Dette dokument gennemgår opsætning af Tæller / Timer udløste Interrupts.

Links til afsnit i dokumentet:

Arduino Timer Interrupts.

Hvordan får man Arduino til fx at udføre et interrupt og køre en programstump nøjagtig 100 gange i sekundet. ??

Det program, der normalt kører på en controller, er en række af sekventielle instruktioner udført efter hinanden i et loop. Den tid, et loop tager, er derfor afhængig af hvor stort programmet er.

Derfor er man nødt til at anvende Interrupts, hvis man skal være sikker på timingen, altså at kodestumper udføres efter en ganske bestemt tid !!.

Et interrupt er et **event** – eller **hændelse**, - udløst enten internt fra en tæller / counter, - eller eksternt fra en pin, der ændrer status. Dette er beskrevet i et andet dokument !!

En Interrupt-rutine er en programstump, der udløses af en hændelse og kører selvstændigt. Dvs. ikke i et loop, men som en kort selvstændig programdel, med et start- og et slut-punkt.

Interrupt'et udløses fx af et key-tryk, - eller af en forløbet tid, - timer-overflow, tæller-overflow osv.

Interrupts udløses – og udføres et vilkårligt sted i et kørende loop-program.

Det program, der køres når interruptet opstår, kaldes en **interrupt service routine, forkortet: ISR.**

Altså: Når et Interrupt opstår, afbrydes det kørende program øjeblikkeligt, og interrupt service rutinen, dvs. en subrutine, kaldes.

Det vi ikke ser i C-kode, er, at der automatisk bliver gemt en tilbagehopsadresse, ligesom vitale registres indhold gemmes i RAM, i et område kaldet "stakken". Det sker i baggrunden, og det er compileren, der genererer kode til det.

Når ISR'en er færdig, hentes de gemte værdier, som processoren var ved at arbejde med da interruptet opstod, og loop() fortsætter som om intet er hændt.

Typisk er der gjort brug af interrupts allerede i de tidligere opgaver, der er lavet. De er bare "gemt"!

Det er fordi, Arduino´s underlæggende compiler automatisk indbygger mulighed for at bruge funktioner som, millis() , micros() og delayMicroseconds(). Disse funktioner gør brug af interrupts og bruger timere!!

PWM-funktionen analogWrite() bruger timere, ligesom tone(), noTone() og Servo library gør.

Og seriel kommunikation bruger interrupts!!! Det sker, når der er modtaget en hel byte, og den skal placeres i modtagebufferen. Og ligeledes når der skal sendes data fra sendebufferen !!

Regler og tips for en Interrupt-rutine.

Når der bruges interrupts i et program, er der mange faldgruber og begrænsninger der skal tages højde for.

Overholder man reglerne / rådene, kan man spare sig for mange unødvendige og ”hard to find” debug problemer.

Her gennemgås nogle:

En interruptsubrutine (eller Funktion, ISR) kan udføres mange gange i sekundet. Fx hvis det er et timerudløst interrupt. Derfor er det vigtigt, at selve rutinen ikke tager ret lang tid.

Dvs. der bør ikke være tunge regneoperationer i en ISR. En ISR bør max tage nogle få microsekunder. Hvis en Interrupt-funktion kaldes igen før den forrige er færdig, vil der jo opstå problemer.

ISR’s skal altid holdes så korte som muligt.

En ISR kan kommunikere med loop() ved at sætte flag, dvs. en bit, en boolean. I loop() reageres der så på flaget, og en ønsket funktion udføres, - og flaget lægges ned igen !

Men husk at flaget skal være erklæret som volatile.

Fx: `volatile bool MoveMotor = false;`

I processoren kan der kun udføres 1 interruptfunktion ad gangen. Der er lukket for, at én interrupt kan blive interruptet af en anden interrupt. Det kan man sagtens i andre processorer !!

Brug ikke ”tids-funktioner” i en ISR:

Som en tommelfingerregel bør der ikke bruges ”tidsfunktioner” i en interruptfunktion.

Arduino´s underlæggende compiler indbygger automatisk mulighed for at bruge funktioner som, millis(),og micros(). Disse funktioner gør i sig selv brug af interrupts og bruger Timer0 !!

- Millis(): bruger nogle andre interruptfunktioner for at tælle, og de vil ikke køre i en anden ISR.
- Delay(): virker ikke i en ISR. Den bruger selv millis().
- Micros(): vil virke i starten af et interrupt, men vil efter 1 til 2 millisekunder blive unøjagtig. Bruger også timer0.
- delayMicroseconds(): bruger ikke en tæller, så den vil virke, men brug den helst ikke.

- Serial kommunikation: Brug heller ikke Serial i et interrupt. Denne funktion bruger i sig selv interrupts! Dvs. Modtaget eller sendt data kan gå tabt !

Volatile – globale variable.

Hvis man bruger variable, der bruges både i en ISR-funktion og i den normale loop()-kode, skal de altid defineres som volatile. Dette fortæller compileren, at sådanne variables indhold kan ændres når som helst og hvor som helst i et program.

Derfor skal man " fortælle compileren ", at den – fx for at tidsoptimere koden, - ikke " bare " placerer en kopi af disse variable i processorens indre registre.

Det svarer vel lidt til at en PC henter et dokument ned i en cache.

Eks:

```
int pin = 13;
volatile int a = 3;
volatile uint8_t Count = 0;
volatile bool state = LOW;  // eller 0 !
```

Seriel kommunikation:

Bruges seriel kommunikation i en ISR, kan serielt modtagne data blive tabt.

