Atmega328P Sleep-Mode

Links til afsnit i dokumentet:

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 processoren i **sleep-mode**, mellem aktiviteterne.

Herved kan der spares på strømmen.

Måske skal der bruges noget ekstra udstyr til processoren. Dette bør så disables eller afbrydes før sleep. Det kan fx gøres via en transistor-switch.

Drejer det sig fx om radiosenderen HC12, har den indbygget sleep-mode, som så kun bruger 22 uA.

Arduinos strømforbrug / StandAlone:

Arduino-boardet har i sig selv et strømforbrug på ca. 40 - 50 mA. Dette går til USB-IC'en på boardet, til 5-volts-regulatoren, og selvfølgelig til processoren.

Opbygger man et kredsløb med Atmega328P som Stand-Alone, - på fumlebrædt eller print – vil strømforbruget ifølge forskellige kilder være i størrelsen 15 – 20 mA. (Selv målt til 17 mA. !)

Men sætter man processoren i Sleep-Mode kan strømforbruget iflg. kilder komme ned i størrelsen 360 µA

Der er dog nogle forhold, der skal iagttages ! Herom senere !!

Pins:

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

Spændingsregulatoren.

Da der ikke findes et 5 Volts batteri, inkluderer et typisk kredsløb en spændingsregulator. [1] Normalt bruges en xx7805.

Men det viser sig desværre, at 7805 har et egetforbrug (Såkaldt Quiescent Current) på ca. 5 mA.

Og derudover har den behov for en inputspænding der er 5 Volt plus ca. 3 Volt, for at kunne opretholde en spænding på 5 Volt på udgangen. (En dropout spænding (se note [2]) på ca. 3 Volt) Dvs. det passer fint til et 9 Volts batteri.

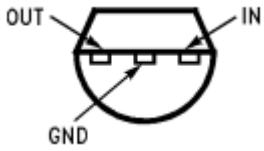
Men hvis man bygger udstyr til batteri-drift, vil det selvfølgelig betyde noget, hvis spændingsregulatoren selv bruger en ” stor ” strøm.

Det kan gøres meget bedre af en spændingsgenerator af LDO-typen, dvs. en ” Low Drop Out ”, fx LM2931Z-5.0V.

Det er ikke dens lave dropout-spænding på ca. 0,5 Volt, der betyder noget i denne sammenhæng, - hvis man da ikke forsyner den med 6 Volt.

Men den har en meget lavere egetforbrug, på ca. 0,4 mA.

Oversigt:

Type	Dropout Voltage [V]	Quiescent m [A]	Maks strøm m [A]	Hus
7805	3	ca. 5	1000	TO-220
78L05	2	2 - 5	100	TO92
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:

Batterier er jo forskellige. I spænding, men også i energiindhold.

[1] Regulator. Opfat en regulator som en enhed der lukker så mange elektroner ud på indgangen, at der opretholdes et elektrontryk – dvs. spænding, på – her - 5 Volt.
[2] DropOut-spænding er den højere spænding, regulatoren skal ha på dens indgang end output-spændingen for at kunne regulere.

Her er forsøgt en oversigt over forskellige typiske batterier.

Type	Kapacitet mAh
CR1212	18
CR1620	68
CR2032	210
Alkaline AAA	1250
Alkaline AA	2890
Alkaline 9 V	350 - 500
Li-Ion *	4400
(Cirka-værdier)	

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

Regneeksempel:

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 vist også sådan, at batterier har noget selvafladning.

Sleep Modes og Opvågning

Ved at sætte en processor i Sleepmode, sparer man jo på dens strømforbrug.

En ting er, at sætte en processor i Sleepmode. Men man må jo 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, fra dets datablad:

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/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 up ” muligheder. Altså en måde at bringe CPU-en tilbage til ”arbejde”, og fortsætte det program, den var ved at udføre lige før Sleep-instruktionen blev udført.

Power-down-mode

I det følgende er det valgt at se på den Sleep-mode der hedder **Power-down.** Det er den mode, der kan give det laveste strømforbrug.

Og der ses på to Wake-up 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). Det gør det selvfølgelig let at bruge, hvad sker der egentlig ??

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

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

Sleep-Mode SFR-Register:

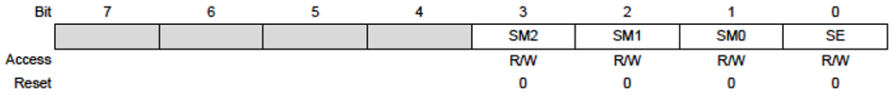
Når en MCU sættes i Sleepmode, går den i dvale. Afhængig af valgte Mode, vil forskellige af de indre enheder blive disablet, så de ikke bruger strøm.

