Dokoment-links:

Arduino Timer Interrupts.

Hvordan får man Arduino til fx at udføre et interrupt og køre en programstump fx 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.

Et interrupt er et event – eller hændelse, - udløst enten internt fra en tæller / counter, - eller eksternt fra en pin, der ændrer status.

En Interruptrutine er en programstump, der udløses af en hændelse og kører asynkront, 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, tælleroverflow osv.

Interrupts udløses – og udføres et vilkårligt sted i et kørende program – og uafhængig heraf.

Fx kan et program køre – og samtidig kan der af en timer udløses et interrupt, der blinker en LED.

Det program, der køres når interuptet opstår, kaldes en **interrupt service routine (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 tilbagehopsadresse, ligesom vitale registres indhold gemmes ud på stakken. Det sker i baggrunden, og det er compileren, der genererer kode til det.

Når en ISR'en er færdig, vil det kørende program fortsætte hvorfra det blev afbrudt 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 også interupts!!! Det sker, når der er modtaget en hel byte, og den skal lægges i modtagebufferen. Og ligeledes i sendebufferen !!

Regler for en Interrupt-rutine.

En interruptsubrutine (eller Funktion) 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.

Her et par tommelfingerregler for interruptsubrutiner:

- 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

Volatile variable:

Variable, der deles mellem ISR-funktioner og normal kode bør erklæres "volatile". Dette fortæller compileren, at sådanne variables indhold kan ændres når som helst. Og derfor skal de reloades i

processorens interne registre hver gang der refereres til dem, snarere end at bruge en kopi, der måtte være i de interne registre.

Eks:

```
int pin = 13;
volatile int a = 3;
volatile int Count = 0;
volatile byte state = LOW;
```

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 bestemmes, 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.

Der gennemgås her to generelt forskellige interrupts: **Pinudløste**, - og **timer/counter** udløste interrupts.

Fordelen ved brug af interrupts er, at man kan have et Loop-program til fx at update et 7-segment display, - og så kan interrupts fx udløst hver 100-del sekund opdatere variable, der indeholder 100-del sekunder, sekunder, minutter osv.

Pin-udløste interrupts.

Kilder til afsnittet:

Se: http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/

Pinudløste interrupts kan sættes op til at ske ved en ændring på Arduino-pin 2 eller 3, henholdsvis INT0 & INT1.

Der kan vælges om det signal på pin'en, der udløser interruptet skal være rising, falling edge, eller pinchange.

Eksterne interrupts kan sættes op " the arduino way " – eller man kan selv "pille bit"!!

Et eksempel, " the Arduino Way":

```
void setup()
{
    pinMode(13, OUTPUT);     // Pin 13 is output to which an LED is connected
    digitalWrite(13, LOW);   // Make pin 13 low, switch LED off
    pinMode(2, INPUT);     // Pin 2 is input to which a switch is connected = INT0
    pinMode(3, INPUT);     // Pin 3 is input to which a switch is connected = INT1
    attachInterrupt(0, blink1, RISING);
    attachInterrupt(1, blink2, RISING);
                           // attachInterrupt(0, INT0_handler, CHANGE);  //
}

void loop() {
    /* Nothing to do: the program jumps automatically to Interrupt Service Routine
"blink" in case of a hardware interrupt on Ardiono pin 2 = INT0 */
}

void blink1(){              // Interrupt service routine
    digitalWrite(13, HIGH);
}

void blink2(){              // Interrupt service routine
    digitalWrite(13, LOW);
}
```

http://www.geertlangereis.nl/Electronics/Pin_Change_Interrupts/PinChange_en.html

```
detachInterrupt(0); // stopper interrupt !
detachInterrupt(1);
```

Arduino-funktionerne *attachInterrupt*() og *detachInterrupt*() bruges kun til pin-udløste interrupts.

Opsætningen af eksterne interrupts kan også styres ved at sætte bits i SFR-registre.

De SFR´s der bruges hertil er:

I External Interrupt Control Register A indstilles måden, man ønsker et interrupt udløst.

Eks: EICRA = 0b00000011 ;

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EICRA	-	-	-	-	ISC11	ISC10	ISC01	ISC00

External Interrupt Control Register A

ISCx1	ISCx0	DESCRIPTION
0	0	Low level of INTx generates an interrupt request
0	1	Any logic change on INTx generates an interrupt request
1	0	The falling edge of INTx generates an interrupt request
1	1	The rising edge of INTx generates an interrupt request

ISC Bit Settings

Et Extern Interrupt skal enables. Det sker i External Interrupt Mask Register, kaldet EIMSK