Serial.print() kan bruges i en ISR til debugging men kun til debugging. Det er bedre at sætte et flag (husk volatile) , og så tjekke flaget i LOOP(), og ved sat flag, seriel printes der, og flaget lægges ned igen

Seriel kommunikation bruger selv interrupts!! Det sker, når der er modtaget en hel byte, og den skal flyttes over i modtagebufferen. Og ligeledes i sendebufferen når der er afsendt en hel byte !!

Interruptparametre

En interruptfunktion kan ikke medtage parametre, og kan heller ikke returnere parametre !!

Pin Input / Output i en ISR.

digitalRead() eller digitalWrite() kan godt bruges i en ISR, men bør ikke overgøres

Altså:

- Hold interrupt-service-rutiner så korte som muligt.
- Brug aldrig delay() i en interruptrutine, aldrig !!
- Brug aldrig Serial.print eller Serial.write
- Erklær variable der deles med kode udenfor ISR-en ”volatile”, dvs. globale. (før setup)
- Prøv ikke at enable/disable interrups

Opsætning af et interrupt

Før man kan benytte interrupts, skal der noget opsætning til. Ved at sætte forskellige bits i forskellige **special function registre** kan man opsætte processoren til at interrupte på forskellige events.

Interrupts skal altså enables og den tilhørende interrupt-maske skal enables, ligesom det skal beskrives (programmeres), hvad processoren skal udføre, under et interrupt.

Interrupt Service Routine (ISR)

Interrupts can generally be enabled / disabled with the function interrupts() / noInterrupts(). By default in the Arduino firmware interrupts are enabled.

Interrupt masks are enabled / disabled by setting / clearing bits in the Interrupt mask register (TIMSKx).

When an interrupt occurs, a flag in the interrupt flag register (TIFRx) is been set. This interrupt flag will be automatically cleared when entering the ISR or by manually clearing the bit in the interrupt flag register.

Timerudløste interrupts

Alle Microcontrollere har flere forskellige hardware-funktioner indbygget.

A/D-konvertere, UARTS, Tællere mmm.

Alle tællerne er bygget op af transistorer, der tilsammen udgør gates. De er så sat sammen til Flip Flops.

Og vha. FF's kan man bygge tællere.

Atmegaen ATMEGA328P har 3 indbyggede tællere: timer0: 8 bit, timer1: 16 bit & timer2: 8 bit.

Tællerne kan fx bruges til at tælle pulser på en inputpin, men også til timingformål. Det kan foregå ved at man leder krystallets kendte – og konstante 16 MHz frekvens - til en tæller, for så på den måde at udmåle en tid ved at tælle et bestemt antal pulser.

Hver gang der er gået en bestemt tid, skal der så ske noget, fx at få opdateret et stopur hver 100-del sekund.

Eller man kan anvende en uC som frekvensgenerator. Eller fx som cykelcomputer, hvor der skal tælles pulser fra en magnetføler der føler dæk-omdrejninger - i en bestemt tid.

Dette kan opnås ved at få et specielt programstump, en "interrupt-service-rutine" til at køre hver gang der er gået en bestemt tid.

Det hele styres ved at sætte nogle bits i nogle SFR's – dvs. Registre, eller RAM-adresser, som igen svarer til en variabel i en bestemt RAM-adresse

Timerne kan indstilles til udløse et interrupt på 2 forskellige måder:

Overflow / Compare Match

Interrupt kan udløses når timeren har talt op til overflow fra en indstillet startværdi

Compare match

Interrupt kan udløses når der er talt fra 0000h og op til samme værdi som er gemt i et andet register.

Oversigt:

Counter Overflow Interrupt	**Counter Compare Match Interrupt**
Tællerens startværdi indstilles, og når timeren giver **overflow**, kan der udløses et interrupt. Efter et interrupt skal timerens startværdi igen sættes ind i tælleren.	Der skal indsættes et tal, - en værdi - i et Compare-register. Tælleren starter på 00 00h og der kan udløses et interrupt, når dens tællerværdi når op til samme

	værdi som er lagt ind i Compare-registeret. Ved match resettes timeren automatisk.

Normal Mode:

In Normal Mode the counter will count until it reaches the TOP value. When the TCNT1 register passes the TOP value (0xFFFF or 65535) it simply overflows (or overruns) back to 0, at the same time the TOV1 flag is set.

CTC Mode:

In CTC (Clear Timer on Compare) mode the counter will count until it hits the value specified in the OCR1 register. When the TCNT1 passes the TOP value (Specified by the OCR1) it resets to 0 and at the same time sets the TOV1 flag.

An interrupt can be configured to trigger when the TOV1 flag is set.

Indbyggede tællere i Arduino Uno

Arduino Uno bruger uC-en Atmega328P, der indeholder 3 timere, timer0, timer1 og timer2.

Timer 0: 8 bit
Timer 1: 16 bit.
Timer 2: 8 bit

Vi vil her kun arbejde med Tæller1, der er en 16 bit tæller.

Tællerne kan bruges til flere forskellige ting. De kan arbejde i flere forskellige MODES, som fremgår af skemaet her.

De fleste modes drejer sig om PWM-funktioner, - men her ser vi på Mode 4, CTC. (Clear Timer on Comparematch) Men Overflow virker også !!

De forskellige Modes vælges ved at sætte nogle bit i nogle kontrolregistre, dvs. nogle SFR.

Mode 4 vælges ved at sætte bit WGM12 i register TCCR1A / TCCR1B

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV 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	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	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	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Herom senere !!

Prescaler. / Frekvensdeler.

På Arduinoen sidder der et 16 MHz krystal. Hvis disse 16 MHz ledes til en 16-bit tæller vil den give overflow i løbet af ca. 4 mS.

$$Overflow = \frac{65535}{16.000.000} = 0{,}0041 \sim 4\ mS.$$

Men heldigvis er der indbygget mulighed for at vælge en neddelingsfaktor for clockfrekvensen til tælleren. Det kaldes en "**prescaler**".

