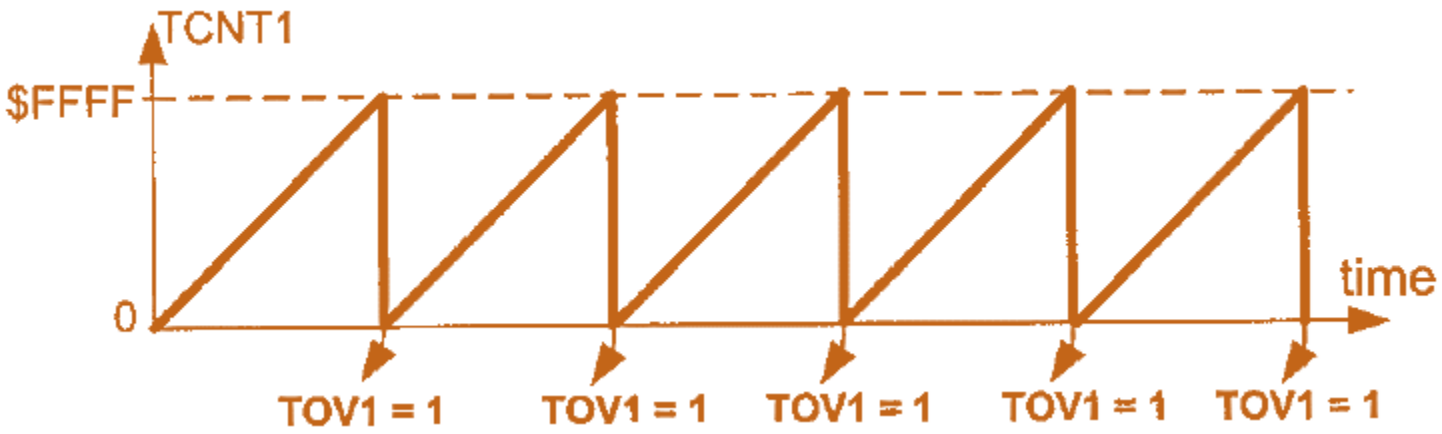
CS12:CS10	D2 D1 D0	Timer1 clock selector
	0 0 0	No clock source (Timer/Counter stopped)
	0 0 1	clk (no prescaling)
	0 1 0	clk / 8
	0 1 1	clk / 64
	1 0 0	clk / 256
	1 0 1	clk / 1024
	1 1 0	External clock source on T1 pin. Clock on falling edge.
	1 1 1	External clock source on T1 pin. Clock on rising edge.

که یک مد اون به صورت reserved نگه داشته شده ینی حالتی که مقدار WGM11 ما صفر هستش و بقیه یک اند

کلاک سلکت رو هم ینی CS رو پایین نشون داده...

Normal Mode (WGM13:10=0000)

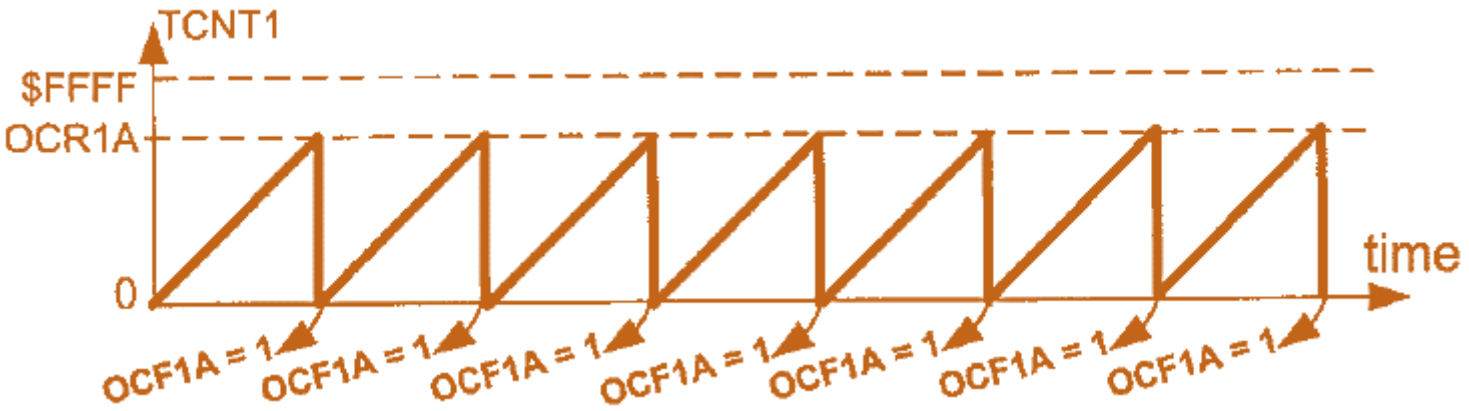
In this mode, the timer counts up until it reaches $FFFF (which is the maximum value) and then it rolls over from $FFFF to 0000. When the timer rolls over from $FFFF to 0000, the TOV1 flag will be set.