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EIMSK	-	-	-	-	-	-	INT1	INT0

External Interrupt Mask Register

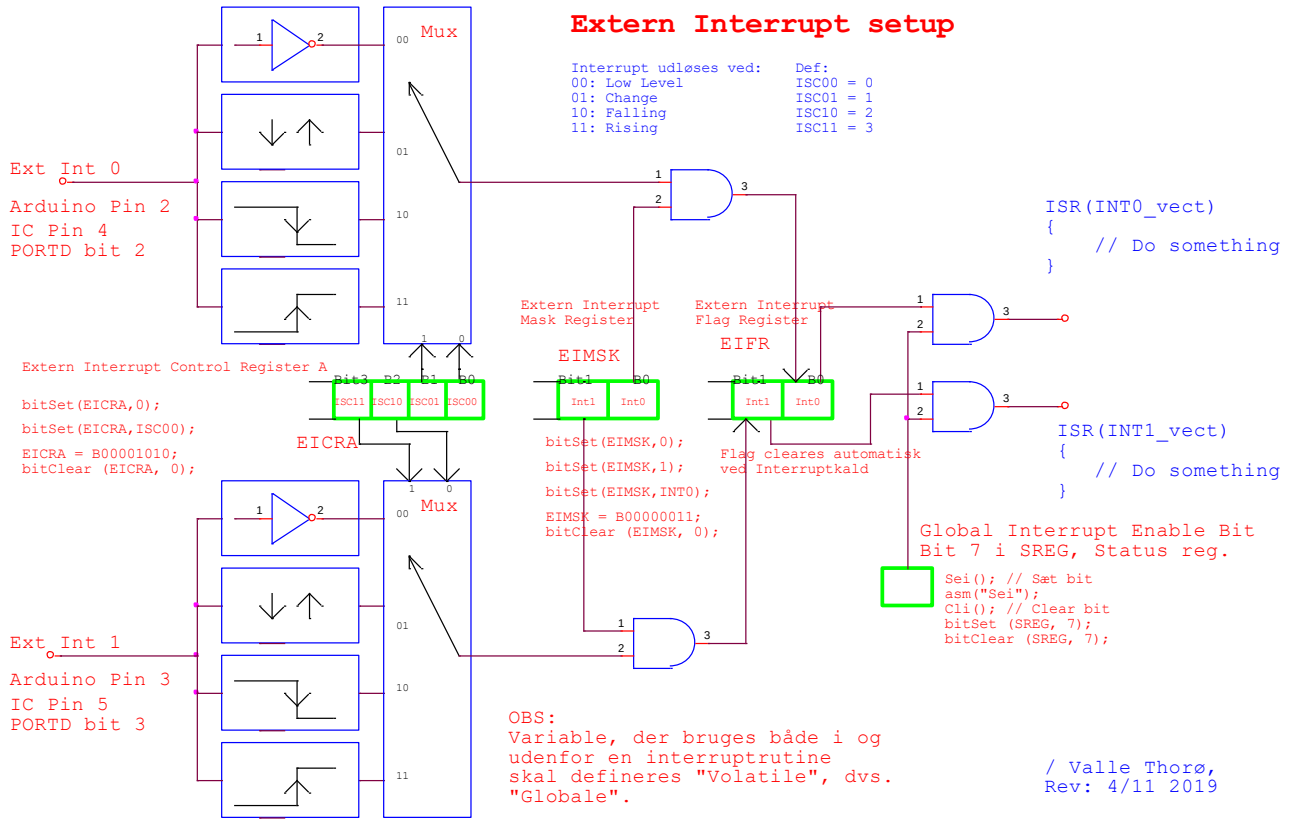
Og når et INTF flag bliver høj, udløses et interrupt.
Flaget resettes automatisk når Interrupt Service Rutinen udføres.

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EIFR	-	-	-	-	-	-	INTF1	INTF0

External Interrupt Flag Register

Kilde: https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328

Her forsøgt vist grafisk – og med eksempler - at sætte bit:



Kodeeksempler:

```
void setup(void)
{
    pinMode(2, INPUT);
    pinMode(13, OUTPUT);
    digitalWrite(2, HIGH);     // Enable pullup resistor
    sei();                     // Enable global interrupts
    EIMSK |= 0b00000001;       // Enable external interrupt INT0
    EICRA |= 0b00000010;       // Trigger INT0 on falling edge
}
// eller brug     bitSet(EICRA, 1);  fra:  bitSet(x, bitPosition)

void loop(void)
{
 // Do something
}
                                //
ISR(INT0_vect)  // Interrupt Service Routine attached to INT0 vector

{
    digitalWrite(13, !digitalRead(13));    // Toggle LED on pin 13
}
```

Men det er vist muligt, at få flere pins til at udføre interrupts. Se
http://www.geertlangereis.nl/Electronics/Pin_Change_Interrupts/PinChange_en.html
¿??

Arduino pin Interrupts, Blink LED

```
int pbIn = 0;                   // Interrupt 0 is on DIGITAL PIN 2!
int ledOut = 4;                 // The output LED pin
volatile int state = LOW;       // The input state toggle

void setup()
{
    // Set up the digital pin 2 to an Interrupt and Pin 4 to an Output
    pinMode(ledOut, OUTPUT);

    //Attach the interrupt to the input pin and monitor for ANY Change
    attachInterrupt(pbIn, stateChange, CHANGE);
}

void loop()
{
    //Simulate a long running process or complex task
    for (int i = 0; i < 100; i++)
    {
        // do nothing but waste some time
        delay(10);
    }
}

void stateChange()
{
    state = !state;
    digitalWrite(ledOut, state);
}
```

http://www.dave-auld.net/index.php?option=com_content&view=article&id=107:arduino-interrupts&catid=53:arduino-input-output-basics&Itemid=107

Her kommunikerer interruptet til hoved-loop via et flag, at der skal udføres noget

```
volatile boolean flag;

// Interrupt Service Routine (ISR)
void isr ()
{
```

```
 flag = true;
}  // end of isr

void setup ()
{
  attachInterrupt (0, isr, CHANGE);  // attach interrupt handler
}  // end of setup

void loop ()
{
  if (flag)
    {
    // interrupt has occurred
    }
}  // end of loop
```

Timerudløste interrupts

Alle Microcontrollere har indbyggede tællere, Nogle har flere! De 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 frekvens - til en tæller, for så på den måde at udmåle en tid der medgår til 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 tæller 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.