Hver tæller har sin egen prescaler.

Prescaleren kan indstilles til at dele krystallets frekvens med nogle faste værdier: 1, 8, 64, 256 eller 1024.

(Eller man kan også indstille, at eksterne pulser ledes ind til tælleren fra en pin.)

Indstillingen af Prescaleren styres ved at sætte nogle bit i to RAM-adresser, såkaldte SFR-registre.

Timer1 styres af 2 registre, TCCR1A og TCCR1B.

Navnene kommer af: **T**imer**C**ounter**C**ontrol-**R**egister, timer 1, register A eller B.

Og styrebittene i registrene hedder:

CSn0, CSn1 og CSn2, hvor tallet n er timerens nummer.

CS står for **C**lock **S**elect bit.

Kilde: http://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture7.pdf

Udregning af Prescaler-værdi:

For at få udløst et interrupt med en korrekt frekvens, skal man udregne hvor meget man skal neddele krystallets frekvens, - og man skal vælge hvilken værdi:

Der skal tælles op til hvis metoden **Compare Match** vælges,
Eller fra hvilken værdi, der skal startes ved hvis metoden **Overflow** vælges.

Compare Match:

Udregningen sker som flg:

Mulige prescaler-værdier er: 1, 8, 64, 256, 1024.

$$interrupt\ frekvens = \frac{16\ MHz}{Prescaler\ \cdot ((CompareMatchReg) + 1)}$$

(+ 1 fordi det tager 1 clock cycle at resette tælleren.

Her er ligningen løst for Compare-Match register-værdi:

$$CompareMatchRegister = \left(\frac{16.000.000}{Prescaler \cdot interrupt\ frekv.}\right) - 1$$

CompareMatch registeret er på 16 bit, og værdien kan derfor ikke være større end 65535.

Overflow:

Her ses ligningen for timer1:

$$Timer1load = 65535 - \frac{16000000}{Prescaler\ \cdot\ interruptfrekvens}$$

Værdien skal jo være mindre end 65.536 (for Tæller1), for at det kan lade sig gøre:

Hjælp til udregning af prescaler:

Der kan findes hjælp på en række hjemmesider til at vælge den rette Prescaler til et bestemt formål.

Fx http://www.8bit-era.cz/arduino-timer-interrupts-calculator.html

Eller: https://www.arduinoslovakia.eu/application/timer-calculator
(Siderne genererer også kode !!)

Mulige Prescalerværdier vs. Mulig Interruptfrekvens

Her er vist en oversigt for de forskellige prescaler-værdier for Tæller1 og mulig interruptfrekvens.

Prescaler Værdi	Mulig interrupt-Frekvens

1	16 MHz til 244 Hz
8	2 MHz til 30,5 Hz
64	250 KHz til 3,8 Hz
256	62,5 KHz til 0,95 Hz
1024	15,625 KHz til 0,24 Hz

Og her en liste over udvalgte interruptfrekvenser, prescalerværdier og timer Loadværdier.

Skema med Interruptfrekvens, Prescalervalg og Timerpreload

Timer Overflow	Frekvens Hz	Prescaler:	Timer Preload
	1	256	`TCNT1 = 3036;`
	2	256	`TCNT1 = 34286;`
	50	256	`TCNT1 = 64286;`
	100	256	`TCNT1 = 64886;`

Compare Match			Comp. reg. Preload
	0,1	1024	`OCR1A = 1562;`
	1	1024	`OCR1A = 15624;`
	2	256	`OCR1A = 31250;`
	10	64	`OCR1A = 24999;`
	50	8	`OCR1A = 39999;`
	100	8	`OCR1A = 19999;`
	100	256	`OCR1A = 624;`
	1000	1	`OCR1A = 15999;`

Indstilling af Prescaler:

Hver timer styres af og bruger nogle SFR-registre, fx Timer/Counter Control Registre:

De bruges både til timer opsætning,- men også til selve tællerne.

Det er ret komplekst, også fordi timerne kan bruges til mange andre ting end blot at tælle.

Først en oversigt over registre, - og derefter en graf:

TCCR1A og **TCCR1B**

Prescaleren konfigureres i disse 2 registre.

Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Prescaleren kan vælges til:
1, 8, 64, 256, 1024,
- eller til ekstern puls.

Prescaleren styres af bit0, bit1 og bit2 i register TCCR1B.

Bit	7	6	5	4	3	2	1	0	
(0x81)	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	

