



Atmega328P poweret af et batteri

Links til afsnit i dokumentet:

[Batterier](#), [Spændingsregulator](#),

[Sleep-Modes](#), [Wake-up-metoder](#), [WatchDog-Timer](#), [External Interrupt-Wake-Up](#),

[Hvordan laves ”lang” Sleep-periode ?](#)

Ekstra Strømbesparelse: [BrownOut Detector](#), [A/D-Converter](#)

Kodeeksempler

Hvis noget udstyr skal køre på batteri, - er det vigtigt, at det bruger så lidt strøm, som muligt.

Og hvis udstyret fx skal foretage sig noget med mellemrum, fx at foretage en måling hvert minut, er det en meget stor fordel at putte processor i sleep-mode, mellem aktiviteterne.

Herved spares der på strømmen.

Måske skal der bruges ekstra udstyr, - kan processoren disable / afbryde det ekstra udstyr før sleep. Fx via en Mosfet-switch.

Drejer det sig fx om radiosenderen HC12, har den indbygget sleep-mode, som kunburger 22 μ A

Arduinos strømforbrug / StandAlone:

Arduino-boardet har et forbrug i sig selv på ca. 40 - 50 mA. Det er incl. USB-IC-en på boardet og 5-volts-regulatoren.

Opbygger man kredsløbet som Stand-alone, - fx på fumlebrædt – med et 16 MHz krystal vil strømforbruget iflg. forskellige kilder være i størrelsen 15 – 20 mA (Selv målt til 17 mA.)

I Stand Alone – og Sleep-Mode kan strømforbruget iflg. kilder komme ned i størrelsen 360 μ A

Pins:

I forskellige kilder, er der anvist, at for at opnå størst strømbesparelse, bør alle pins defineres som OUTPUT, og være LOW, - eller defineres som INPUT, og være LOW. Dvs. ingen interne PullUps enabled !!

Spændingsregulator.



Hvis man bygger udstyr til batteri-drift, vil det selvfølgelig betyde noget, hvis spændingsregulatoren selv bruger en ” stor ” strøm.

Da der ikke findes et 5 Volts batteri, inkluderer et kredsløb typisk også en spændingsregulator.

En almindelig LM7805 har en dropout spænding (se note ¹) på ca. 3 Volt, og et egetforbrug (Quiescent) på 5 mA.

Det kan gøres meget bedre af en LDO-type, (Low Drop Out) fx LM2931Z-5.0V, men den er til max 100 mA.

Type	Dropout Voltage [V]	Quiescent m [A]	Maks strøm m [A]	Hus
7805	3	ca. 5	1000	TO-220
TS2940, LDO	0,1 – 0,6 @ Iout = 100 – 800 mA	10 ?	1000	TO-220
LM2931z-5.0V, LDO	0,6	0,4 Afhængig af load !	100	TO92



Batterier:

Her er en oversigt over forskellige typiske batterier.

Type	Kapacitet mAh
CR1212	18
CR1620	68
CR2032	210
NiMH AAA	900
Alkaline AAA	1250
NiMH AA	2400
Alkaline AA	2890
Li-Ion *	4400

* Li-Ion batterier fås i forskellige variationer og størrelse, så dette er blot et eksempel !

Regneeksempel:

¹ Dropout-spænding er den højere spænding, regulatoren skal ha på dens indgang end output-spændingen.



Bruges et Alkaline AAA batteri, og der gennemsnitlig trækkes 0,05 mA, kan udstyret køre 1250 / 0.05 = 25000 timer (1041 dage, eller 33 måneder).

Herudover er det jo bare sådan, at batterier har noget selvafladning.

Sleep Modes og Opvågning

Hver gang, man lader en processor gå i Sleep, må man også forholde sig til, hvordan man får den til at vågne igen.

Arduinos ATMEGA328P har 5 sleep-modes. Hver mode virker forskelligt, og i hver mode kan man ”slukke” for forskellige dele af de indre funktioner, der derved ikke trækker strøm.

Og i hver mode kan strømforbruget komme forskelligt ned. Og til hver mode er der forskellige Wake-muligheder.