Man kan få det kørende program, der kører i Loop() { } til at afbrydes – interruptes - og udføre et såkaldt **interrupt-program** med bestemte tidsintervaller.

Tællerne kan fx indstilles / bringes til at udløse et interrupt når de har talt fra 0000h og op til en bestemt værdi, - det kaldes (**Compare match**), - eller nå der er talt op fra en given start-værdi og indtil tælleren giver **overflow**.

Korte interrupt-perioder kan direkte udløses ved hjælp af en timer. Men fordi timeren kun er 16 bit, er der grænser for, hvor lang tid, der kan udmåles direkte.

Derfor kan længere varighed håndteres ved fx at lade et 1-Hz interruptprogram tælle en variabel op, og når den fx når 100, (eller 1000) skal programmet fx foretage en måling. Programmet kunne her fx aflæse en sensor!!

Altså:

En Timer/Counter kan arbejde på to forskellige måder, når en tid skal udmåles.

Man kan indstille en startværdi for Tælleren, og så få et interrupt ved **overflow**

Eller man kan tælle fra 0, og så få et interrupt, når tælleren når en bestemt værdi, indstillet i et register, et **Compare**-register.

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	Der skal indsættes et tal, - en værdi i et Compare-register. Tælleren starter på 00 00h og der kan udløses et

sættes ind i tælleren.	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 timere 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

Prescaler. / Frekvensdeler.

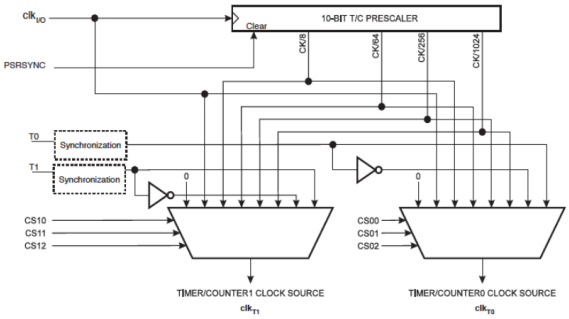
På Arduinoen sidder der et 16 MHz krystal. Derfor vil Timeren / Counteren køre ret hurtigt, og for et 16 bit register give overflow i løbet af ca. 4 mS.

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

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 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 registrene TCCR1A og TCCR1B. Navnene kommer af:

Timer**C**ounter**C**ontrol-**R**egister.

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

Se også fx http://forum.arduino.cc/index.php/topic,27390.0.html
Og: http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/

Udregning af Prescaler-værdi:

For at få udløst et interrupt med en korrekt frekvens, skal man udregne hvor meget man skal dele krystallets frekvens ned, så timeren ikke når at give overflow før tiden er gået.

Og man skal vælge hvilken værdi, der skal tælles op til (Compare Match) eller fra hvilken værdi, der skal tælles fra (Overflow)

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:

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

Compare match registeret er på 16 bit, og værdien kan derfor ikke være > 65535.

Overflow:

Her findes 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
(Denne side genererer også kode !!)

Mulige Prescalerværdier vs. Interruptfrekvens

Her er en oversigt for Tæller1 over **mulige interruptfrekvens-valg** og mulig prescaler-indstilling.

Prescaler Værdi	Maximal interrupt-Frevens	Minimal interrupt-Frekvens
1	16MHz	244Hz
8	2MHz	30,5Hz
64	250KHz	3,8Hz
256	62,5KHz	0,95Hz
1024	15,625KHz	0,24Hz

Og her en liste over udvalgte interruptfrekvenser, prescalerværdier og 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: Oversigt over involverede SFR-registre:

Hver timer har nogle Timer/Counter Control Registre tilknyttet:

Både for timer opsætning, for selve Tællerne,- men også bits, der skal sættes for at der kan udløses et interrupt:

TCCR1A og TCCR1B
Timer/Counter Control Register.
Prescaleren konfigureres her.

Består af 2 x 8 bit registre!!

Hjælp til valg af Prescaler til forskellige interruptfrekvenser, se fx calculator ovenfor

Prescaler kan vælges til 1, 8, 64, 256, 1024. (1024 kan ikke laves på timer 2)

Og Bit WGM12 og WGM13 bestemmer mode.
Vi vælger CTC-mode (timer) ved at sætte bit WGM12.

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	

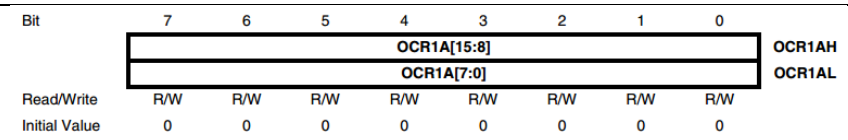
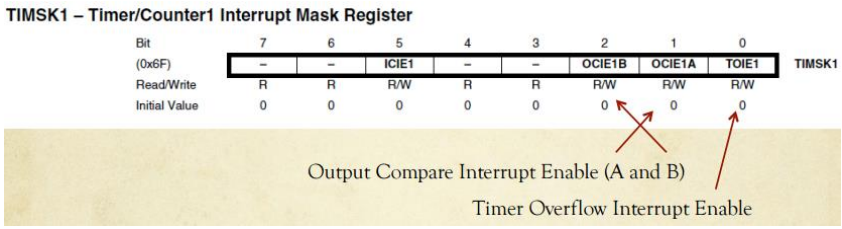
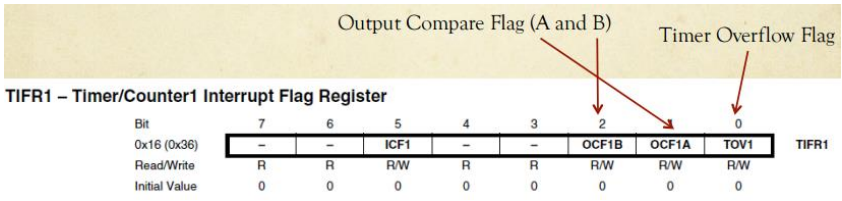
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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.