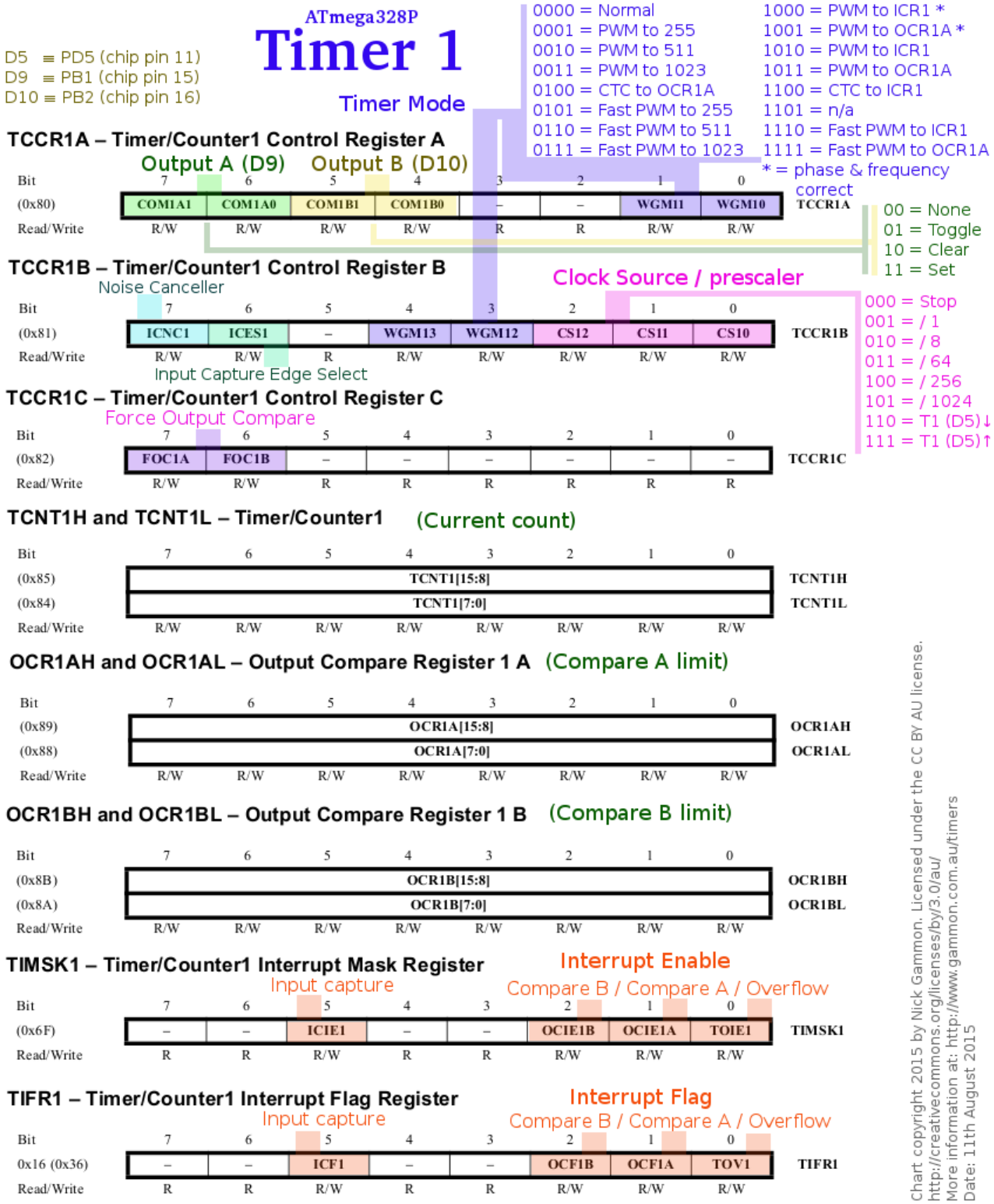
CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

TCNTx *Timer/Counter Register.*

Indeholder den aktuelle tæller-værdi, 16 bit.
TCNT1H og **TCNT1L**

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

Her ses en grafik over de relevante SFR's og hvilke bit, der skal sættes for ønsket funktion.:

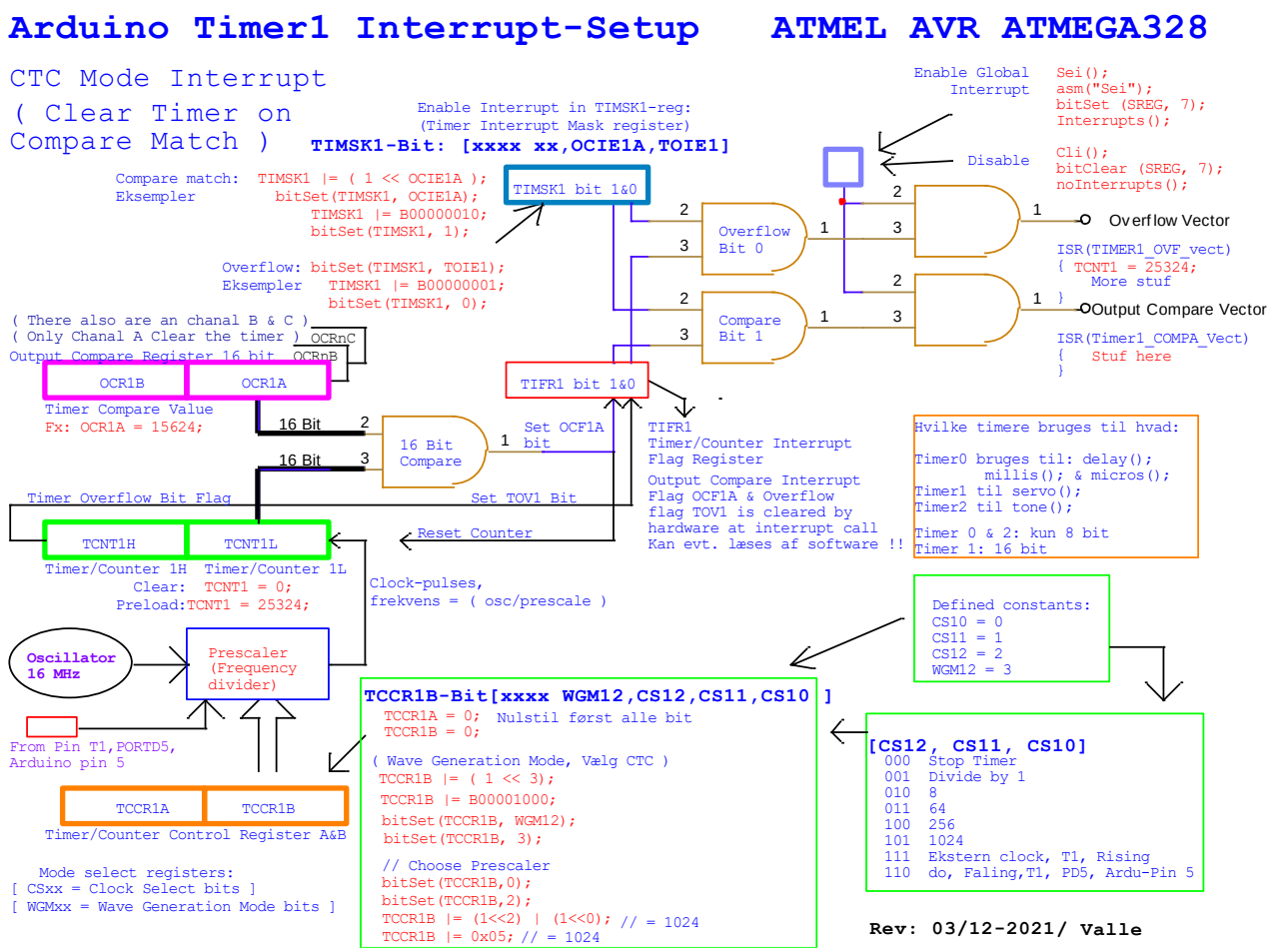


Fra: https://arduino.stackexchange.com/questions/16698/arduino-constant-clock-output

https://gammon.com.au/timers

Oversigt:

Jeg har her forsøgt at lave et samlet diagram over tæller-interrupt-strukturen for Tæller1:



Tegningen viser de forskellige SFR-registre involveret i opsætningen, og der er vist forskellige eksempler på instruktioner der kan bruges til at indstille bittene i registrene:

ISR-Eksempler

Når der så udløses et interrupt, skal der køres noget kode. Det skal placeres i en Interrupt Service Routine. (ISR).

Her er det vigtigt, at det udformes på den korrekte måde.

Først et eksempel på Compare-Match-udløst ISR:

```
ISR(TIMER1_COMPA_vect)  // En ”Vektor” peger i hukommelsen.
{      // Timer is automatic reset
// Do stuf, fx:
  digitalWrite(13, !digitalRead(13)); // do something
}
```

Og her overflow-udløst ISR:.

```
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 34286;      // reload timer, nødvendig
// Do stuf, fx:
  digitalWrite(13, !digitalRead(13)); //  Toggle
}
```

Her er en oversigt over, hvad der skal opsættes:

Hvad skal gøres?	Kode-eksempel (vælg det relevante)
Disable global interrupt:	`cli();  // = noInterrupts();`
Nulstil tæller:	`TCNT1  = 0;  // Virker på alle 16 bit.`
Indstil Compare værdi:	`OCR1A = 15624; // OCR1A = 0x3D08; på hex form;`

Nulstil først alle bit:	`TCCR1A = 0;  // Nulstil først alle bit i`
	`TCCR1B = 0;  // kontrolregistrene.`

Vælg Counter/timer Mode:	`bitSet(TCCR1B, 3); // Wave Form Generation`
	`// Bit "WGM12" = 3`

Indstil Prescalerværdi:	`bitSet(TCCR1B, 0);  // = 00000001 = Bit 0`
	`bitSet(TCCR1B, 1);  // = 00000010 = Bit 1`
	`bitSet(TCCR1B, 2);  // = 00000100 = Bit 2`

Enable Timer1 interrupt for enten Compare Match: eller Overflow:	`bitSet(TIMSK1, 1); //Enable CompareInt.`
	`bitSet(TIMSK1, 0);  //Enable Overfl. Int`

Enabel Global interrupt:	`sei();  //= interrupts(); //`

Se evt: [1]

Funktion til at opsætte interrupt

[1] For en oversigt se side 40 i: definerede værdier compileren kender:
Se flere eksempler på bitmanipulation i dokumentet: Port- og bitmanipulation

Her er lavet et eksempel på en ” generel ” funktion, til interrupt-opsætning. Den kan kopieres og indsættes i en Arduino-sketch som en funktion, og rettes til. Fx i en separat tab. Den skal så blot kaldes, fx i setup().

Ideen er, at man blot sletter eller ud-kommenterer de linjer, der ikke skal bruges.

```
void interruptSetup() {          //  Choose correct values

  cli();                         // = noInterrupts(); Disable all Interrupts

  TCCR1A = 0;                    // Reset Timer Counter Control Registers
  TCCR1B = 0;                    // Same for B

  bitSet(TCCR1B, 3);             // turn on CTC mode:

                                 // Set Presaler:
                                 //       1: Set bit 0     (CS10)
                                 //       8: Set bit 1     (CS11)
                                 //      64: Set bit 0 & 1 (CS10 & CS11)
                                 //     256: Set bit 2     (CS12)
                                 //    1024: Set bit 0 & 2 (CS10 & CS12)
                                 // Extern pulses: Set bit 0, 1 & 2

  bitSet(TCCR1B, 0);             // = 00000001 = Bit 0
  bitSet(TCCR1B, 1);             // = 00000010 = Bit 1
  bitSet(TCCR1B, 2);             // = 00000100 = Bit 2

                                 // Chose Compare Mode
  bitSet(TIMSK1, 1);             //Enable CompareInterrupt.
  OCR1A = 15624;                 // Set Compare Value

 // eller
                                 // Chose Overflow Mode
  bitSet(TIMSK1, 0);             //Enable Overfl. Int
  TCNT1  = 34286;                // Preset Counter, alle 16 bit.

  sei();                         //= interrupts();  Enable all Interrupts
} // end of timeropsætning

// ************************
// De tilhørende ISR's. Slet den, der ikke skal bruges
//
ISR(TIMER1_COMPA_vect)  // Compare-Match Vector
{      // Timer is automatic reset
// Do Compare Match Interrupt-stuf:
}  // End of Compare-vector
// ************************
ISR(TIMER1_OVF_vect)  // Overflow Vector
{
  TCNT1 = 34286;      // reload startværdi af timer. Er nødvendigt
// Do Overflow Interrupt-stuf
}  // End of Overflow-vector
// Valle Thorø
```

BONUS-INFO

De resterende sider er eksempler og Bonusmateriale samlet derudefra 😊

Eksempler på timer Interrupt-rutiner.