مدهای عملیاتی تایمر یک:
مشابه قبلی ها است
توی بیت نرمال سقف ما FFFF است
این مد رو زمانی داریم که WGM10 تا WGM13 ما با 0000 پر شده باشه

CTC Mode (WGM13:10=0100)

In mode 4, the timer counts up until the content of the TCNT1 register becomes equal to the content of OCR1A (compare match occurs); then, the timer will be cleared when the next clock occurs. The OCF1A flag will be set as a result of the compare match as well.



مد CTC:
توی این مد میایم WGM ها رو با 0100 پر می کنیم
مشابه قبلی ها است

Example

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 0 (Normal), with no prescaler. Use AVR's crystal oscillator for the clock source.

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 0001 WGM13 = 0, WGM12 = 0, oscillator clock source, no prescaler

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 4 (CTC, Top = OCR1A), no prescaler. Use AVR's crystal oscillator for the clock source.

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 1001 WGM13 = 0, WGM12 = 1, oscillator clock source, no prescaler

مثال:
میخوایم تایمر شماره یک رو به نحوی تنظیم بکنیم که توی مد نرمال و بدون هیچ اسکیلی بخواد کار بکنه:

مثال:
میخوایم تایمر یک رو در مد 4 ینی مد CTC فعال بکنیم و هیچ اسکیلی هم نداشته باشیم:

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x20
      SBI   DDRB,5       ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17    ;PB5 = 0
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D5 of R17
      OUT   PORTB,R17    ;toggle PB5
      RJMP  BEGIN
;--------------- Timer1 delay
DELAY:LDI   R20,0xD8
      OUT   TCNT1H,R20   ;TCNT1H = 0xD8
      LDI   R20,0xF0
      OUT   TCNT1L,R20   ;TCNT1L = 0xF0
      LDI   R20,0x00
      OUT   TCCR1A,R20   ;WGM11:10 = 00
      LDI   R20,0x01
      OUT   TCCR1B,R20   ;WGM13:12 = 00, Normal mode, prescaler = 1
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV1     ;if TOV1 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR1B,R20   ;stop Timer1
      LDI   R20,0x04
      OUT   TIFR,R20     ;clear TOV1 flag
      RET
```

مثال:
میخوایم فرکانس موج مربعی که این برنامه داره اینجا تولید میکنه رو محاسبه کنیم:
فرض میکنیم فرکانس کلاک ما 8 مگاهرتز هستش

جواب میشه صفحه بعدی...

Example cnt.

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

WGM13:10 = 0000 = 0x00, so Timer1 is working in mode 0, which is Normal mode, and the top is 0xFFFF.
FFFF + 1 – D8F0 = 0x2710 = 10,000 clocks, which means that it takes 10,000 clocks.
As XTAL = 8 MHz each clock lasts 1/(8M) = 0.125 μs and delay = 10,000 × 0.125 μs = 1250 μs = 1.25 ms and frequency = 1 / (1.25 ms × 2) = 400 Hz.
In this calculation, the overhead due to all the instructions in the loop is not included.

Notice that instead of using hex numbers we can use HIGH and LOW directives, as shown below:

```
LDI   R20,HIGH (65536-10000)        ;load Timer1 high byte
OUT   TCNT1H,R20  ;TCNT1H = 0xD8
LDI   R20,LOW (65536-10000)         ;load Timer1 low byte
OUT   TCNT1L,R20  ;TCNT1L = 0xF0
```

or we can simply write it as follows:

```
LDI   R20,HIGH (-10000)             ;load Timer1 high byte
OUT   TCNT1H,R20  ;TCNT1H = 0xD8
LDI   R20,LOW (-10000)              ;load Timer1 low byte
OUT   TCNT1L,R20  ;TCNT1L = 0xF0
```

با توجه به این که WGM های ما با صفر پر شدن ینی مد نرمال رو داریم

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,5            ;PB5 as an output
BEGIN:SBI   PORTB,5           ;PB5 = 1
      RCALL DELAY
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY
      RJMP  BEGIN
;---------------- Timer1 delay
DELAY:LDI   R20,0x00
      OUT   TCNT1H,R20
      OUT   TCNT1L,R20        ;TCNT1 = 0
      LDI   R20,0
      OUT   OCR1AH,R20
      LDI   R20,159
      OUT   OCR1AL,R20        ;OCR1A = 159 = 0x9F
      LDI   R20,0x0
      OUT   TCCR1A,R20        ;WGM11:10 = 00
      LDI   R20,0x09
      OUT   TCCR1B,R20        ;WGM13:12 = 01,CTC mode, prescaler = 1
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,OCF1A         ;if OCF1A is set skip next instruction
      RJMP  AGAIN
      LDI   R20,1<<OCF1A
      OUT   TIFR,R20          ;clear OCF1A flag
      LDI   R19,0
      OUT   TCCR1B,R19        ;stop timer
      OUT   TCCR1A,R19
      RET
```

مثال:
میخوایم فرکانس موج مربعی رو حساب بکنیم که این برنامه ای که اینجا هستش در اختیار ما قرار میده
و overhead دستورات اضافی تری هم که توی این توابع تاخیر داریم رو نمیخوایم مورد محاسبه قرار بدیم

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

WGM13:10 = 0100 = 0x04 therefore, Timer1 is working in mode 4, which is a CTC mode, and max is defined by OCR1A.
159 + 1 = 160 clocks
XTAL = 8 MHz, so each clock lasts 1/(8M) = 0.125 μs.
Delay = 160 × 0.125 μs = 20 μs and frequency = 1 / (20 μs × 2) = 25 kHz.
In this calculation, the overhead due to all the instructions in the loop is not included.

جواب صفحه قبلی...
اینجا داریم تاخیر رو محاسبه می کنیم

Accessing 16-bit Registers

The AVR is an 8-bit microcontroller, which means it can manipulate data 8 bits at a time, only. But some Timer1 registers, such as TCNT1, OCR1A, ICR1, and so on, are 16-bit; in this case, the registers are split into two 8-bit registers, and each one is accessed individually. This is fine for most cases. For example, when we want to load the content of SP (stack pointer), we first load one half and then the other half, as shown below:

```
LDI   R16, 0x12
OUT   SPL, R16
LDI   R16, 0x34
OUT   SPH, R16    ;SP = 0x3412
```

In 16-bit timers, however, we should read/write the entire content of a register at once, otherwise we might have problems. For example, imagine the following scenario:

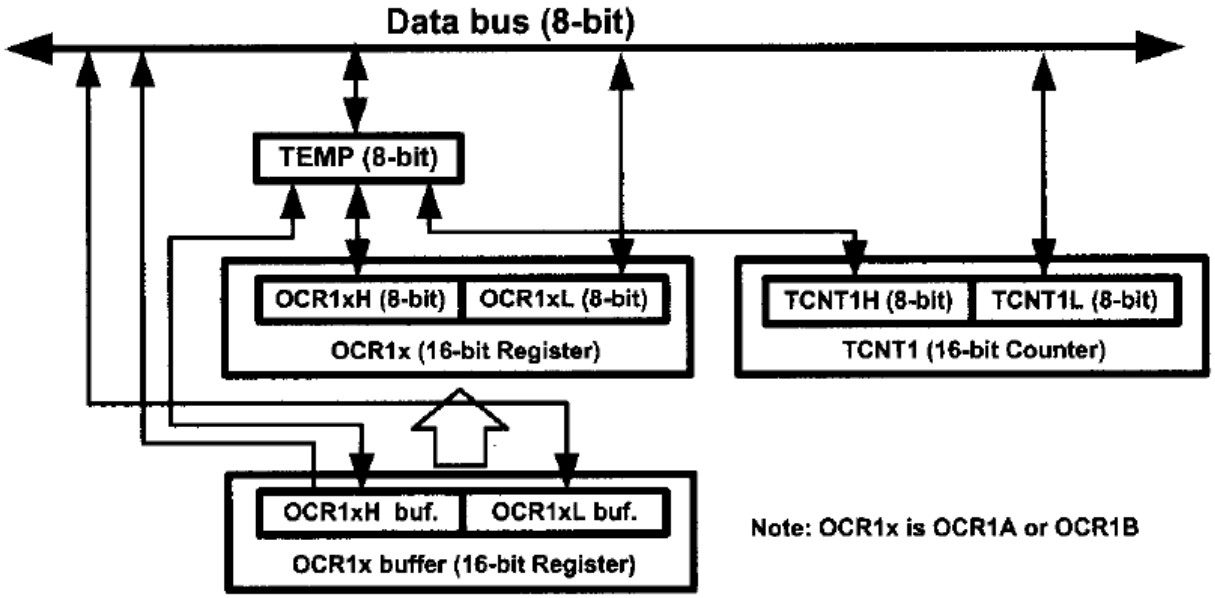
The TCNT1 register contains 0x15FF. We read the low byte of TCNT1, which is 0xFF, and store it in R20. At the same time a timer clock occurs, and the content of TCNT1 becomes 0x1600; now we read the high byte of TCNT1, which is now 0x16, and store it in R21. If we look at the value we have read, R21:R20 = 0x16FF. So, we believe that TCNT1 contains 0x16FF, although it actually contains 0x15FF.