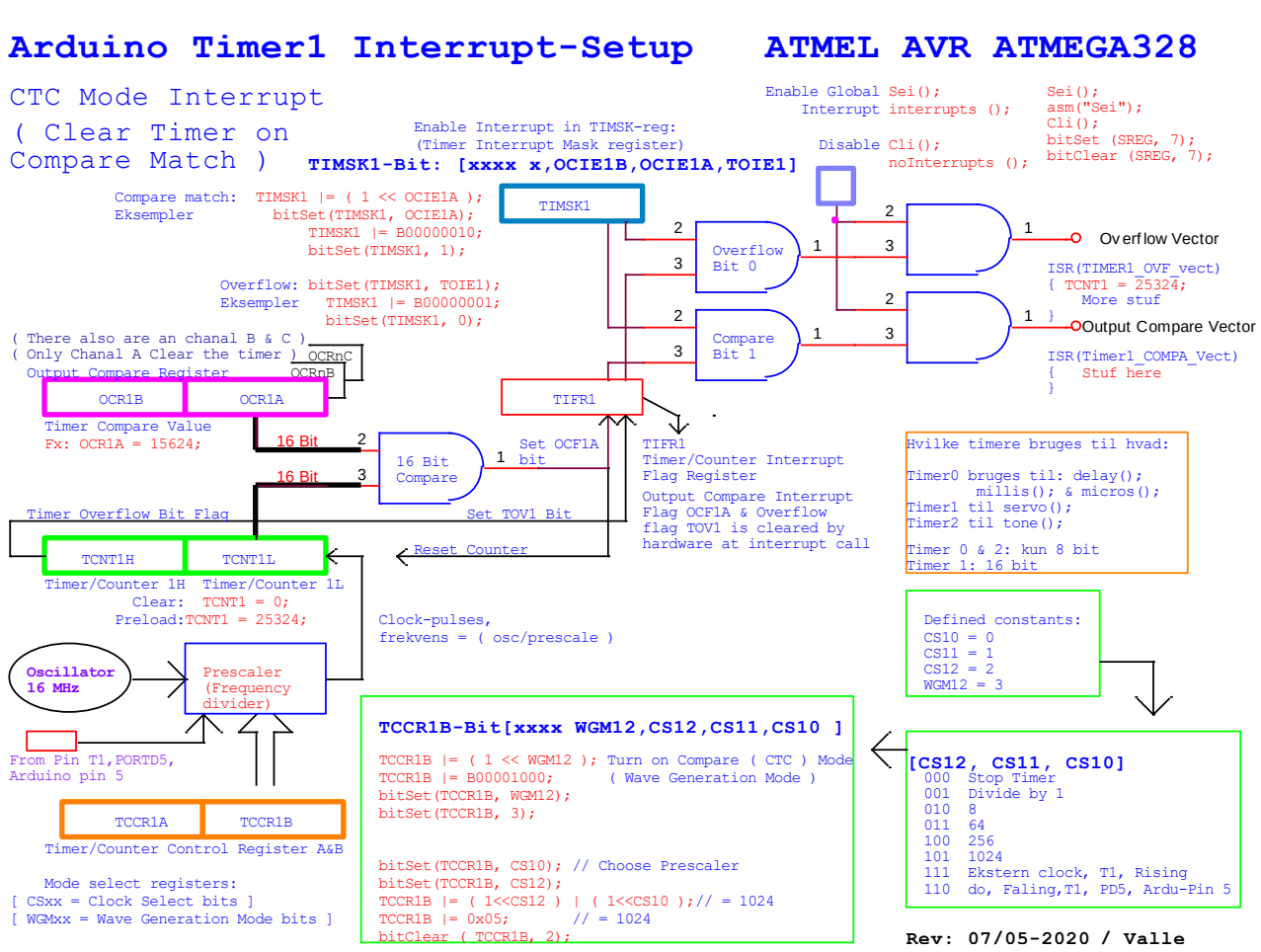
Indeholder den aktuelle tæller-værdi, 16 bit.

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	

TCNT1H og TCNT1L	
OCRxx - Output Compare Register, 16 bit. OCR1AH og OCR1AL	
TIMSKx Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.	 Timer_Overflow_
TIFRx - Timer/Counter Interrupt Flag Register. Indicates a pending timer interrupt.	