” vect ” står for vektor, der skal forstås som en pil, der peger på det sted i program-hukommelsen, hvor kodestumpen i interrupt-programmet er placeret.

```
ISR(TIMER1_COMPA_vect)            // En ”Vektor” peger på programstump i
                                  // hukommelsen.
{
                                  // Timer is automatic reset
  digitalWrite(ledPin, HIGH);     // do something
}
```

Langtids-timer

Eks: på hvordan man kan få noget til at ske med lang tids mellemrum. Der er opsat et interrupt på 1 sekund, men selve programmet i interruptrutinen udføres her kun hver 10. sekund.

```
ISR(TIMER1_COMPA_vect)       //  Compare match
{
    seconds++;
    if (seconds == 10)
    {
        seconds = 0;
        readMySensor();      // Funktionen kaldes kun hver 10’ende sekund
    }
}
```

Eksempel på et ”Overflow-udløst” interrupt-rutine.

```
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 34286;                  // reload timer, nødvendig

  digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // Bitwise XOR = Toggle

}
```

Udregning af Prescaler, 10 Hz:

```
Interrupt vælges i dette eksempel til 10 gange pr. sekund, dvs. hver 1/10
sekund, eller til 10 Hz.

Timer 1 giver overflow ved tællerværdien FFFFh + 1 = 65536d.

Frekvensen på krystallet er 16 MHz. Dvs. på 0,1 sek. kommer 1.600.000 pulser.

Det er nødvendigt med en frekvensdeler, en prescaler, fordi der kommer for
mange pulser på 0,1 sekund.

Der skal mindst deles med  ( 1.600.000 / 65.536 ) =  ca. 24,8

Prescaleren kan indstilles til 1, 8, 64, 256, 1024.

Der vælges fx 64

Derfor kommer på 1/10 sekund 1.600.000/64 = 25.000 pulser.

Vælges overflow mode, må tællerens startværdi indstilles til:
 ( FFFF + 1 ) - 25.000 = 40.536

Vælges Output Compare, tælles fra 0000h og til 25.000. Compare registeret skal
derfor indstilles til 25.000 decimal.
```

Clock select and timer frequency

Different clock sources can be selected for each timer independently. To calculate the timer frequency (for example 2Hz using timer1) you will need:

CPU frequency 16Mhz for Arduino

maximum timer counter value (256 for 8bit, 65536 for 16bit timer)
Divide CPU frequency through the choosen prescaler (16000000 / 256 = 62500)
Divide result through the desired frequency (62500 / 2Hz = 31250)

Verify the result against the maximum timer counter value (31250 < 65536 success) if fail, choose bigger prescaler.

Kilder:

https://roboticsbackend.com/arduino-interrupts/

http://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture7.pdf
http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Die_Timer_und_Z%C3%A4hler_des_AVR
http://www.eng.utah.edu/~cs5789/2010/slides/Interruptsx2.pdf

Se datablad: https://www.microchip.com/en-us/product/ATmega328PB

Timer Overflow eksempler:

Timer overflow means the timer has reached is limit value. When a timer overflow interrupt occurs, the timer overflow bit TOVx will be set in the interrupt flag register TIFRx. When the timer overflow interrupt enable bit TOIEx in the interrupt mask register TIMSKx is set, the timer overflow interrupt service routine ISR(TIMERx_OVF_vect) will be called.

I denne mode timeren presettes til en startværdi, - og overflow udløser så et interrupt.

Overflow-eksempel:

```
/*
Eksempel på Interrupt ved timer overflow.
Testet!!

Valle / 8/11-2013

*/

#define ledPin 13
int timer1_startvalue;
int sekund = 0;
int minut = 0;
volatile boolean flag = 0;
//boolean flag = 0;

void setup()
{
    pinMode(ledPin, OUTPUT);

    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for Leonardo only
    }


    // initialize timer1
    noInterrupts();           // disable all interrupts
    TCCR1A = 0;
    TCCR1B = 0;
```

```
    // Set timer1_startvalue to the correct value for our interrupt interval
    //timer1_startvalue = 64886;   // preload timer 65536-16MHz/256/100Hz
    //timer1_startvalue = 64286;   // preload timer 65536-16MHz/256/50Hz
    //timer1_startvalue = 34286;   // preload timer 65536-16MHz/256/2Hz
    timer1_startvalue = 3036;      // preload timer 65536-16MHz/256/1Hz

    TCNT1 = timer1_startvalue;   // preload timer
    bitSet(TCCR1B, CS12);        // 256 prescaler
    bitSet(TIMSK1, TOIE1);       // enable timer1 overflow interrupt
    interrupts();                // enable all interrupts
}

ISR(TIMER1_OVF_vect)        // interrupt service routine
{
    TCNT1 = timer1_startvalue;   // gen-load timer1
    digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // Exor, dvs. Toggle.

    sekund++;
    flag = HIGH;
    if (sekund > 59) {
        sekund = 0;
        minut++;

    }
}
void loop()
{
    while(flag==LOW) {  // Wait til change !!
    }
    Serial.print(minut);
    Serial.print(':');
    if(sekund<10) Serial.print('0');
    Serial.println(sekund);
    flag=0;
    //  delay(1000);
    // your program here...
}


// Se:  http://www.hobbytronics.co.uk/arduino-timer-interrupts
```

Output Compare Match eksempler:

When a output compare match interrupt occurs, the OCFxy flag will be set in the interrupt flag register TIFRx . When the output compare interrupt enable bit OCIExy in the interrupt mask register TIMSKx is set, the output compare match interrupt service ISR(TIMERx_COMPy_vect) routine will be called.