اولین نکته که درباره تایمر به خصوص تایمر شماره 1 وجود داره اینه که این تایمر 16 بیتی هستش ولی میکروکنترلر ما 8 بیتی است
در بسیاری از موارد دیگر هم ما رجیسترهای 16 بیتی هم داریم که با اونها کار میکنیم و هیچ مشکلی نیست مثلا استک پوینتر الان مثلا توی کد روبه رو ما اومدیم استک پوینتر رو قسمت پایین و بالاش رو به صورت مجزا با یک عددی که مورد نظرمون بوده بارگذاری کردیم
ما در مورد تایمر شماره 1 که رجیسترهای ICR , OCR , TCNT اش 16 بیتی هستند یک مقدار ممکنه در بعضی از کاربردها این نوع داده بارگذاری که مثل استک داشتیم در درون رجیسترهای مشکل ساز بشه برای همین نیازه که یک مقدار تمدیدات بیشتری در این خصوص اندیشیده بشه
یکی از مشکلاتی که میشه بهش اشاره کرد:
مثلا میخوایم با رجیستر TCNT کار بکنیم و محتوای 15FF هگز رو در درون خودش داره و اگر بخوایم ابتدا قسمت پایین اون رو ینی FF رو بخونیم و بعد بخوایم قسمت بالای اون رو بخونیم ممکنه در این بین یک کلاک به تایمر وارد بشه و اون یک شمارش انجام بده --> الان ما توی موقعیتی هستیم که قسمت پایین FF داریم و یک کلاک بیاد این تبدیل میشه به 1600 هگز و بعد اگر بیایم قسمت بالای اون رو بخونیم 16 هگز رو میخونیم و عددی که به این صورت خوانده میشه تهش 16FF هگز هستش که این یک مقدار اشتباهی هست
برای رفع این مشکل.... صفحه بعدی...

Accessing 16-bit Registers

This problem exists in many 8-bit microcontrollers. But the AVR designers have resolved this issue with an 8-bit register called TEMP, which is used as a buffer. When we write or read the high byte of a 16-bit register, such as TCNT1, the value will be written into the TEMP register. When we write into the low byte of a 16-bit register, the content of TEMP will be written into the high byte of the 16-bit register as well. For example, consider the following program:

```
LDI   R16, 0x15
OUT   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
LDI   R16, 0xFF
OUT   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
```



ادامه....
برای همین در میکروکنترلر برای رفع این مشکل یک رجیستر تحت عنوان رجیستر TEMP استفاده میشه به این صورت که داده ها به صورت موقت ذخیره بشه و باعث میشه که این مشکلی که الان گفتیم رفع بشه
نحوه کار به این صورته که ما ابتدا میایم قسمت بالای اون رجیستری که میخوایم بخونیم مثلا رجیستر TCNT1H رو میخونیم و این به صورت اتوماتیک در رجیستر TEMP ذخیره میشه و زمانی که میخوایم اون قسمت پایین رو بخونیم و در جایی بنویسیم اینا همزمان در اون قسمت نوشته میشه

شکل رو ببین قسمت بالای TCNT1H به TEMP وصل است و قسمت پایینش ینی TCNT1L به باس وصله

Accessing 16-bit Registers cnt.