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

Oversigt:



Tegningen viser de forskellige SFR-registre involveret i opsætningen, og der er vist forskellige eksempler på hvordan man kan manipulere med bittene i registrene:

Her er en oversigt over, hvad der skal opsættes:

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

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

Indstil Prescal-værdi	`bitSet(TCCR1B, CS10); // = 00000001 = Bit 0`
	`bitSet(TCCR1B, CS11); // = 00000010 = Bit 1`
	`bitSet(TCCR1B, CS12); // = 00000100 = Bit 2`

Enable Timer1 interrupt for:
Compare Match
Overflow
	`bitSet(TIMSK1, OCIE1A); //Enable CompareInt.`
	`bitSet(TIMSK1, TOIE1); //Enable Overfl. Int`

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

Her er vist hvilke bit, der skal sættes for at vælge ønsket prescaler:

```
[CS12, CS11, CS10]
  000  Stop Timer
  001  Divide by 1
  010  8
  011  64
  100  256
  101  1024
  111  Ekstern clock, T1, Rising
  110  ", Faling,T1, PD5, Pin 5
```

Compileren kender alle SFR-navnene og deres placering i RAM, og også reserverede bitnavne, fx CS10. Derfor kan de umiddelbart bruges i fx en bitSet-instruktion.

For en oversigt over definerede værdier compileren kender:

Se flere eksempler på bitmanipulation i dokumentet: Port- og bitmanipulation

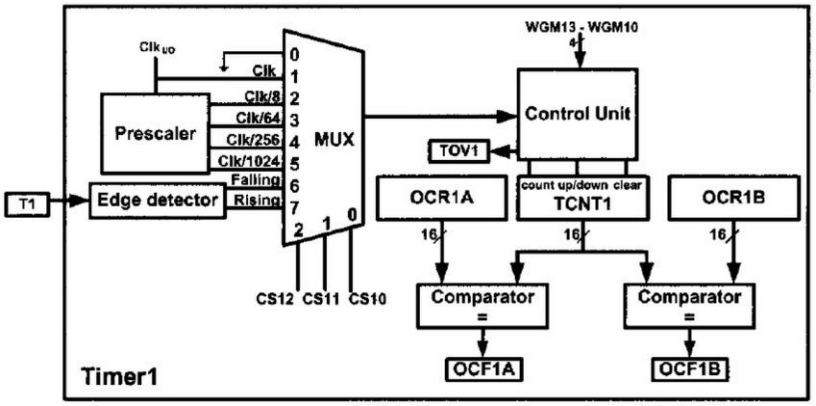
Interruptsetup henlagt til funktion

Her er vist et eksempel på en ” generel ” interruptopsætning. Den er henlagt til en funktion, og kan blot kopieres. Man skal så selv disable dvs. ud-kommentere de linjer, der ikke skal udføres.

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

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

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

  bitSet(TCCR1B, WGM12);         // turn on CTC mode:

                                 // Set Presaler:
                                 //       1: Set CS10
                                 //       8: Set CS11
                                 //      64: Set CS10 & CS11
                                 //     256: Set CS12
                                 //    1024: Set CS10 & CS12
                                 // Extern pulses: Set CS10, CS11 & CS12

  bitSet(TCCR1B, CS10);          // = 00000001 = Bit 0
  bitSet(TCCR1B, CS11);          // = 00000010 = Bit 1
  bitSet(TCCR1B, CS12);          // = 00000100 = Bit 2

                                 // Chose Compare Mode
  bitSet(TIMSK1, OCIE1A);        //Enable CompareInterrupt.
  OCR1A = 15624;                 // Set Compare Value

 // eller
                                 // Chose Overflow Mode
  bitSet(TIMSK1, TOIE1);         //Enable Overfl. Int
  TCNT1  = 34286;                // Preset Counter, all 16 bit.

  sei();                         //= interrupts();  Enable all Interrupts
}
```

Her en skitse af tæller1 og Output Compare register.

Der er faktisk 2 Compare-registre, men her er der kun set på OCR1A!!



Her følger nu et antal programeksempler, fundet derude !!!

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 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 "Overflov-udløst" interrupt-rutine.

```
ISR(TIMER1_OVF_vect)
```

```
{
  TCNT1 = 34286;                  // reload timer, nødvendig

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

// Bits that are "bitwise XORed" with 1 are toggled
}
```

Redigering hertil 31/10-2019

Valg af Timermode

Timerne i uC-en kan arbejde i et større antal modes. Derfor er man nødt til at vælge i hvilken mode, man vil bruge timerne.

Simpel pulstælling kaldes CTC Mode (Counter-Timer-Compare)

Denne mode vælges ved at sætte ”WGM”-bit for pågældende tæller.

```
bitSet (TCCR0A, WGM01);        //for timer0
bitSet(TCCR1B, WGM12);         //for timer1
bitSet(TCCR2A, WGM21);         //for timer2
```

Kilde # .[1]

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.
```

[1] Fra: (http://www.instructables.com/id/Arduino-Timer-Interrupts/step2/Structuring-Timer-Interrupts/)
Datablad: http://www.atmel.com/Images/doc8161.pdf

```
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:

http://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture7.pdf
http://www.avrbeginners.net/architecture/timers/timers.html
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 en god dok: http://www.eng.utah.edu/~cs5789/2010/slides/Interruptsx2.pdf

Se datablad: http://www.atmel.com/Images/doc8161.pdf

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 presettes timerne 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 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 det 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 en 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.

Resten er endnu ikke helt gennemarbejdet, kan betragtes som Bonus Info

Arduino timer compare match interrupts:

Timer 0:

Eksempler på opsætning af Prescaler:

```
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
```

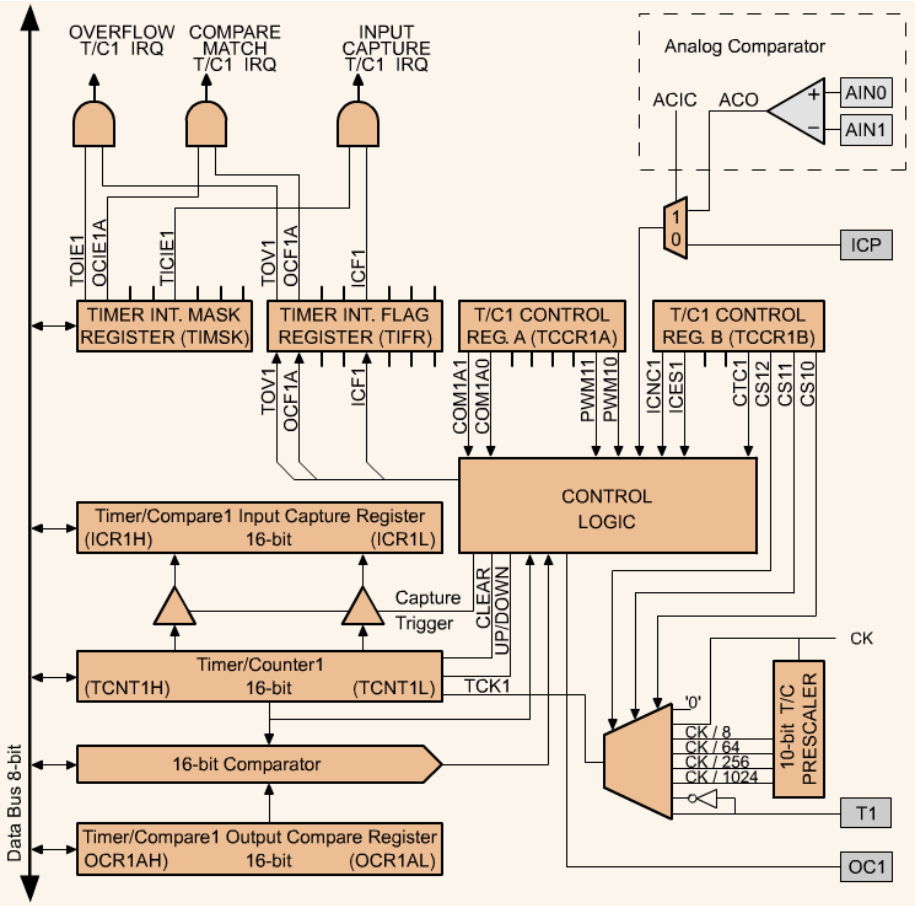
```
  TCCR1B = 0;           // same for TCCR1B
  TCNT1  = 0;           //initialize counter value to 0
  OCR1A = 15624;        // set compare match register for 1hz
                        // increments
                        // = (16*10^6) / (1*1024) - 1 (must be <65536)

  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/

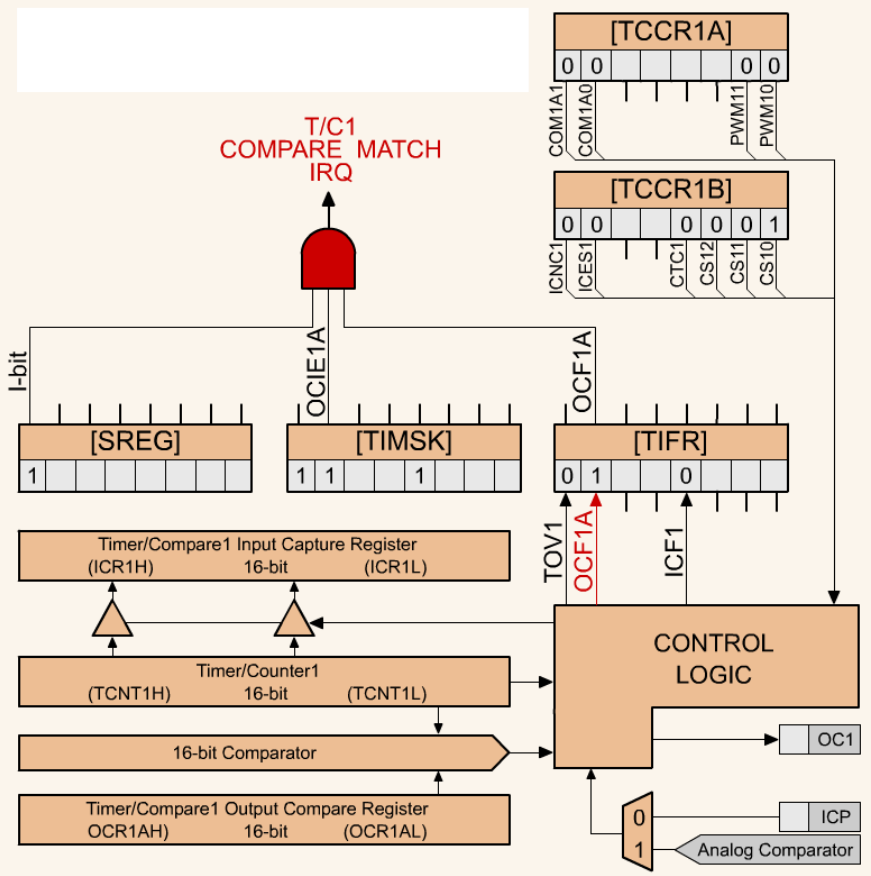
Oversigtstegninger og animationer over timer-opbygning i ATMEGA processoren.