Det skitseres i følgende graf:

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
	clk _{CPU}	clk _{FLASH}	clk _{IO}	clk _{ADC}	clk _{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/I/O	
Idle			X	X	X	X	X ⁽²⁾	X	X	X	X	X	X	X	
ADC noise Reduction				X	X	X	X ⁽²⁾	X ⁽³⁾	X	X ⁽²⁾	X	X	X		
Power-down								X ⁽³⁾	X				X		X
Power-save					X		X ⁽²⁾	X ⁽³⁾	X	X			X		X
Standby ⁽¹⁾						X		X ⁽³⁾	X				X		X
Extended Standby					X ⁽²⁾	X	X ⁽²⁾	X ⁽³⁾	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.

2. If Timer/Counter2 is running in asynchronous mode.

3. For INT1 and INT0, only level interrupt.

Til hver Sleep-mode hører der et antal ”Wake” muligheder. Altså en måde at bringe CPU-en tilbage til ”arbejde”, og fortsætte det program, den var ved at udføre lige før Sleep-instruktionen.

I det følgende er det valgt at se på den Sleep-mode, - **Power-down**, - der kan give den mindste strømforbrug, - og de to Wake-muligheder, **Watchdog Timer Interrupt**, og **Ekstern Pin-Interrupt**.



På nettet findes et hav af eksempler, hvor der gøres brug af forskellige biblioteker der kan inkluderes og bruges til Sleep og Wake. (fx sleep.h eller LowPower.h).

Men her gennemgås, hvordan man selv kan ” pille bit ” i nogle SFR-registre.

Det er jo også fuldstændigt det samme, der sker skjult inde i bibliotekerne !!

Sleep-Mode SFR-Register:

Når en MCU sættes i Sleep, går den i dvale. Afhængig af valgte Mode, vil forskellige af de indre enheder blive disablet, så de ikke bruger strøm.

Man kan så få Processoren til at vågne igen på forskellige måder afhængig af valgte Sleep-Mode, hvorefter den fortsætter det program, den var i gang med.

Valg af Sleep-Mode Sleepmode styres ved at sætte bits i ” Sleep Mode Control Registeret ”, SMCR-Registeret:

Bit	7	6	5	4	3	2	1	0
Access					SM2	SM1	SM0	SE
Reset					R/W	R/W	R/W	R/W

Bit 1, 2 og 3 vælger ønsket sleepmode.

Her er vist hvordan de 3 bit, der vælger Sleep-Mode skal sættes:

Her arbejdes med **Power-Down**, dvs. at der skal sættes et ”1” i bit 2 i Sleep-mode kontrolregisteret, SMCR

Og også bit 0, Sleep Enable-bit skal sættes

SM2,SM1,SM0	Sleep Mode
000	Idle
001	
010	Power-down
011	Power-save
100	Reserved
101	Reserved
110	Standby ⁽¹⁾
111	Extended Standby ⁽¹⁾

```
bitSet(SMCR, 2); //set_sleep_mode(SLEEP_MODE_PWR_DOWN);
```

Bit 0 er et Sleep Enable Bit, der kan enable eller disable den valgte Sleep Mode.

I databladet anbefales, at Sleep Enable-bittet sættes lige før hver Sleep, og at det cleares igen efter Wake.

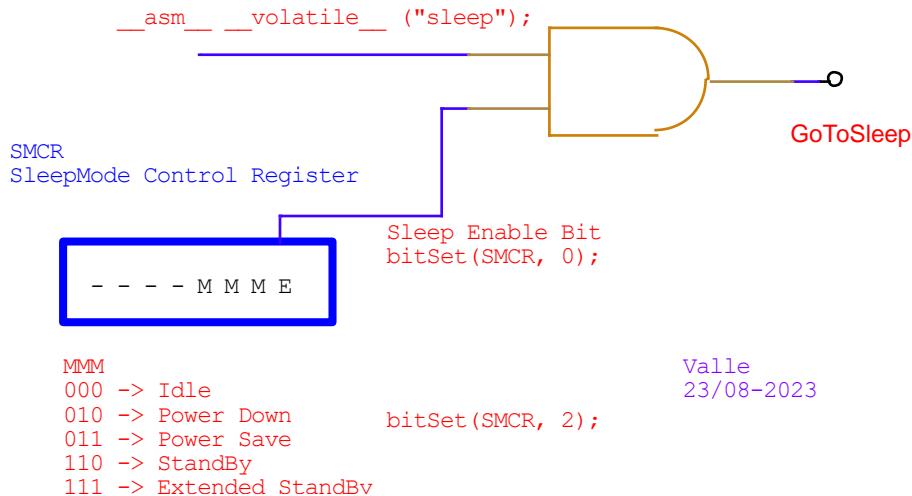


```
bitSet(SMCR, 0); //sleep_enable();
-
bitClear(SMCR, 0); //sleep_disable();
```