After the execution of "OUT TCNT1H, R16", the content of R16, 0x15, will be stored in the TEMP register. When the instruction "OUT TCNT1L, R16" is executed, the content of R16, 0xFF, is loaded into TCNT1L, and the content of the TEMP register, 0x15, is loaded into TCNT1H. So, 0x15FF will be loaded into the TCNT1 register at once.

Notice that according to the internal circuitry of the AVR, we should first write into the high byte of the 16-bit registers and then write into the lower byte. Otherwise, the program does not work properly. For example, the following code:

```
LDI   R16, 0xFF
OUT   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
LDI   R16, 0x15
OUT   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
```

does not work properly. This is because, when the TCNT1L is loaded, the content of TEMP will be loaded into TCNT1H. But when the TCNT1L register is loaded, TEMP contains garbage (improper data), and this is not what we want.

ادامه...
بعد از اینکه دستور OUT رو انجام میده برای قسمت بالا محتوای اون رجیستری که مدنظر ما بوده میاد و ذخیره میشه و در قسمت بعدی که میخوایم قسمت LOW رو بریزیم توی اون رجیستر مورد نظرمون به صورت اتوماتیک این محتوای رجیستر TEMP هم میاد همزمان در رجیستر مقصد ریخته میشه
پس توی این حالت اول میایم قسمت high رو می خونیم و بعد قسمت low رو و اگر این ترتیب رو رعایت نکنیم ینی اول قسمت low رو بخونیم و بعد قسمت high رو بخونیم اون موقع مقداری از محتوای TEMP که می خواد توی رجیستر مقصد ریخته بشه ممکنه اون مقداری نباشه که ما مورد نظر داشتیم پس برای اینکه بخوایم از این ساختار استفاده بکنیم نیازه که حتما این ترتیب رو رعایت بکنیم