Der findes også andre skitser over hvordan interrupt-strukturen sættes op.

Bemærk, at PWM1x vist nu hedder WGM1x ?????



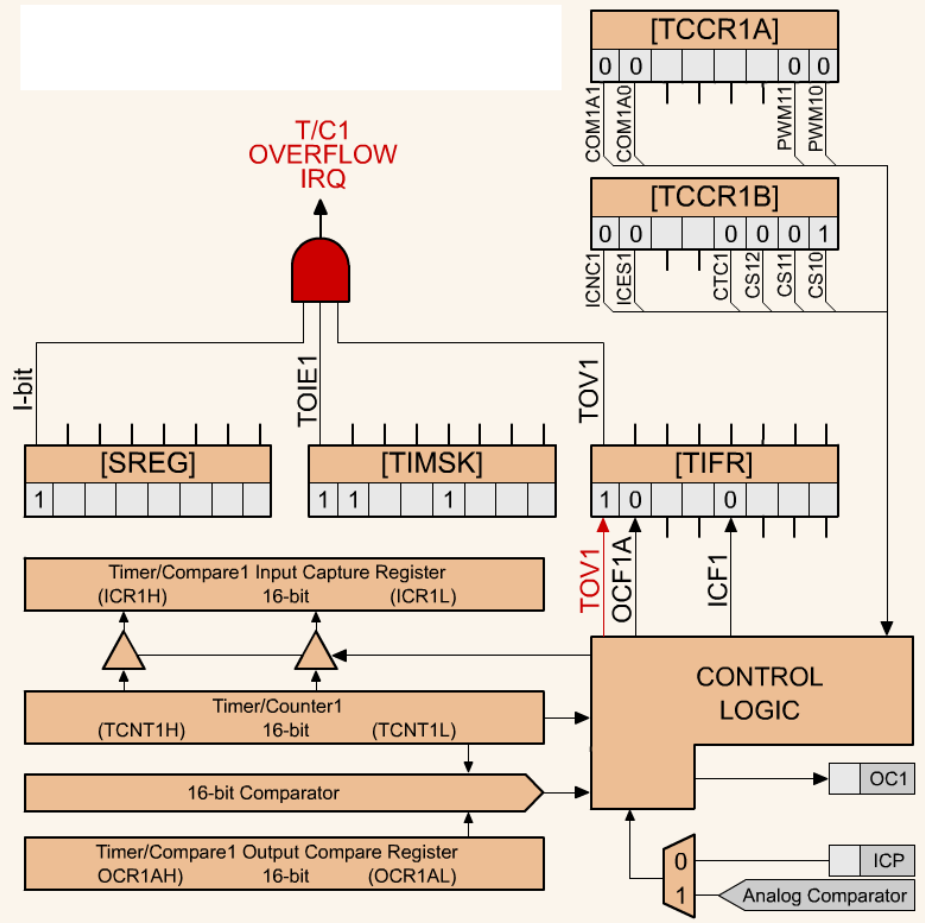
http://www.kner.at/home/40.avr/_architektur/ztcpwm.html



http://www.kner.at/home/40.avr/_architektur/ztcbint.html

Her er animeret en compare match. Se link:

Og en overflow match.



http://www.kner.at/home/40.avr/_architektur/ztcbint.html

Example- the following sketch sets up and executes 3 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();
}
```

For this project, I used timer2 interrupts to periodically check if there was any incoming serial data, read it, and store it in the matrix "ledData[]". If you take a look at the code you will see that the main loop of the sketch is what is actually responsible for using the info in ledData to light up the correct LEDs and checking on the status of the buttons (a function called "shift()"). The interrupt routine is as short as possible- just checking for incoming bytes and storing them appropriately.

Here is the setup for timer2:

```
cli();      //stop interrupts
            //set timer2 interrupt every 128us
TCCR2A = 0; // set entire TCCR2A register to 0
TCCR2B = 0; // same for TCCR2B
TCNT2  = 0; //initialize counter value to 0
            // set compare match register for 7.8khz increments
OCR2A = 255;// = (16*10^6) / (7812.5*8) - 1 (must be <256)
            // turn on CTC mode
TCCR2A |= (1 << WGM21);
            // Set CS21 bit for 8 prescaler
TCCR2B |= (1 << CS21);
            // enable timer compare interrupt
TIMSK2 |= (1 << OCIE2A);
sei();      //allow interrupts
```

Here's the complete Arduino sketch: Ret kompliceret !!

```
//BUTTON TEST w/ 74HC595 and 74HC165 and serial communication
//by Amanda Ghassaei
//June 2012
//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 2 of the License, or
 * (at your option) any later version.
 *
*/
```

```
//this firmware will send data back and forth with the maxmsp patch "beat
slicer"

//pin connections
#define ledLatchPin A1
#define ledClockPin A0
#define ledDataPin A2
#define buttonLatchPin 9
#define buttonClockPin 10
#define buttonDataPin A3

//looping variables
byte i;
byte j;
byte k;
byte ledByte;

//storage for led states, 4 bytes
byte ledData[] = {0, 0, 0, 0};
//storage for buttons, 4 bytes
byte buttonCurrent[] = {0,0,0,0};
byte buttonLast[] = {0,0,0,0};
byte buttonEvent[] = {0,0,0,0};
byte buttonState[] = {0,0,0,0};
//button debounce counter- 16 bytes
byte buttonDebounceCounter[4][4];


void setup() {
  DDRC = 0xF7;//set A0-2 and A4-5 output, A3 input
  DDRB = 0xFF;//digital pins 8-13 output