Efter at Sleep Enable bit er sat, skal der udføres en speciel SLEEP-instruktion, for at processoren går i Sleep Mode. Det er ikke en ”C++-Komando”, men en fra den gamle Assembler-verden:

```
asm volatile ("sleep"); // "Volatile" for at "hjælpe" compileren
```

Når processoren vågner, udføres først – i dette tilfælde - et interrupt, hvorefter programmet fortsætter i næste linje efter ”Sleep-instruktionen”. Indholdet i RAM mm. ændres ikke under Sleep!!



Ekstra enheder, der kan disable for yderligere at nedsætte strømforbruget under Sleep:

For yderligere at nedsætte strømforbruget under Sleep, kan man før Sleep slukke for et par af de indre enheder. Det drejer sig her om [AD-Converteren](#) og [Brown-Out-detektoren](#).

Herved kan vist spares i størrelsen 25 uA mere. Se senere !!

Wake up fra Power-Down Mode

Wake-metoder

I Power-Down mode kan processoren vækkes af en af disse hændelser

- External Reset
- Watchdog System Reset



Den / de valgte wake-funktioner skal også indstilles i ” deres ” respektive SFR-register.

- **Watchdog Interrupt**
- Brown-out Reset
- 2-wire Serial Interface address match
- **External interrupt on INT0 / INT1**
- Pin change interrupt

I det følgende gennemgås de to af mulighederne, – [Watchdog-Timer-Interrupt](#) og [Extern Interrupt fra INT0](#).

Watch Dog Timer

En watch dog timer, nogle gange kaldet ” Computer Operating Properly Timer, (COP timer) er en timer, som bruges til at overvåge, at CPU-en kører normalt.

I store programmer er der næsten altid fejl (Bugs). Men også hardwaren, der udfører perfekt kode, vil også kunne fejle.

Herudover vil kosmisk stråling kunne give problemer. Kosmisk Stråling består for det meste af højenergi-protoner fra rummet. Og de kan interagere med transistorer på IC'er og evt. flippe en bit.

I Mikroprocessorens barndom var det ikke så stort et problem som i dag, fordi geometrien og banetykkelserne var meget større end i dag. Så der var ikke nok energi i strålingen til at påvirke en bit.

Men efterhånden som processorerne fremstilles med 45 nm eller 28 nm baner, er problemet blevet større.

I 90'erne fandt IBM, at en typisk computer oplevede omkring 1 fejl pga. kosmisk stråling pr. 256 MB RAM. Med meget mindre geometri i dag er problemet formodentlig større.

Ideen med en WatchDog er, at en timer - med en bestemt inputfrekvens - når at tælle ud, dvs. give overflow, og kan resette selve CPU-en, - eller starte et interrupt.

I det normalt kørende program, skal WatchDog Timeren så resettes med jævne mellemrum, så overflow forhindres.

Kører programmet ” i skoven ” resettes timeren ikke – hvorefter en handling udføres.

Hvis handlingen er et interrupt, kan eventuelle essentielle data gemmes før program-reset.

Men dette WDT-overflow kan også bruges til at Wake CPU-en fra Sleep.

Opsætning af WDT



WatchDog-timeren sættes op ved at sætte bits i SFR-en

WatchDog Control Register:

Bit	7	6	5	4	3	2	1	0	WDTCSR
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Her er både bits til at indstille en Prescaler, - bit 5, 2, 1 & 0.

Og der er nogle bit til at styre hvordan WDT-en skal virke.