Atomic Read Of TCNT1

Assembly Code Example

```
TIM16_ReadTCNT1:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Read TCNT1 into r17:r16
  in  r16,TCNT1L
  in  r17,TCNT1H
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

Counter Programming

- Use of timer as an event counter
- Source of frequency
  - timer→ AVR's crystal
  - Counter→ pulse outside the AVR
- In counter mode
  - Registers such as TCNT, TCCR and OCR0 are the same as for the timer

ما میتونیم از تایمرها به دو منظور استفاده بکنیم:
1- استفاده تایمری ینی یک تاخیر رو تولید بکنیم
2- یا شمارش یکسری رویدادها

دراین قسمت میخوایم از تایمرها برای شمارش یکسری رویدادها صحبت بکنیم و ببینیم چجوری اونارو برنامه ریزی بکنیم که یکسری رویدادهای خارجی را برای ما شمارش بکنن

توی ساختار تایمرها دو منبع فرکانس یا کلاک رو می تونیم برای تایمرها در نظر بگیریم:
یا منبع داخلی
یا یک رویداد خارجی

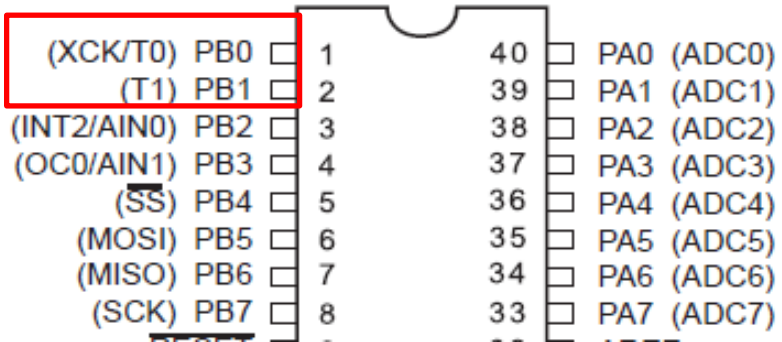
اگر بخوایم رویدادهای خارجی رو شمارش بکنیم توی این حالت به این سیستم میگن counter و باید به گونه ای تنظیم بشه که پالس های یک رویداد خارجی رو پایش بکنه و اون هارو شمارش بکنه توی این حالت ینی counter: رجیسترهایی که قبلا توی ساختار تایمرها دیدیم همون ها هستند و با همون ها کار میکنیم ینی رجیسترهای TCNT , TCCR , OCR رو داریم

Counter Programming

TCCR0 register decide the source of the clock for the timer. If CS02:00 is between 1 and 5, the timer gets pulses from the crystal oscillator. In contrast, when CS02:00 is 6 or 7, the timer is used as a counter and gets its pulses from a source outside the AVR chip.

Therefore, when CS02:00 is 6 or 7, the TCNT0 counter counts up as pulses are fed from pin T0 (Timer/Counter 0 External Clockinput). In ATmega32/ATmega16, T0 is the alternative function of PORTB.0.

In the Timer0, when CS02:00 is 6 or 7, pin T0 provides the clock pulse and the counter counts up after each clock pulse coming from that pin. Similarly, for Timer1, when CS12:10 is 6 or 7, the clock pulse coming in from pin T1 (Timer/Counter 1 External Clock input) makes the TCNT1 counter count up. When CS12:10 is 6, the counter counts up on the negative (falling) edge. When CS12:10 is 7, the counter counts up on the positive (rising) edge. In ATmega32/ATmega16, T1 is the alternative function of PORTB.1.



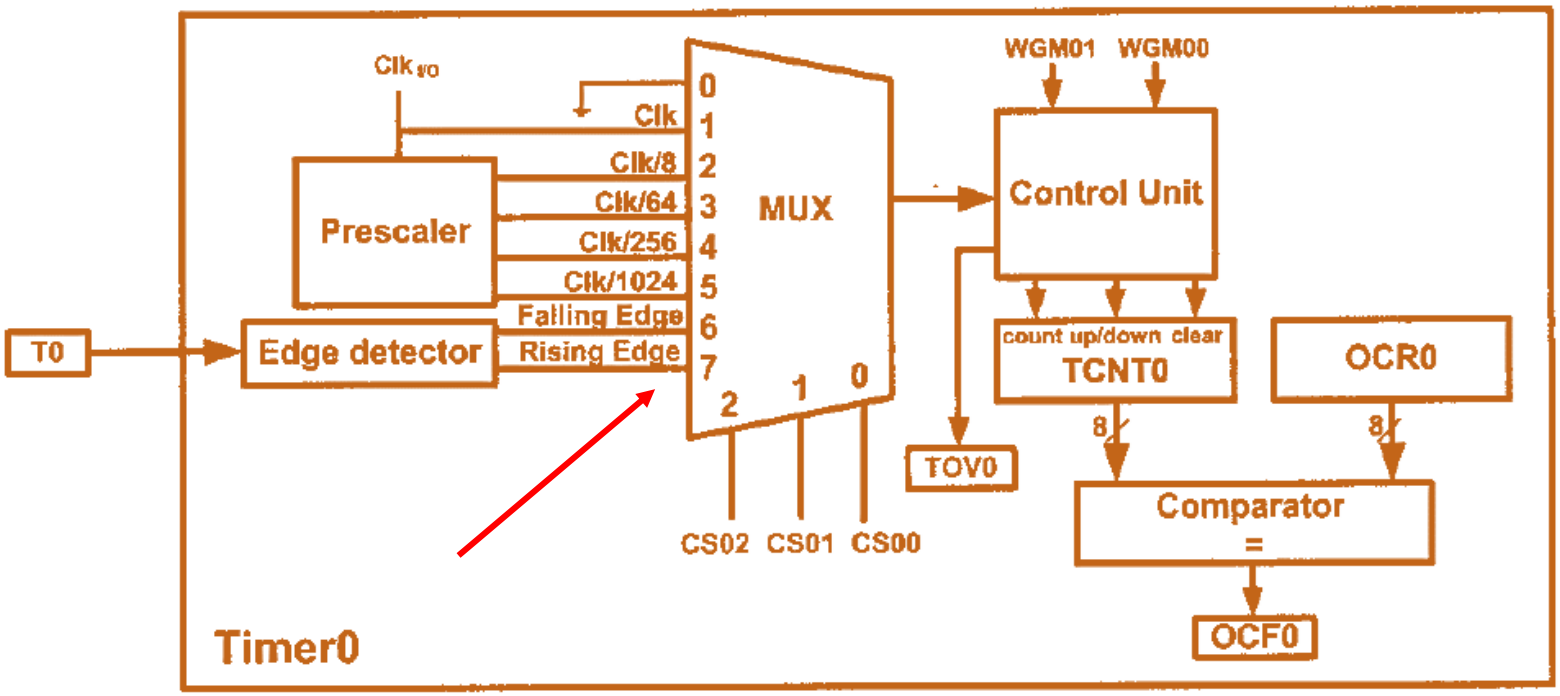
برای اینکه بیایم سیستم این counter رو تنظیم بکنیم تا بیاد و رویدادهای خارجی رو شمارش بکنه برای ما: نیاز داریم که اون رجیستر کنترل رو برای تایمرهامون به گونه ای تنظیم بکنیم که مبنع کلاک رو از اون پالس هایی که از اون رویداد خارجی میاد تحریک بکنه و با پالس هایی که از سمت اون رویداد میاد شمارش بکنه
نکته: توی تایمر صفر و یک این امکان رو ما داریم
می تونستیم منبع کلاک رو با کلاک سلکتور انتخاب بکنیم
توی تایمر صفر اگر مقدار سه بیت کلاک سلکتور ما ینی CS ما 6 یا 7 باشه این میاد پالسی که از اون منبع خارجی می اومد رو به کانتر ما وصل می کرد و کانتر با این پالس شمارش میکرد:
توی حالت 6 ما از لبه پایین رونده اون پالس استفاده میکنیم و اگر 7 باشه لبه بالارونده اون رو به عنوان تحریک کننده شمارنده انتخاب میکنه
برای اینکه بیایم و اون پالس رویداد خارجی رو به سیستم شمارنده توی تایمر صفر متصل بکنیم این رو از طریق پایه صفر پورت B انجام میده ینی T0

همین کارو می تونیم برای تایمر یک هم داشته باشیم: برای تایمر یک ما CS ها که بیت های شماره 10 تا 12 بود رو باید روی 6 یا 7 تنظیم بکنیم که اگر روی 6 تنظیم بشه به لبه پایین رونده حساسه و اگر روی 7 تنظیم بشه به لبه بالارونده حساسه
و پایه PB1 ینی T1 می تونه اون محلی باشه که ما پالس های رویداد خارجی رو متصل میکنیم تا به تایمر یک متصل بشه

Example

Find the value for TCCR0 if we want to program Timer0 as a Normal mode counter. Use an external clock for the clock source and increment on the positive edge.

TCCR0 = 0000 0111 Normal, external clock source, no prescaler



مثال:
می خوایم تایمر صفر رو به گونه برنامه ریزی بکنیم که توی مد نرمال به عنوان کانتر عمل بکنه:
چون می خوایم از کلاک خارجی استفاده بکنیم و یک رویداد رو شمارش بکنیم و اون هم توی لبه بالارونده، میایم مقدار کلاک سلکتور هامون رو ینی سه بیت رو با یک پر می کنیم این باعث میشه که ما بتونیم اون رویداد خارجی به شمارندمون متصل بکنیم و با لبه بالارونده اون کانترمون افزایش پیدا بکنه

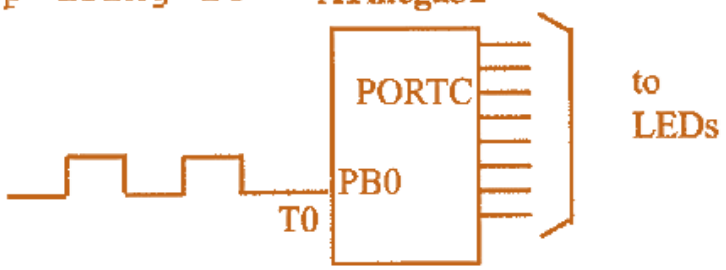
توی شکل:
اون سیگنالی که از خارج میاد به پایه T0 متصل میکنیم

Example

Assuming that a 1 Hz clock pulse is fed into pin T0 (PB0), write a program for Counter0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.

