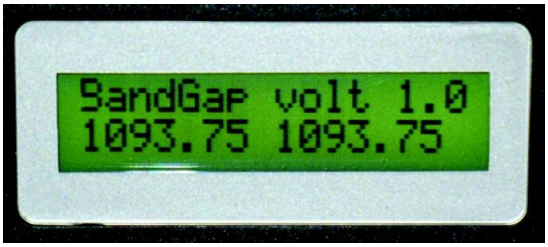
Arduino del 4

Band Gap Reference

Af OZ7EC Erik Christiansen

I forbindelse med et kommende projekt, hvor Arduino AD konverteren skal læse en ulineært voksende DC med meget stor præcision og nøjagtighed, har jeg valgt at bruge den interne Band Gap Reference på1,1 V som reference for konverteren.
Nu har den interne reference det ry blandt programmører, at den er meget støjende, så afkoblingen fra AREF til stel er usædvanlig vigtig denne gang, selvom den jo for øvrigt altid bør være der.
Selv om jeg har nærlæst AVR 351, 352 og 353 fandt jeg ingen specifikation af referencespændingens amplitude, men kun om dens temperaturstabilitet. Derfor blev dette projekt påbegyndt. Husk at den aktuelle referencespænding til AD konverteren kan måles på AREF når den er valgt, MEN den må ikke belastes. Et højimpedanset voltmeter er nødvendigt.



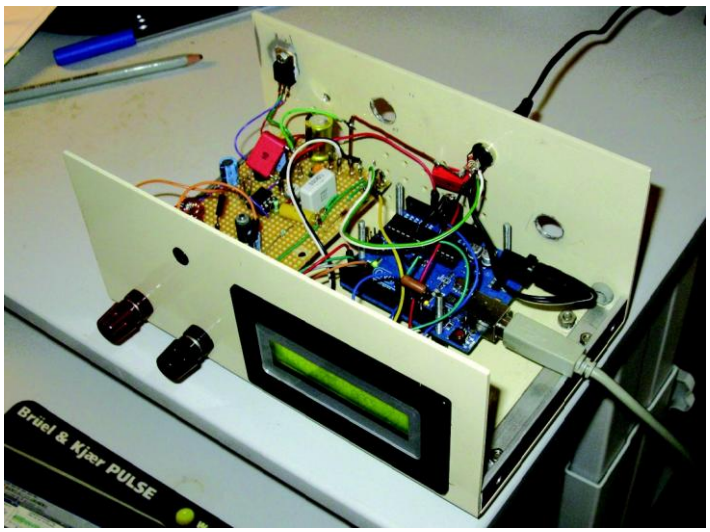
Nederst til venstre den seneste sample, nederst til højre gennemsnit af de seneste 256 samples. Der mangler en nedstreg i B'et på mit display!

Programmet "Band_gap_volt" ligger som sædvanlig hos OZ1EDR under Arduino og OZ. Vores superbe webmaster OZ5LT Tommy lægger 3 filer ud, dels en .INO (Arduino læsbart format), dels en .DOC (.DOCX) (læsbar med en tekstbehandler), samt en .PDF (Læsbar med Adobe Reader). .INO filen er umiddelbart læsbar med en Arduino software.
De to tekstfiler .DOC og .PDF henter man ind som tekst, markerer hele filen og højreklikker.

Tryk "Kopier". Nu ligger teksten på operativsystemets clipbord. Afslut din teksbehandler og start din Arduino. I en tom sketch højreklikker du og vælger "Sæt Ind". Vupti, nu er programmet inde i din Arduino. Husk lige at gemme en kopi.

Band Gap
Band Gap er en kendt metode, når man skal lave en temperaturstabil referencespænding. Du kan jo google ordene "Band Gap". Der ligger rigtig gode forklaringer på nettet. I dette tilfælde er det for så vidt uinteressant, jeg vil blot vide hvor spændingen på Band Gap i den aktuelle Arduino ligger.
Høj præcision opnår jeg ved at tage mange samples og beregne gennemsnittet. Det kræver mange konverteringer, så jeg opgav at benytte højniveausprogets "analogRead". Den var for langsom. En hastighedsforøgelse opnår jeg ved at bruge interrupt på den færdige konvertering. Desuden har jeg valgt at gå et sprog niveau nedad, ikke helt ned til assembler, men kun til chip programmering.
Oversat til dansk: Det betyder at jeg sætter de enkelte bit i min konverters registre. Det er hverken slemt, langhåret eller nørdet, det kræver kun at man kan læse databladet for ADC i ATmega328p processorens afsnit om ADC, og det starter på side 250 i databladet for Atmega328p processoren.

Hvad fik jeg så ud af det ?
Min DC reference spænding til min ADC er 4,98 V, når jeg måler på min Band Gap, og som det fremgår af billederne er min Band Gap Reference spænding 1093,75 mV. Begge er for lave, så den spænding mine programmer aflæser med ADC i den aktuelle Arduino skal korrigeres med 2 promille for 5 V, men vi skal korrigere med 5,71 promille for Band Gap referencen. 5,71 promille er da en betragtelig størrelse.