Indstilling af WDT

Indstilling af WDT'en er lidt kompliceret.

Det er lavet sådan, for at man ikke ved et uheld kommer til at ændre på opsætningen.

Først skal bit 4 (WDCE, Watch Dog Change Enable) sættes, og inden 4 clockcykler skal de øvrige bit indstilles.

Også Prescaleren !

Prescaler:

Ved at indstille nogle bit, kan man vælge den tid, der går før WD-timeren giver overflow.

Vha. en prescaler kan man vælge den tid, der går før Watchdog timeren giver overflow:

Prescaleren neddeler frekvensen fra en separat intern ca. 128 kHz oscillator. !

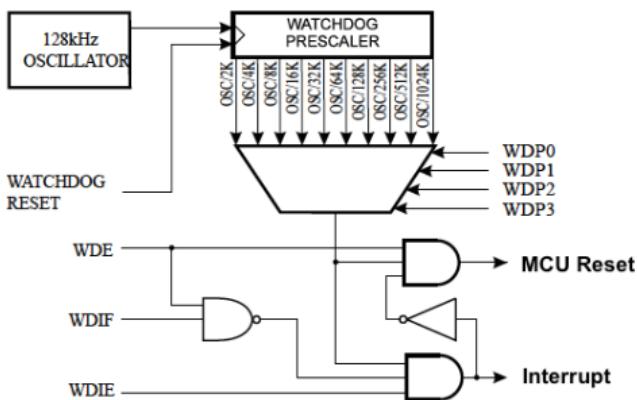
WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V _{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125s
0	1	0	0	32K (32768) cycles	0.25s
0	1	0	1	64K (65536) cycles	0.5s
0	1	1	0	128K (131072) cycles	1.0s
0	1	1	1	256K (262144) cycles	2.0s
1	0	0	0	512K (524288) cycles	4.0s
1	0	0	1	1024K (1048576) cycles	8.0s

De viste 8.0 sekunder er nærmere 8,192 sekunder, men 128 kHz oscillatoren er ikke krystal-styret, så den er jo ikke nøjagtig !!

Værdien 1001 i de 4 Prescaler-bit giver ca. 8 sekunder !!



WDTCSR:



WDP0 til 3 er til at indstille prescalen.

WDE: WD MCU- reset Enable bit

WDF: WD Interrupt Flag. *1)

WDIE: WD Interrupt Enable

WDCE: WD Change Enable*2)

*1, Sættes ved overflow, Cleares automatisk ved Interrupt-kald

*2, Sættes først, og inden for 4 clk-cycles skal de øvrige bit sættes !

(Under ”normal” brug af WatchDog timeren, skal programmet regelmæssigt resette Watchdog timeren, så den ikke giver overflow – og aktiverer et program-genstart. Det kan ske med følgende instruktion:

```
asm volatile ("wdr"); ( Se note 2) ( Se note 3 )
```

WatchDog Control Register: (igen)

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Bemærk, at Watch Dog Prescaler-bittene er bit 5, 2, 1 & 0

WDIE: Watchdog Interrupt Enable – hvis dette bit er sat, vil en timeout trigge et interrupt.

WDCE: Watchdog Change Enable – hvis man ønsker at ændre indstillingerne af WDT-en, skal dette bit først sættes, og inden for 4 clockcykler skal de øvrige bit være sat

WDE: Watchdog System Reset Enable – hvis dette bit er sat, vil en timeout trigge et reset af microcontrolleren.

² The volatile qualifier with the asm statement is something one should do in order to avoid that the compiler optimizes the assembly code away or moves it out of a loop (strictly speaking, this is only necessary when there is an output operand that is not used afterwards in the C++ code, but it never hurts anyway).