```
.INCLUDE "M32DEF.INC"
      CBI   DDRB,0          ;make T0 (PB0) input
      LDI   R20,0xFF
      OUT   DDRC,R20        ;make PORTC output
      LDI   R20,0x06
      OUT   TCCR0,R20       ;counter, falling edge
AGAIN:
      IN    R20,TCNT0
      OUT   PORTC,R20       ;PORTC = TCNT0
      IN    R16,TIFR
      SBRS  R16,TOV0        ;monitor TOV0 flag
      RJMP  AGAIN           ;keep doing if Timer0 flag is low
      LDI   R16,1<<TOV0
      OUT   TIFR, R16       ;clear TOV0 flag
      RJMP  AGAIN           ;keep doing it
```

PORTC is connected to 8 LEDs and input T0 (PB0) to 1 Hz pulse.



مثال:
یک پالسی داریم که با یک هرتز میاد و متصل میشه به پین T0 ینی پورت B پین شماره صفر و میخوایم یک برنامه ای بنویسیم که از تایمر به عنوان کانتر توی مد نرمال استفاده میکنه و بیاد تعداد این پالس هارو برای ما شمارش بکنه واین حساس به لبه پایین رونده باشه و در نهایت میزان تعداد این پالس هارو روی پورت C برای ما نمایش بده

جواب:
کافیه که پین صفر پورت B ینی T0 رو به عنوان ورودی تنظیم بکنیم
و پورت C به عنوان خروجی
و تایمر صفر را به گونه تنظیم بکنیم که در مد نرمال بیاد و از لبه پایین رونده که یک سیگنال خروجی که به T0 متصل است استفاده بکنیم و شمارش رو انجام بده و کافیه که مقدار TCCR0 رو با 6 پر بکنیم

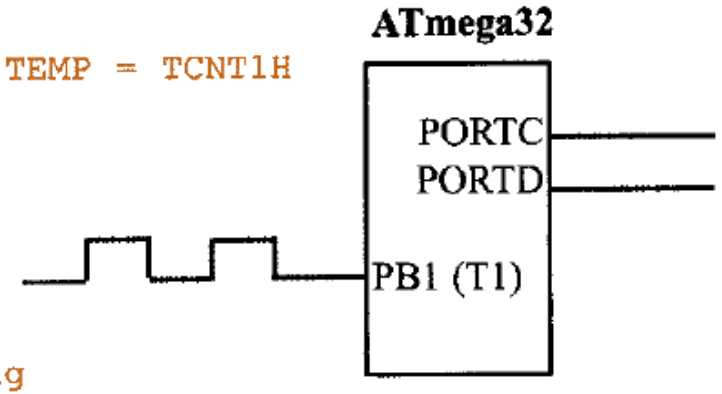
Example

Assuming that clock pulses are fed into pin T1 (PB1), write a program for Counter1 in Normal mode to count the pulses on falling edge and display the state of the TCNT1 count on PORTC and PORTD.