Band Gap resiederer som gæst hos ESR meteret. Bygget af rodekassen, mere i et senere nummer af OZ

Dette gælder for den aktuelle Arduino ADC, mål på din egen!
Jeg ønsker - for at optimere min præcision - at hente mange samples og derefter beregne gennemsnittet af dem. Her kører jeg med 256 samples i en FIFO buffer. FIFO = First In First Out. Den seneste sample kastes simpelthen ind forneden efter at de andre er skubbet en plads opad.
Mange samples kræver et ekstraordinært tempo på leverancen af samples, så jeg benytter interrupt fra ADC, når der er en ny sample klar. Det håndterer de 256 samples på en tilfredsstillende tid.
Hvis du vil med videre i denne tekst, må du nødvendigvis have databladet for Atmega328p liggende ved siden af. Det er en sag på godt 440 sider.
ADC er beskrevet startende på side 250, og interrupter for 328p starter på side 65. Desuden er det en fordel at du har læst Application note AVR353, "Voltage Reference" fra Atmel. Nem lille sag på 6 sider, fyldt med formler, men alligevel.
Interrupter i Arduino kræver nok en ekstra forklaring. Når ADC er færdig med en konvertering sættes en bit falsk, nemlig ADCSC, også kaldet "ADC Start Conversion". Det er den samme bit man selv sætter sand, når man vil have startet en konvertering. Når ADCSC går falsk, genererer den en hardware interrupt.
Hvis du ser på side 65 i databladet kan du se at AD Conversion Complete genererer en hardware interrupt på hex adressen 0x002A. (Vi arbejder her i assembler - det er bare for at forskrække dig). Det skal du nu ikke bekymre dig om, det håndterer din compiler for dig.
Det du skal gøre er oppe i højniveausproget at medtage en function, i dette tilfælde hedder den ISR(ADC_vect). Simpelthen: Når din ADC er færdig med en konvertering bliver den function (ISR(ADC_vect) i dit højniveausprog aktiveret. ISR står naturligvis for Interrupt Service Routine. Teknik/taktik/mange års erfaringer: ISR skal være så kort som mulig for at opnå stor hastighed.
Når du nu læser i programmet vil du opdage at jeg har valgt kun at sætte et flag sandt i interrupt rutinen. I loopen derimod ser jeg på om flaget er sat sand, og kvitterer med at bede om en ny konvertering (ADSC bitten sættes sand i konverteren). Derfor får jeg den maksimale hastighed på mine samples. Hvis du vil vide mere om Arduino interrupter henter du et dokument:

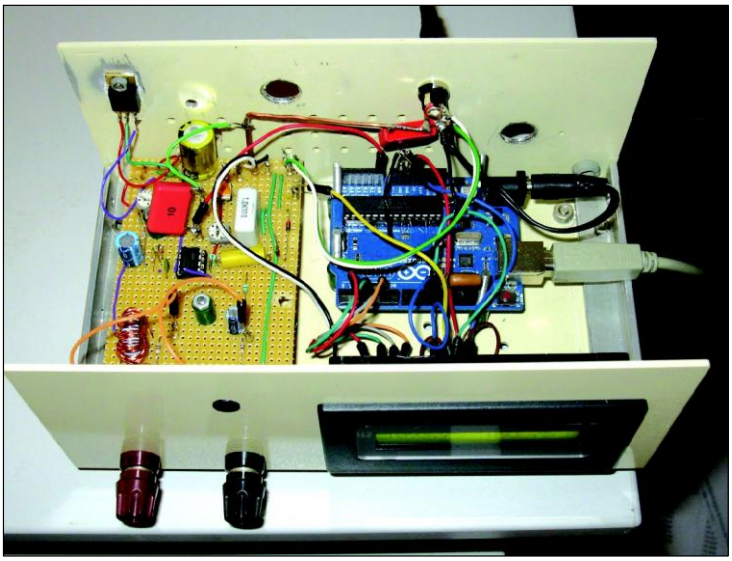
http:/www.nongnu.org/avr-libc/user-manual/group_avr_interrupts.html.

Start med småt: avr-libc er forresten et rigtig godt sted at hente Arduino informationer. Om sproget er det "Language Reference" det handler om. Det er et dokument på 912 sider, hvoraf du dog kun er interesseret i de første 550 sider.

Programmet "Band_gap_volt" har du hentet og det ligger nu ved siden af dig.
Det er, som sine forgængere og efterfølgere, opdelt i 4 sektioner: 1. Definitioner, 2. Initialisering, 3. Loop og endelig 4. Function samlingen til sidst.
Definitioner ligner denne gang alle de andre programmer, dog med én undtagelse. Jeg bruger en konstant ved navn "bufferantal". Bufferen, som indeholder rækken af samplinger, er her defineret til 256, idet bufferens størrelse går igen mange gange i det samlede program. Det er derfor nemt at ændre det antal samples du vil have i dit gennemsnit, du ændrer blot denne konstant og uploader. (Jeg har ikke prøvet med mere end 256, gør det hvis du har mod til det).