Timerne kan indstilles til at udløse et interrupt hver gang, en timer når op til samme værdi, som er gemt i et timer-match register, - eller hvis de når deres max count-værdi, og giver overflow.

Når en timer når et match,- bliver den resat på det næste klockpuls – og fortsætter så opad igen fra 0 til næste match.

Ved at vælge Compare match værdien, - og vælge klock-frekvensen (16 MHz) sammen med en frekvensdeler, en prescaler, kan man kontrollere interruptfrekvensen.

Timeren sætter ved overflow et overflow-bit som evt. kan tjekkes manuelt af programmet.

ISR-en (Interrupt Service Routine resetter overflowbit.

2 Hz Compare Match eksempel

```
/* timer and interrupts
   Timer1 compare match interrupt example
   more infos: http://www.letmakerobots.com/node/28278
   created by RobotFreak
*/

#define ledPin 13

void setup()
{
  pinMode(ledPin, OUTPUT);

  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;

  OCR1A = 31250;            // compare match register 16MHz/256/2Hz
  bitSet(TCCR1B, WGM12);   // CTC mode
  bitSet(TCCR1B, CS12);    // 256 prescaler
  bitSet(TIMSK1, OCIE1A);  // enable timer compare interrupt
  interrupts();             // enable all interrupts
}

ISR(TIMER1_COMPA_vect)          // timer compare interrupt service routine
{
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);   // toggle LED pin
}

void loop()
{
  // your program here...
}
```

```
  //  Denne interrupt kaldes ved 1kHz
  //   http://petemills.blogspot.dk/2011/05/real-time-clock-part-1.html

ISR(TIMER1_COMPA_vect)
{
 static uint16_t milliseconds = 0;  // mS value for timekeeping 1000mS/1S
 static uint16_t clock_cal_counter = 0; // counting up the milliseconds to MS_ADJ
 const uint16_t MS_ADJ = 35088;   // F_CPU / (F_CPU * PPM_ERROR)
 const uint16_t MS_IN_SEC = 1000;  // 1000mS/1S

 milliseconds++;
 clock_cal_counter++;


 if( milliseconds >= MS_IN_SEC )
 {
  milliseconds = 0;
  ss++;    // increment seconds
  toggle_led();   // toggle led

  if( ss > 59 )
  {
   mm++;   // increment minutes
   ss = 0;   // reset seconds
  }

  if( mm > 59 )
  {
   hh++;   // increment hours
   mm = 0;   // reset minutes
  }

  if( hh > 23 )
  {
   // increment day
   hh = 0;   // reset hours
  }
 }


 // milliseconds must be less than 999 to avoid missing an adjustment.
 // eg if milliseconds were to be 999 and we increment it here to 1000
 // the next ISR call will make it 1001 and reset to zero just as if it
 // would for 1000 and the adjustment would be effectively canceled out.
```

```
 if( ( clock_cal_counter >= MS_ADJ ) && ( milliseconds < MS_IN_SEC - 1 ) )
 {
  milliseconds++;

  // it may be that clock_cal_counter is > than MS_ADJ in which case
  // I want to count the tick towards the next adjustment
  // should always be 1 or 0

  clock_cal_counter = clock_cal_counter - MS_ADJ;
 }

}
```

Timer0:
Timer0 is a 8bit timer.
In the Arduino world timer0 is been used for the timer functions, like delay(), millis() and micros(). If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

The delay(1000) function works on Timer-0's overflow interrupt.

Timer1:
Timer1 is a 16bit timer.
In the Arduino world the Servo library uses timer1 on Arduino Uno.

Timer2:
Timer2 is a 8bit timer like timer0.
In the Arduino world the tone() function uses timer2.

Eksempler på opsætning af timer0, timer1 og timer2:

```
void setup(){

cli();              //stop interrupts

//set timer0 interrupt at 2kHz
TCCR0A = 0;         // set entire TCCR0A register to 0
TCCR0B = 0;         // same for TCCR0B
TCNT0  = 0;         //initialize counter value to 0
OCR0A = 124;        // set compare match register for 2khz
                    // increments
                    // = (16*10^6) / (2000*64) - 1 (must be <256)
  bitSet(TCCR0A, WGM01);    // turn on CTC mode
  bitSet(TCCR0B, CS01);
  bitSet(TCCR0B, CS00); // Set CS01 and CS00 bits for 64 presc.
  bitSet(TIMSK0, OCIE0A);   // enable timer compare interrupt
```

```
//set timer1 interrupt at 1Hz
  TCCR1A = 0;           // set entire TCCR1A register to 0
  TCCR1B = 0;           // same for TCCR1B
  TCNT1  = 0;           //initialize counter value to 0
  OCR1A = 15624;        // set compare match register for 1hz
                        // increments
                        // = (16*10^6) / (1*1024) - 1 (must be <65536)

  bitSet(TCCR1B, WGM12);    // turn on CTC mode
  bitSet(TCCR1B, CS12);
  bitSet(TCCR1B, CS10);     // Set CS10 og CS12 bits for 1024 presc.
  bitSet(TIMSK1, OCIE1A);       // enable timer compare interrupt

//set timer2 interrupt at 8kHz
  TCCR2A = 0;           // set entire TCCR2A register to 0
  TCCR2B = 0;           // same for TCCR2B
  TCNT2  = 0;           //initialize counter value to 0
  OCR2A = 249;          // set compare match register for 8khz
                        // increments
                        // = (16*10^6) / (8000*8) - 1 (must be <256)

  bitSet(TCCR2A, WGM21);    // turn on CTC mode
  bitSet(TCCR2B, CS21);     // Set CS21 bit for 8 prescaler
  bitSet(TIMSK2, OCIE2A);   // enable timer compare interrupt

sei();                  //allow interrupts

}                       //end setup


ISR(TIMER0_COMPA_vect){     //change the 0 to 1 for timer1 and 2 for timer2

//interrupt stuf here

}
```

http://www.instructables.com/id/Arduino-Timer-Interrupts/

Bike Speedometer

In this example I made an arduino bike speedometer. It works by attaching a magnet to the wheel and measuring the amount of time it takes to pass by a magnetic switch mounted on the frame- the time for one complete rotation of the wheel.