Og man kan så få Processoren til at vågne igen på forskellige måder afhængig af valgte Sleepmode, hvorefter den umiddelbart fortsætter det program, den var i gang med.

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

Sleep Mode Control Register, SMCR

SMCR registeret:



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

Valg af sleep-mode sker ved at sætte en bitkombination i registeret.
Bit 3, 2 og 1 vælger ønsket sleepmode.

Her på skemaet er vist hvordan de 3 bit, der vælger Sleep-Mode skal sættes for at vælge ønskede mode:

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)

Her arbejdes med **Power-Down**, dvs. at der skal sættes et ”1” i det bit, der kaldes SM1, som er bit 2 i registeret.

Og også bit 0, Sleep Enable-bit skal sættes

```
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-ordre, og at det cleares igen efter Wake.

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

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

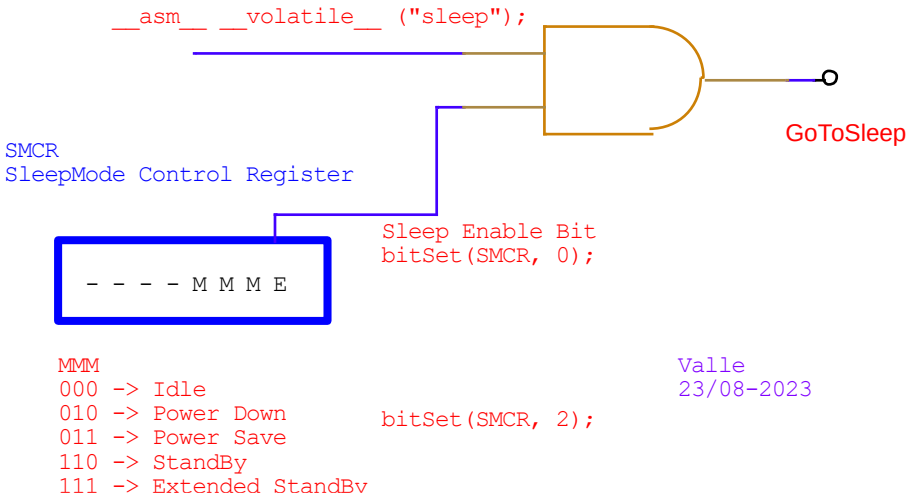
Det sker som flg:

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

Når processoren vågner, udføres automatisk først et interrupt (i dette tilfælde) -, hvorefter programmet fortsætter i næste linje efter ” Sleep-instruktionen ”.

Bemærk: Indholdet i diverse variable, dvs. RAM mm. ændres ikke under Sleep!!

På grafisk vis kan det illustreres som flg:



Ekstra enheder, der kan disables 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 !!

Wakeup fra Power-Down Mode

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

	Wake-metoder:
Den valgte wake-funktioner skal også indstilles i ” deres ” respektive SFR-register.	• External Reset • Watchdog System Reset • ***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 Watchdog timer, nogle gange kaldet ” Computer Operating Properly Timer, (COP timer) er en timer, som kan bruges til at overvåge, om CPU-en kører normalt.

I store programmer er der næsten altid fejl (Bugs). Men også hardwaren, der udfører perfekt kode, vil 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 i chippen 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.
Kilde: Se fx: [3]

Ideen med en WatchDog timer er, at en timer når at tælle op til max, og give overflow - hvis den ikke resettes med jævne mellemrum.

Overflowet kan så bringes til at starte et interrupt, eller direkte resette selve CPU-en.

I det normalt kørende program, skal koden så med jævne mellemrum resette WatchDog-timeren, 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 der udføres et cpu-reset.

Men dette WatchDog Timer-overflow kan også bruges til at få processoren til at vågne fra Sleep.

Opsætning af WDT

[3] Se fx: https://www.digikey.dk/da/articles/a-designers-guide-to-watchdog-timers