³ Kan også defineres (omdøbes) før setup med:

```
#define wdt_reset() __asm__ __volatile__ ("wdr")
Herefter kan wdt_reset(); bruges:
```



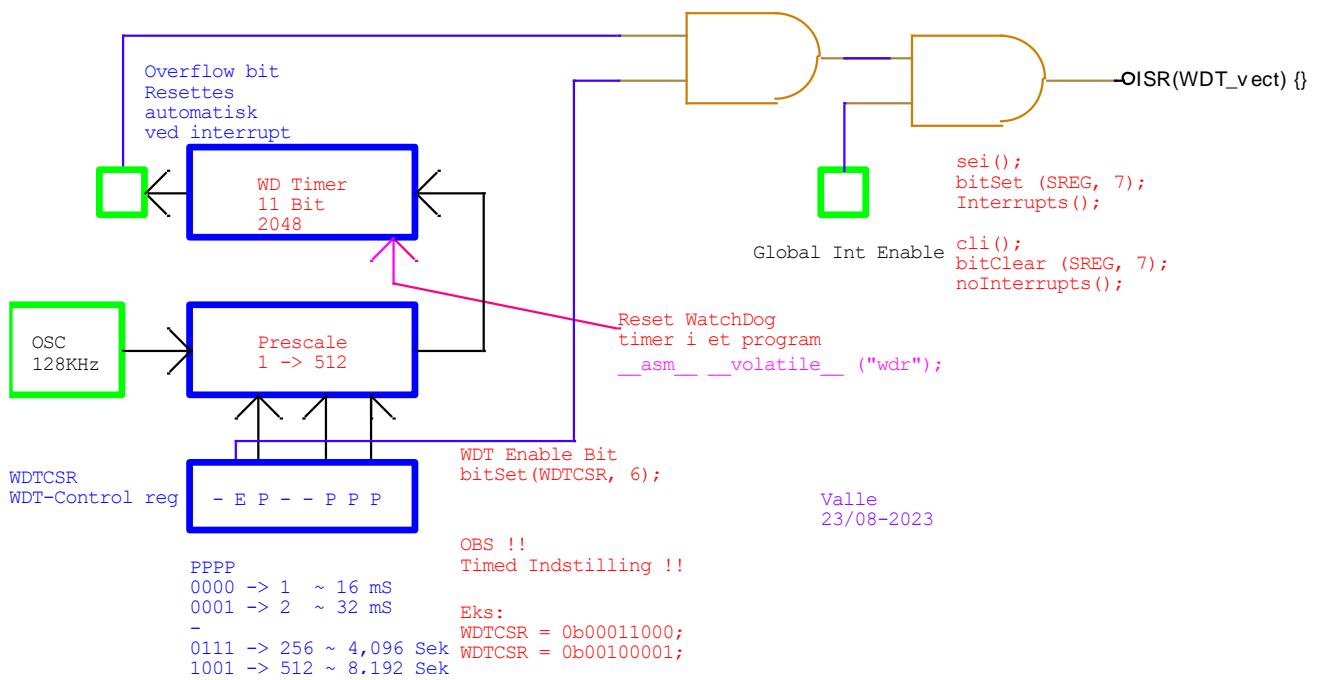
Her er vist et opsætningseksempel til opsætning af WDT til Interrupt:

```
// Setup Watchdog Timer
WDTCSR = 0b00011000;      // 18h change enable and WDE - also resets
WDTCSR = 0b00100001;      // 21h, Prescalers only - get rid of WDE og WDCE bit
bitSet(WDTCSR, 6);        // Sæt bit WDIE, Watch Dog Interrupt Enable
```

Når der er opsat et interrupt, skal man huske den tilhørende ISR, - Interrupt-Service Routine:

```
ISR(WDT_vect) {
    // Do Nothing
}
```

Illustreret i en graf, kunne det se således ud:



Der behøves ikke nødvendigvis ske noget i ISR-en, blot kaldet til den vækker CPU-en.

Wake up med extern Pin Interrupt.

Med denne metode er det et externt interrupt, der får CPU-en til at vågne.



Der behøver ikke at ske noget i Interrupt-Service Rutinen. Det er selve kaldet til den, der vækker CPU-en.

Det eksterne interrupt skal sættes op før processoren går i Sleep-mode. Evt. i setup !!

Det eksterne interrupt virker her på samme måde som gennemgået i et andet dokument om Ekstern interrupt.

Der er 2 muligheder for eksterne interrupt, INT0 og INT1, på IC-ens hhv. pin 4 og 5

Et eksternt interrupt kan opsættes til at ske på et stigende eller et faldende signal på dets pin.