Setup starter med tekst på displayet, som sædvanlig. Men så kommer der en række anderledes statements. Vi skal have prescaleren sat til at dividere 16 MHz med128, så vi har en 125 kHz klok til ADC., og vi skal have alle 10 bits aktiveret i ADC'en.
Det gøres meget elegant med de tre statements inde i parenteserne, ADPS2, ADPS1 og ADPS0 (AD Pre Scaler X) sættes alle til 1 i ADCSRA registeret.(AD Control and Status Register A). Se ATmega328 manualen side 263.
Det næste register der skal sættes er ADMUX registeret. (AD Multiplexer). Det gør vi binært således: (side 262)
Bit 7= 0 = REFS1, Bit 6= 1 = REFS0. Denne kombination vælger ADC referencen til at være Vcc på 5 volt fra Arduinoens stabilisator.
Bit 5: ADLAR: Vend op og ned på ADC High og ADC Low dataene. Sættes på 0, vi er nemlig normale.
Bit 4: Adgang forbudt, sættes på 0.
Bit 3 til og med bit 0: Multiplexeren som vælger hvad konverteren skal have som input. Se side 263. Vi sætter bit 3, bit 2 ,bit 1 på ettaller, og bit 0 på 0. Så er input vores Band Gap Reference på 1.1 volt valgt som input.
Det var så input multiplexeren.
Nu skal vi så have gang i konverteringerne. Det først vi gør er at enable AD konverteren, det gøres elegant ved at sætte ADEN på 1 i ADCSRA registeret, der hvor vi før satte prescaleren. Bemærk den elegante måde compileren håndterer det at sætte et enkelt bit på 1.
Det samme skal vi også gøre med ADIE, (AD Interrupt Enable)
Og så skal vi lige have hele chippens interrupt system op på skinnerne, "sei" er en tilladt procedure i vores højniveausprog.
Til allersidst kommer så den vigtigste af dem alle, vi skal endnu en gang i ADCSRA registeret og have startet en konvertering (som giver interrupt når den er færdig). Det gøres ved et velrettet spark til ADCSC bitten ADSC (AD Start Conversion).
HUSK: Så længe dette bit er sandt er konverteren i gang med en konvertering. Vi spørger sidenhen på dette bits tilstand.

Loop er som sædvanlig der hvor tingene sker. Det første vi gør er at vente på at konverterens ADSC bit skifter til falsk fra sand. Mens vi venter eksekveres den næste linie i programmet, delay(x).
Her er din store chance: Hvis du vil sløve systemet ned, forøger du parameteren til delay. Du kan roligt gå op til delay(10000). Prøv det!
Næste trin: Nu skal vi lige ned i programdel 4: Functions. Der finder du en function ved navn ISR(ADC-vect). Det er , som navnet lyder, Interrupt Servide Routine. Når ADC er færdig med en konvertering, kaldes denne rutine. Som du ser er indholdet meget enkelt: sæt et flag at der er friske data at hente fra ADC. Godt råd: ISR routiner skal være korte, denne er nu ekstrem kort.
Så vender vi tilbage til loopen igen.
Hvis AD konvertern er færdig --- friskedata er sand - skal vi til at arbejde. Først kører vi "Bump". Først og fremmest sætter den "friskedata" falsk som kvittering for et sæt data fra ADC. Så tæller den "antalsamples" en op. Der-



Band Gap gæster ESR. Hulprintet til venstre er udvidelsen til at køre ESR

næst flytter den alle data i "resultat", startende med position 0, som flyttes et trin højere op. Husk at den nederste position er position 0. Derefter henter den ADC data - ordet hedder ADC. Så får vi konverterens registre High og Low rigtigt ind fra konverteren. Den værdi laver vi så om til mV og stopper den så ind i "resultat" position 0. ADC, som vi får den fra konverteren, er en samling bits, som vi ganger med værdien af LSB (Least Significant Bit), som er er 5000 mV divideret med antal bits = 1024. Sådan laves en FIFO buffer. (First In First Out)
Det sidste er en function kaldet "beregnsnit".

OZ December 2013

Det er lige bestemt hvad den gør, beregner gennemsnit af det antal samples du bad om i definitionen øverst oppe.

Litteratur:
Atmel AVR351, 352 og 353
Atmega329p datablad. (Fylder 3 ringbind her hos mig)
Om interrupt: http:/www.nongnu.org/avr-libc/user-manual/group_avr_interrupts.html.
Arduino sproget: http://arduino.cc/en/reference
Kildetekster: WWW.OZ1EDR.DK . Vælg Arduino og OZ.