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

Timeren får tilført nogle pulser fra en ca. 128 kHz oscillator, og vha. bits kan der indstilles en frekvensdeler, - en Prescaler.

WDTCSR: WatchDog Control Register:

I registeret er der bits til at vælge en Prescaler.

- bit 5, 2, 1 & 0.

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	

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

Vha. Prescaleren indstilles, hvor hurtigt timeren giver overflow.

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 der er gået 4 clockcykler skal de øvrige bit indstilles.

Også Prescaler-bittene !

Prescaler:

Watchdog timeren får pulser fra en ca. 128 kHz fritløbende oscillator. Derfor er den ikke nøjagtig !

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

I følgende skema ses, hvordan sammenhængen er mellem de 128 kHz, frekvensdelingen (Prescaleren) og den tid, der går før overflow, - og derfor udløsning af en hændelse.

Prescaleren neddeler frekvensen fra en separat intern oscillator, der giver **ca.** 128 kHz !

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

Bit 5, 2, 1, 0

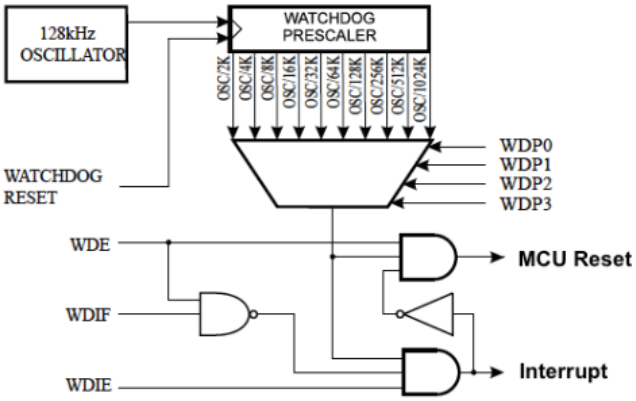
Skemaet viser, hvordan prescaler-bittene skal indstilles sammenstillet med overflow-tider.

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 !!

Systemet er illustreret i følgende skitse:

WDTCSR:



WDP0 til 3 er til at indstille prescaleren.

WDE: WD MCU- reset Enable-bit
WDIF: WD Interrupt Flag. *1)
WDIE: WD Interrupt Enable

WDCE: WD Change Enable*2)

*1, bit bliver sat 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 [4] og [5])

[4] 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).

[5] Kan også defineres (omdøbes) før setup med:
```
#define wdt_reset() __asm__ __volatile__ ("wdr")
```
Herefter kan `wdt_reset();` bruges:

WatchDog Control Register (igen):

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

WDTCSR:

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	

Men også bits til at indstille øvrige forhold:

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.

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

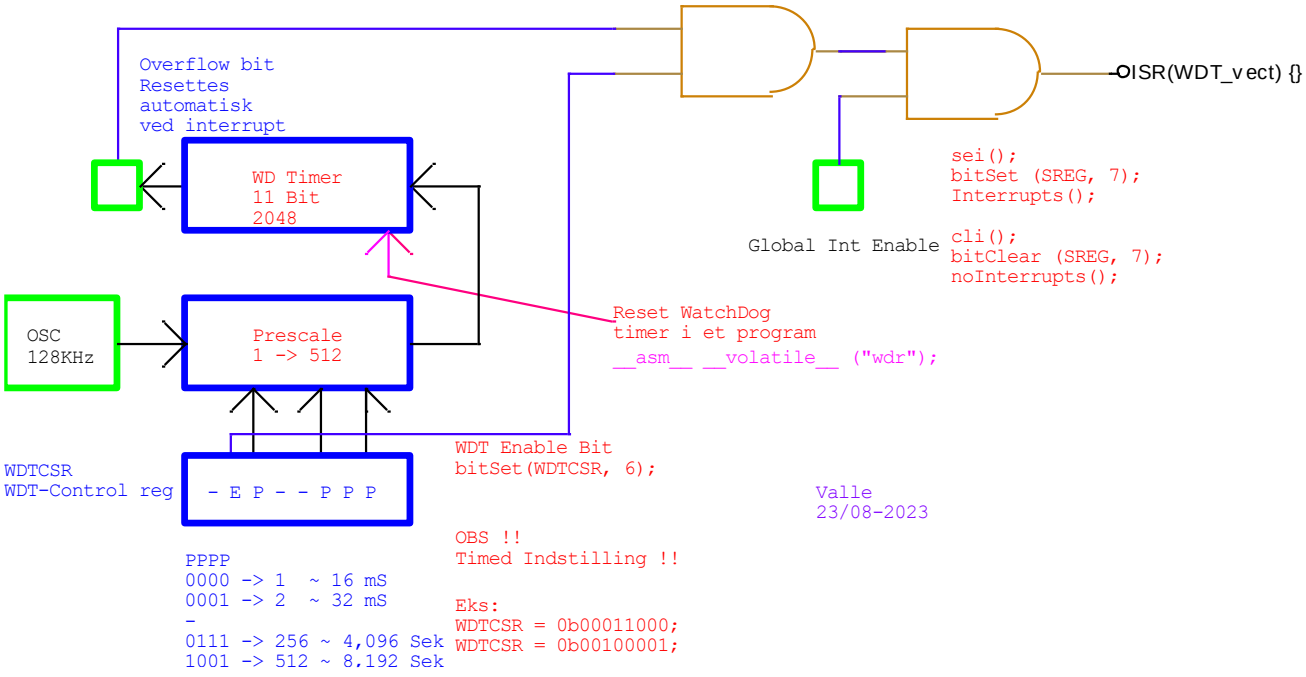
```
// Setup Watchdog Timer
WDTCSR = 0b00011000;    // 18h change enable and WDE - also resets
WDTCSR = 0b00100001;    // 21h, Prescalers only - get rid of WDE og WDCE bit
bitSet(WDTCSR, 6);      // Sæt bit WDIE, Watch Dog Interrupt Enable
```

Når der er opsat et interrupt, skal man huske den tilhørende Interrupt-Service Rutine, ISR, fx placeret efter void loop();

Der behøves ikke at ske noget i ISR-en. Blot kaldet til den vækker CPU-en.

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

Illustreret i en graf, kunne det se således ud:



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, placeret 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 wakes via trykknappen.

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

Se Fodnote: [6] 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. CMOS kræver minimalt energi.

Se tutorial om Lowpower design af Arduino: (23:13)
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	

Interrupt-type for INT0 indstilles vha. bit 0 & 1.

Og for INT1 vha. bit 2 & 3.

I dette skema er vist hvilket signal, der på pin INT0 udløser et interrupt.

Tilsvarende for 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.

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

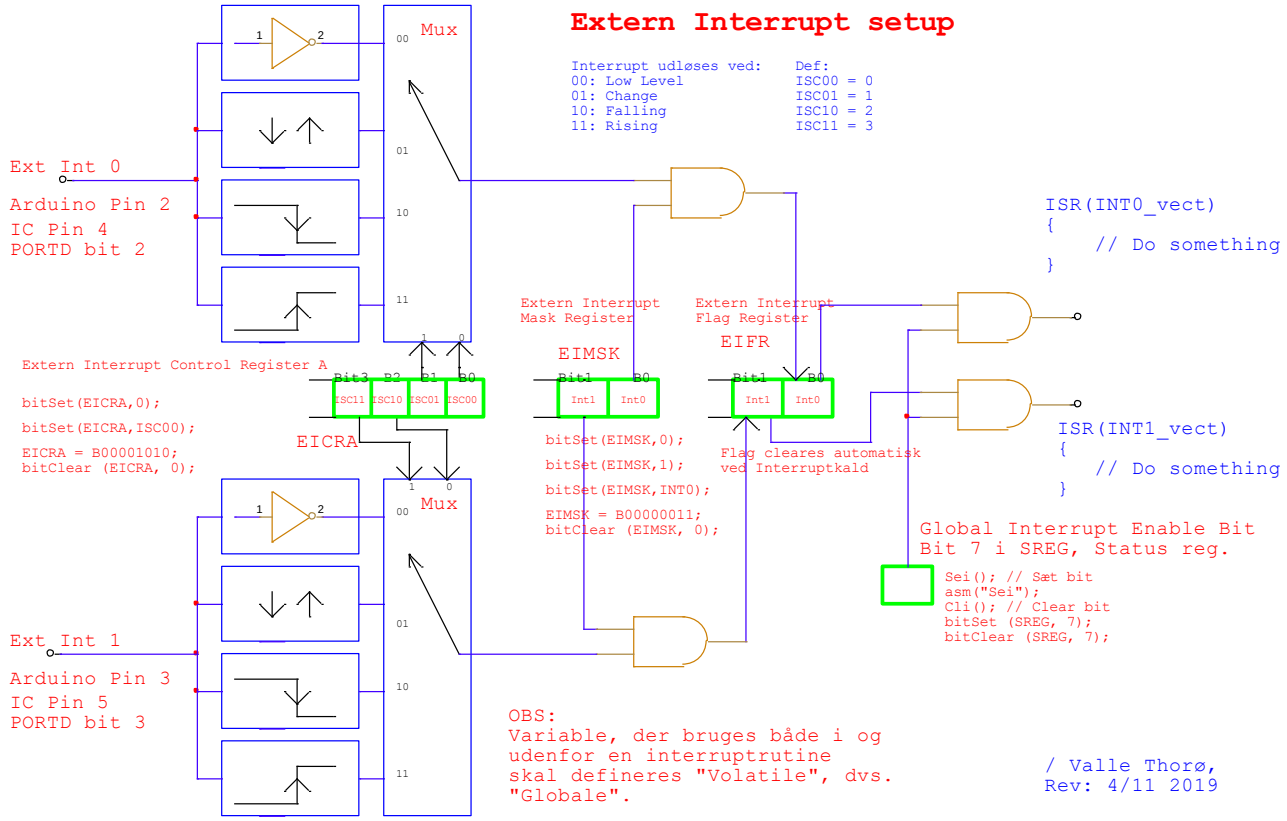
EIMSK

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

[6] 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.

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

Her er vist en graf fra et andet dokument, ” Pin-Interrupt ”.




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

Husk at beskrive i programmet, 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:

Det er muligt vha. et par IC-er at bygge en CMOS oscillator og en tæller, der med passende frekvens og antal tælle-bit, udløser et pin-interrupt med jævne mellemrum udløser et pin-interrupt.

Fx en 4093 oscillator og en 4020 eller 4040 tæller. (evt. en 4060 tæller, der har indbygget oscillator !)

WatchDog:

Det er ikke muligt at få WD-Timeren til at sove mere end ca. 8 Sekunder. Men man kan skrive et program, der får processoren til at sove et antal gange a´8 sekunder.

Her er vist 2 metoder

Med Variabel:

Ideen er, at en variabel – en counter, - tælles 1 op i Interrupt Service Rutinen. I Loop() tjekkes variablen for hver Wake, og er counteren ikke nået op på de antal gange 8 sekunder, gentages Sleep umiddelbart.

Når ønsket antal x 8 sekunder er 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 < 5; 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.

ADCSRA

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 sker i SFR'en ADCSRA, (ADC Control and Status Register A)

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 er et kredsløb i processoren, der 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'ens spænding 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 sparer lidt mere energi.

Opsætning af BOD

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

MCUCR

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	

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

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 | 0b01100000;  // _BV(BODS) | _BV(BODSE);
    uint8_t mcucr2 = mcucr1 & 0b11011111; // ~_BV(BODSE);
    MCUCR = mcucr1;
    MCUCR = mcucr2;
```