Det kan selvfølgelig laves med en trykknap, men det er jo ikke praktisk hvis der fx skal stå noget udstyr ved en å, og måle temperaturen i intervaller.

Jeg har haft problemer med mine test, hvor jeg brugte intern pullup og en trykknap til Gnd. Ofte gik processoren i koma efter et fåtal af wake via trykknappen.

Det hjalp efter der blev monteret en kontakt-prell-fjerner (lavet med en 4050).

Se Fodnote: ⁴ fra [Databladet](#):

Evt. kan man ty til en CMOS Oscillator og tæller, fx 4093 og 4040, så man kan få et interrupt-signal med passende langt interval.

Se: https://www.youtube.com/watch?v=urLSDi7SD8M&ab_channel=KevinDarrah

External Interrupt Indstilling:

Indstilles der et eksternt interrupt, vil processoren vågne så snart der sker et kald til Interruptets Service-rutine, dets ISR..

Indstillingen sker i SFR-registeret EICRA, External Interrupt Control Register A EICRA

Bit	7	6	5	4	3	2	1	0	
(0x69)	-	-	-	-	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

⁴ If a level triggered interrupt is used for wake up from power down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated.



Vha. bit 0 & 1 indstilles interrupt-type for INT0. Bit 2 & 3 er til INT1

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

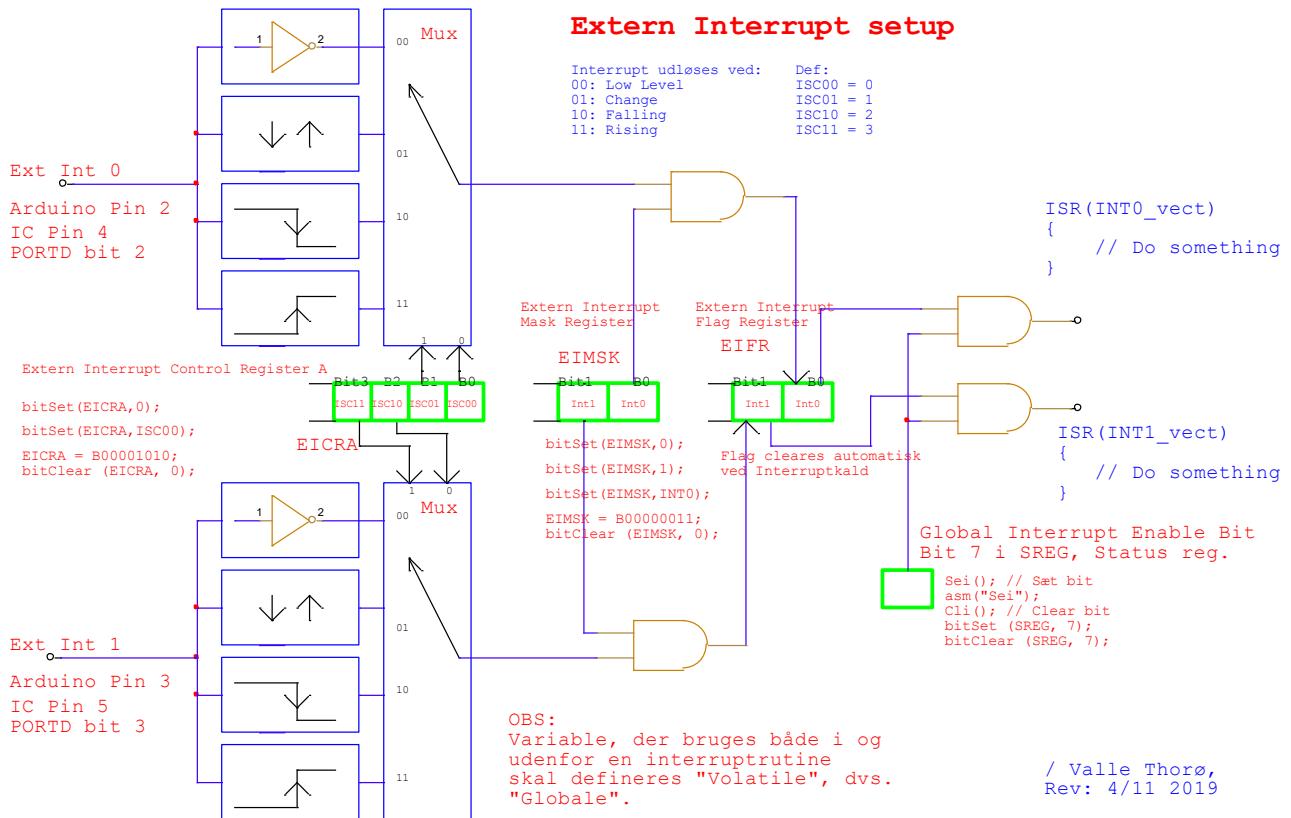
Setting the interrupt for INT0

For at tillade / enable et interrupt, skal et bit sættes i Extern Interrupt MaSK registeret

Bit	7	6	5	4	3	2	1	0	EIMSK
0x1D (0x3D)	-	-	-	-	-	-	INT1	INT0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Endelig skal en Global Interrupt-bit, bit 7 i SREG – registeret sættes !

Her en graf fra dokumentet Pin-Interrupt.



```

bitSet(EIMSK, 0); // Enable external interrupt INT0
bitSet(EICRA, 1); // Trigger INT0 on falling edge

```

Husk at beskrive, hvad der skal ske ved udløst interrupt:

```

ISR(INT0_vect) {
    EIMSK = 0; // disable external interrupts (only need one to wake up)
}

```

Hvordan laves Sleep i mere end 8 sekunder ??

PinInterrupt:

Kan laves med CMOS-timer, der kan trigge en interruptpin !