```
.INCLUDE "M32DEF.INC"

      CBI   DDRB,1          ;make T1 (PB1) input
      LDI   R20,0xFF
      OUT   DDRC,R20        ;make PORTC output
      OUT   DDRD,R20        ;make PORTD output
      LDI   R20,0x0
      OUT   TCCR1A,R20
      LDI   R20,0x06
      OUT   TCCR1B,R20      ;counter, falling edge

AGAIN:
      IN    R20,TCNT1L      ;R20 = TCNT1L, TEMP = TCNT1H
      OUT   PORTC,R20       ;PORTC = TCNT0
      IN    R20,TCNT1H      ;R20 = TEMP
      OUT   PORTD,R20       ;PORTD = TCNT0
      IN    R16,TIFR
      SBRS  R16,TOV1
      RJMP  AGAIN           ;keep doing it
      LDI   R16,1<<TOV1     ;clear TOV1 flag
      OUT   TIFR, R16
      RJMP  AGAIN           ;keep doing it
```



مثال:
برای تایمر یک
یک پالس کلاکی به پایه T1 متصل است و میخوایم یک برنامه ای بنویسیم که توی مد نرمال از تایمر صفر استفاده بکنه و بیاد تعداد این رویدادها رو بشماره و روی پورت C , D نمایش بده

جواب: چون تایمر یک 16 بیتی هستش ما برای نمایش این تعداد پالس ها به دو پورت نیاز داریم که اینجا از پورت C , D استفاده شده

Notice!

might think monitoring the TOV and OCR flags is a waste of the microcontroller's time. You are right. There is a solution to this: the use of interrupts. Using interrupts enables us to do other things with the microcontroller. When a timer Interrupt flag such as TOV0 is raised it will inform us. This important and powerful feature of the AVR is discussed next .

نکته:
وقتی داریم از تایمر استفاده میکنیم ما مرتب میایم و پرچم رو چک میکنیم ینی عملا میایم و میکروکنترلر خودمون رو نگه میداریم
که این مشکل است و راه حل ان استفاده از وقفه ها هستش

پایان

موفق و پیروز باشید