  Serial.begin(57600);

  cli();//stop interrupts

  //set timer2 interrupt every 128us
  TCCR2A = 0;// set entire TCCR2A register to 0
  TCCR2B = 0;// same for TCCR2B
  TCNT2  = 0;//initialize counter value to 0
  // set compare match register for 7.8khz increments
  OCR2A = 255;// = (16*10^6) / (7812.5*8) - 1 (must be <256)
  // turn on CTC mode
  TCCR2A |= (1 << WGM21);
  // Set CS21 bit for 8 prescaler
  TCCR2B |= (1 << CS21);
  // enable timer compare interrupt
  TIMSK2 |= (1 << OCIE2A);

  sei();//allow interrupts

}

// buttonCheck - checks the state of a given button.
//this buttoncheck function is largely copied from the monome 40h firmware by
brian crabtree and joe lake
void buttonCheck(byte row, byte index)
{
```

```
  if (((buttonCurrent[row] ^ buttonLast[row]) & (1 << index)) &&   // if the
current physical button state is different from the
  ((buttonCurrent[row] ^ buttonState[row]) & (1 << index))) {  // last physical
button state AND the current debounced state

    if (buttonCurrent[row] & (1 << index)) {                     // if the
current physical button state is depressed
      buttonEvent[row] = 1 << index;              // queue up a new button
event immediately
      buttonState[row] |= (1 << index);                          // and set the
debounced state to down.
  }
    else{
      buttonDebounceCounter[row][index] = 12;
    }  // otherwise the button was previously depressed and now
    // has been released so we set our debounce counter.
  }
  else if (((buttonCurrent[row] ^ buttonLast[row]) & (1 << index)) == 0 &&  //
if the current physical button state is the same as
  (buttonCurrent[row] ^ buttonState[row]) & (1 << index)) {        // the last
physical button state but the current physical
    // button state is different from the current debounce
    // state...
    if (buttonDebounceCounter[row][index] > 0 && --
buttonDebounceCounter[row][index] == 0) {  // if the the debounce counter has
      // been decremented to 0 (meaning the
      // the button has been up for
      // kButtonUpDefaultDebounceCount
      // iterations///

      buttonEvent[row] = 1 << index;    // queue up a button state change event

      if (buttonCurrent[row] & (1 << index)){          // and toggle the
buttons debounce state.
        buttonState[row] |= (1 << index);
      }
      else{
        buttonState[row] &= ~(1 << index);
      }
    }
  }
}


void shift(){

  for (i=0;i<4;i++){

    buttonLast[i] = buttonCurrent[i];

    byte dataToSend = (1 << (i+4)) | (15 & ~ledData[i]);

    // set latch pin low so the LEDs don't change while sending in bits
    digitalWrite(ledLatchPin, LOW);
    // shift out the bits of dataToSend
    shiftOut(ledDataPin, ledClockPin, LSBFIRST, dataToSend);
    //set latch pin high so the LEDs will receive new data
    digitalWrite(ledLatchPin, HIGH);
```

```
    //once one row has been set high, receive data from buttons
    //set latch pin high
    digitalWrite(buttonLatchPin, HIGH);
    //shift in data
    buttonCurrent[i] = shiftIn(buttonDataPin, buttonClockPin, LSBFIRST) >> 3;
    //latchpin low
    digitalWrite(buttonLatchPin, LOW);

    for (k=0;k<4;k++){
      buttonCheck(i,k);
      if (buttonEvent[i]<<k){
        if (buttonState[i]&1<<k){
          Serial.write(((3-k)<<3)+(i<<1)+1);
        }
        else{
          Serial.write(((3-k)<<3)+(i<<1)+0);
        }
        buttonEvent[i] &= ~(1<<k);
      }
    }
  }
}


ISR(TIMER2_COMPA_vect) {
  do{
    if (Serial.available()){
      ledByte = Serial.read();//000xxyys
      boolean ledstate = ledByte & 1;
      byte ledy = (ledByte >> 1) & 3;
      byte ledx = (ledByte >> 3) & 3;
      if (ledstate){
        ledData[ledy] |= 8 >> ledx;
      }
      else{
        ledData[ledy] &= ~ (8 >> ledx);
      }
    }//end if serial available
  }//end do
 while (Serial.available() > 8);
}


void loop() {
  shift();//updates leds and receives data from buttons
}
```

```
int ledPin = 8;
volatile int compAval = 10;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  sei();             //Enable interrupts
  TCCR2A |= 3;           //Fast PWM mode 3.
  TCCR2A |= (3 << 6);        //fire INT on Compare match
  TCNT2 = 0;             //initialize Timer2 to 0
  OCR2A = compAval;          //Set compare A value
  TIMSK2 |= (1 << 1);        //Enable CompA interrupt
  TIMSK2 |= 1;           //Enable OVF interrupt
  TCCR2B |= 7;           //Clock select. Internal, prescale  1024~64us
}

ISR(TIMER2_COMPA_vect)
{
  digitalWrite(ledPin, HIGH);
}


ISR(TIMER2_OVF_vect)
{
  digitalWrite(ledPin, LOW);
  compAval = compAval + 4;
  if(compAval > 254)
      compAval = 10;
  OCR2A = compAval;
}


void loop()
{
  //Serial.println(TCNT2, DEC);
}
```

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