WatchDog:

Det er ikke muligt at få WD-Timeren til at sove mere end 8 Sekunder. Men man kan bede den om at sove et antal gange a'8 sekunder.

Her er vist 2 metoder



Med Variabel:

Ideen er, at en variabel tælles 1 op i Interrupt Service Rutinen. I Loop() tjekkes variablen for hver Wake. Er ønsket antal 8 sekunder ikke nået, gentages Sleep umiddelbart.

Er ønsket antal x 8 sekunder gået, udføres Stuf, og counteren nulstilles.

```
ISR(WDT_vect) {  
    counter++; // for hver 8 sek øges counteren !  
}  
}
```

I en For-Loop:

Her er Sleep-sekvensen pakket ind i en For-loop:

```
for (int i = 0; i < 2; i++) {  
    sei(); //ensure interrupts enabled so we can wake up again  
    asm volatile ("sleep"); // Go to Sleep  
}  
bitClear(SMCR, 0); // Disable Sleep
```

AD-Converteren

For at spare mest på energi, kan AD-Converteren disables før sleep.
Det sker i SFR'en ADCSRA, (ADC Control and Status Register A)

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

Det er bit 7, " AD Enable ", der disabler AD-Converteren

For ikke at ændre på de øvrige bit i registeret, kan det gøres som følgende:

```
byte adcsra = ADCSRA; // save the ADC Control and Status Register A  
ADCSRA = 0; // disable the ADC  
  
// og efter wake up:
```



```
ADCSRA = adcsra; // Restore ADC
```

Men hvis man ikke bruger AD-konvertering i sit program, kan man blot sætte følgende kode-linje i Setup()

```
ADCSRA = 0; // disable the ADC
```

Brown Out Detector

En Brown-Out Detektor overvåger processorens forsyningsspænding.

Hvis forsyningsspændingen bliver for lav, fx fordi batteriet i batteri-drevet udstyr er ved at være afladt, kan det ske, at processoren laver mærkelige ting. I nogle projekter er dette måske ikke et problem.

En Brown-Out-detector sammenligner powersupply'en med en fast intern spænding, og hvis den kommer under en grænse, reagerer BOD'en. (Hvad sker der egentlig?)

I "picoPower"-udgaverne af ATMEGA328P, ("P"et – står for picoPower), kan Brown-Out-Detektoren slås fra. Dette sparar lidt mere energi.