I set timer 1 to interrupt every ms (frequency of 1kHz) to measure the magnetic switch. If the magnet is passing by the switch, the signal from the switch is high and the variable "time" gets set to zero. If the magnet is not near the switch "time" gets incremented by 1. This way "time" is actually just a measurement of the amount of time in milliseconds that has passed since the magnet last passed by the magnetic switch. This info is used later in the code to calculate rpm and mph of the bike.

Here's the bit of code that sets up timer1 for 1kHz interrupts

Here's the complete code if you want to take a look:

```
//bike speedometer
//by Amanda Ghassaei 2012
//http://www.instructables.com/id/Arduino-Timer-Interrupts/
//http://www.instructables.com/id/Arduino-Timer-Interrupts/

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
*/

//sample calculations
//tire radius ~ 13.5 inches
//circumference = pi*2*r =~85 inches
//max speed of 35mph =~ 616inches/second
//max rps =~7.25

#define reed A0//pin connected to read switch

//storage variables
float radius = 13.5;// tire radius (in inches)- CHANGE THIS FOR YOUR OWN BIKE

int reedVal;
long time = 0;// time between one full rotation (in ms)
float mph = 0.00;
float circumference;
boolean backlight;

int maxReedCounter = 100;//min time (in ms) of one rotation (for debouncing)
int reedCounter;


void setup(){

  reedCounter = maxReedCounter;
  circumference = 2*3.14*radius;
  pinMode(1,OUTPUT);//tx
  pinMode(2,OUTPUT);//backlight switch
  pinMode(reed,INPUT);//redd switch

  checkBacklight();

  Serial.write(12);//clear

  // TIMER SETUP- the timer interrupt allows preceise timed measurements of the
reed switch
  //for mor info about configuration of arduino timers see
http://arduino.cc/playground/Code/Timer1
  cli();//stop interrupts

  //set timer1 interrupt at 1kHz
```

```
  TCCR1A = 0;// set entire TCCR1A register to 0
  TCCR1B = 0;// same for TCCR1B
  TCNT1  = 0;//initialize counter value to 0;
  // set timer count for 1khz increments
  OCR1A = 1999;// = (16*10^6) / (1000*8) - 1
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS11 bit for 8 prescaler
  TCCR1B |= (1 << CS11);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);

  sei();//allow interrupts
  //END TIMER SETUP

  Serial.begin(9600);
}

void checkBacklight(){
  backlight = digitalRead(2);
  if (backlight){
    Serial.write(17);//turn backlight on
  }
  else{
    Serial.write(18);//turn backlight off
  }
}

ISR(TIMER1_COMPA_vect) {//Interrupt at freq of 1kHz to measure reed switch
  reedVal = digitalRead(reed);//get val of A0
  if (reedVal){//if reed switch is closed
    if (reedCounter == 0){//min time between pulses has passed
      mph = (56.8*float(circumference))/float(time);//calculate miles per hour
      time = 0;//reset timer
      reedCounter = maxReedCounter;//reset reedCounter
    }
    else{
      if (reedCounter > 0){//don't let reedCounter go negative
        reedCounter -= 1;//decrement reedCounter
      }
    }
  }
  else{//if reed switch is open
    if (reedCounter > 0){//don't let reedCounter go negative
      reedCounter -= 1;//decrement reedCounter
    }
  }
  if (time > 2000){
    mph = 0;//if no new pulses from reed switch- tire is still, set mph to 0
  }
  else{
    time += 1;//increment timer
  }
}

void displayMPH(){
  Serial.write(12);//clear
  Serial.write("Speed =");
  Serial.write(13);//start a new line
```

```
  Serial.print(mph);
  Serial.write(" MPH ");
  //Serial.write("0.00 MPH ");
}

void loop(){
  //print mph once a second
  displayMPH();
  delay(1000);
  checkBacklight();
}
```

Flere Links

http://www.gammon.com.au/forum/?id=11488

http://www.protostack.com/blog/2010/09/timer-interrupts-on-an-atmega168/
Gode oversigter over register!

http://www.instructables.com/id/Arduino-Timer-Interrupts/

Se: http://harperjiangnew.blogspot.dk/2013/05/arduino-using-atmegas-internal.html

CTC mode: http://maxembedded.com/2011/06/28/avr-timers-timer1/

http://www.youtube.com/watch?v=CRJUdf5TTQQ Jeremy Bloom)

http://www.uchobby.com/index.php/2007/11/24/arduino-interrupts/

http://www.protostack.com/blog/2010/09/timer-interrupts-on-an-atmega168/
http://www.instructables.com/id/Arduino-Timer-Interrupts/step2/Structuring-Timer-Interrupts/
http://www.instructables.com/id/Arduino-Timer-Interrupts/
http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/

God side om timere:
https://sites.google.com/site/qeewiki/books/avr-guide/timer-on-the-atmega8

http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/

Se eksempel på Cykel-computer:
http://www.instructables.com/id/Arduino-Bike-Speedometer/?ALLSTEPS

http://letsmakerobots.com/node/28278