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

Kodeeksempler:

I følgende kodeeksempler er der lavet funktioner for de forskellige indstillinger, der kan vælges. Dvs. de blot skal adderes til ens projekt, - og så skal de funktioner, der ønskes brugt blot kaldes.

Der er følgende funktioner:

```
setupWatchDogTimer();           Indstil Watchdog timer
ATMegaGoToSleep();              Bed Atmega om at sove
doStuf();                       Det program, der skal køre
disableBrownOut();              Low batteri detection
disableADC();                   A/D konverteren

ISR(WDT_vect) { }               ISR, der kaldes efter wake.
```

```
/*-----------------------------------------------------------------------*
Sleep demo for ATmega328P.
Sleep-i Power Down Mode, uden brug af biblioteker
Wake med WatchDog
27/8-2023, Modificeret 9/6-2024 / 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++) {  // Def pinMode 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

  // disableADC();  // Disable ADC, hvis den ikke skal bruges
   setupWatchDogTimer();

} // EndSetup

//--------------

void loop(void) {
ATMegaGoToSleep();
// ATMega Sleeps
// return here after wakeUp

  if (counter >= 2) {
    counter = 0; // reset Sleep-# counter
    doStuf();
  }
} // EndLoop

void doStuf() {  // Example
  for (byte i = 0; i < 5; i++) {  //flash the LED
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    delay(100);
  }
}
```

```
//--------------

void ATMegaGoToSleep(){
  bitSet(SMCR, 0);  //sleep_enable();
  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

//--------------
void setupWatchDogTimer() {  // Setup Watchdog Timer @ 8 sekunder
  WDTCSR = 0b00011000;       //0x18; change enable and WDE - also resets
  WDTCSR = 0b00100001;   //0x21; Prescalers only - get rid of WDE & WDCE bit.
  bitSet(WDTCSR, 6);         // Enable interrupt mode
  bitSet(SMCR, 2);           //set_sleep_mode(SLEEP_MODE_PWR_DOWN);
}  // End setupWatchdDogTimer. Husk ISR(WDT_vect){ }

//--------------

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

//--------------

void disableBrownOut() {
  cli();  //stop interrupts, ensure 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
}
//--------------
```

```
void disableADC() {
  ADCSRA = 0;  //disable ADC, hvis den ikke skal bruges
               // eller bitClear ( ADCSRA, 7);
}  // end-disable-ADC
```

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