Opsætning af BOD

Brown-Out detectoren styres i MCUCR-registeret, (MCU Control Register).

Her kan bits ikke bare sættes. Det er af sikkerhedshensyn lavet sådan, at først skal bit BODSE sættes, og derefter inden for 4 clockcykler skal der sættes andre bits.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	-	BODS	BODSE	PUD	-	-	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

BODS: BOD Sleep

BODSE: BOD Sleep Enable

Først skal begge bits sættes til 1, og umiddelbart efter sættes BOD og BODSE resettes.

Bitsene kan fx sættes som flg:

```
//disable brown-out detection while sleeping (20-25µA)
```



```
uint8_t mcucr1 = MCUCR | _BV(BODS) | _BV(BODSE);
uint8_t mcucr2 = mcucr1 & ~_BV(BODSE);
MCUCR = mcucr1;
MCUCR = mcucr2;
```

Brown-Out skal disables umiddelbart før hver Sleep ordre !!

Kodeeksempel:

```
/*-----
Sleep demo for ATmega328P.
Sleep-i Power Down Mode, Uden brug af biblioteker
Wake med WatchDog
27/8-2023, / Valle
-----*/  
  
const int LED = 13;           //LED on pin 13
const unsigned int wait = 1000; //milliseconds  
  
uint8_t counter = 0; // antal sleep a' 8 sekunder  
  
void setup(void) {
    for (byte i = 2; i < 20; i++) { // for strømbesparelse
        pinMode(i, OUTPUT);
        digitalWrite(i, LOW); // Er vist default
    }
    pinMode(LED, OUTPUT); //make the led pin an output
    digitalWrite(LED, LOW); //drive it low so it doesn't source current  
  
    ADCSRA = 0; //disable the ADC, hvis den ikke skal bruges
                // eller bitClear ( ADCSRA, 7);
// Setup Watchdog Timer
    WDTCSR = 0b00011000; //0x18;      change enable and WDE - also resets
    WDTCSR = 0b00100001; //0x21; // Prescalers only - get rid of WDE og WDCE bit
    bitSet(WDTCSR, 6); // Enable interrupt mode
    bitSet(SMCR, 2); //set_sleep_mode(SLEEP_MODE_PWR_DOWN);
```



}

```
void loop(void) {  
  
    goToSleep();  
  
    if (counter >= 3) {  
        for (byte i = 0; i < 5; i++) { //flash the LED  
            digitalWrite(LED, HIGH);  
            delay(100);  
            digitalWrite(LED, LOW);  
            delay(100);  
        }  
        counter = 0; // reset Sleep-# counter  
    }  
}  
//-----  
void goToSleep(){  
    bitSet(SMCR, 0); //sleep_enable();  
    cli(); //stop interrupts to ensure the BOD timed sequence executes  
as required  
/* disable brown-out detection while sleeping (20-25µA) */  
    uint8_t mcucr1 = MCUCR | 0b01100000; // _BV(BODS) | _BV(BODSE);  
    uint8_t mcucr2 = mcucr1 & 0b11011111; // ~_BV(BODSE);  
    MCUCR = mcucr1;  
    MCUCR = mcucr2;  
    sei(); //ensure interrupts enabled so we can wake up again  
    asm volatile("sleep"); // Volatile for at "hjælpe" compileren  
//wake up here  
    bitClear(SMCR, 0); // Disable Sleep  
} // End-goToSleep  
  
//-----  
  
ISR(WDT_vect) {  
    counter++; // for hver 8 sek øges counteren !  
} // End-ISR
```



Kilder:

https://www.youtube.com/watch?v=urLSDi7SD8M&ab_channel=KevinDarrah

<https://wolles-elektronikkiste.de/en/watchdog-timer>

<https://wolles-elektronikkiste.de/en/sleep-modes-and-power-management>

Om Arduino Uno Sleep:

<https://www.hnhcart.com/blogs/microcontrollers/arduino-in-sleep-mode-to-save-power>

The Arduino-Way: Se:

<https://circuitdigest.com/microcontroller-projects/arduino-sleep-modes-and-how-to-use-them-to-reduce-power-consumption>

<https://microchipdeveloper.com/8avr:avrsleep>