Af: OZ7EC Erik Christiansen

Chiptemperatur med Arduino

Dette er en artikel i en serie, hvor jeg vil forsøge at bringe dig tættere på din mikroprocessor, uanset typen. Jeg kører med Arduino Uno, d.v.s. Atmels processor type ATmega328P. Hvis du kører en anden processor så følg bare med, forskellen er såmænd ikke særlig stor fra den ene processor til den anden.

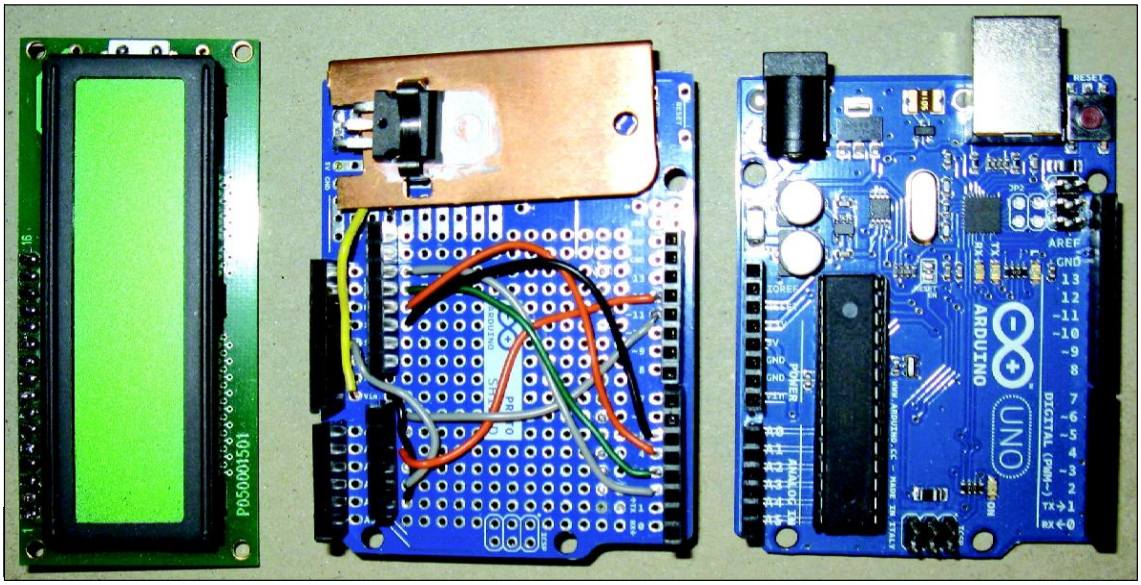
Der er i de moderne processorer rigtig mange indbyggede faciliteter, og i løbet af artikelserien kommer vi igennem de vigtigste. Mit valg af processortype faldt på Arduino, fordi projektet er på et meget attraktivt og dejligt interaktivt stade. God interaktivitet gør livet lettere for programmøren.
Har du lyst, henter du Atmels beskrivelse af ATmega328p'eren. De 450 sider fylder 3 ringbind hos mig, men er absolut godt opslagsstof, også for ikke-nørder.
Det sprog vi anvender til programmering er Arduino'sk, egentlig en afart af C++, men ikke med alle faciliteter, og med visse særheder og goder udover C++ faciliteterne.
Hvis du aktivt vil være med i denne artikelserie, så downloader du Arduinos programmeringsmiljø, anskaffer dig et Arduino Uno processorboard og et display, samt et tomt "Shield" med forlængerpinde til. Det koster et par hundrede kroner.

Dit shield bruger du til at lave en permanent interface mellem processorprint og display. Dit shield kan også samtidig indeholde en 5 V stabilisator til lys i dit display. Se teksterne i starten af programmerne, hvordan du tråder dit Shield og monterer din regulator til lyset.

Strømforsyningen til Arduino Uno R3 kan være en af to metoder:
Arduino Uno display uden lys: Printet forsynes via USB tilslutningen til din PC.
Arduino Uno display MED lys: Hent en af de sorte AC/DC spændingsforsyninger i skuffen af typen med PLUS i midten af stikket. DC spændingen skal være helst med mere end 7 V og mindre end 12 V. Lyset bruger ca. 150 mA, så hvis din regulator til 5 V skal klare det fra 12 V, skal den have en pæn stor køleplade.

Lys er absolut et gode for kontrasten på dit display.

Selve compileren i Arduino miljøet har et godt stort ordforråd men mange specialtilfælde stiller større krav om andre funktioner. Til det formål findes der utallige "Libraries", som ikke er med i standard installationen. Dem finder du nemt ved at Google, eller endnu bedre på "Arduino Playground".



Figur 1. De tre dele: Dispaly til venstre, Shield som er interface mellem Arduino Uno R3 og display med 5V regulator samt Arduino'en til højre.

Her er også et væld af mere eller mindre færdige løsninger, så næsten uanset hvad du vil lave program til, findes der en der allerede har forsøgt sig på det område der er aktuelt for dig.

Programmerne til mine Arduino artikler henter du på hjemmesiden hos OZ1EDR,:

www.OZ1EDR.dk.

Vores dygtige Webmaster OZ5LT samler alle Arduino kildetekster fra Hillerød og Birkerød afdelinger. Samme sted ligger der for resten også en række undervisnings bilag om C++ og om Arduino kurser afholdt af OZ1KQ Knud Krogsgaard Jensen i EDR Hillerød afdeling. De er absolut læseværdige.
Programmerne ligger som kildetekster i formatet .DOCX. Du henter programmet fra hjemmesiden med en tekstbehandler - Office pakken - Open Office - Libre Office, de kan alle læse .DOCX formatet.
Derefter markerer du hele teksten, højreklikker på teksten og trykker "Kopier".
Nu ligger din kildetekst på systemets clipboard.

Du afslutter din tekstbehandler og starter din Arduino.
I en tom sketch højreklikker du og vælger "Sæt Ind". Nu har du kildeteksten placeret i din Arduino, så en "Save" er nok tilrådelig inden du arbejder videre. Proceduren er lidt kompliceret første gang, men har den fordel, at kildeteksten ikke forvanskes af at være en .INI fil til download.

Hvis du ikke er med på Arduino hardwaren kan du jo godt læse de kildetekster du henter hos OZ1EDR og følge de gennemgange der kommer her i artikelserien i OZ.

Programgennemgang.
Jeg forudsætter fra nu af, at du har kildeteksten til "Chiptemperatur" liggende ved siden af og følger med efterhånden som jeg skrider fremad i forklaringerne.

C++ programmer (som ikke er spaghettiprogrammeret) falder i 4 halvlege.

1. Definitioner
2. Setup
3. Loop (I sproget C++ hedder dette MAIN)
4. Functions.

Forrest i kildeteksten til mine programmer ligger altid en almindelig tekst, som beskriver både almindelige og særlige ting ved netop dette program.

Nu starter vi en gennemgang af programmet "Chiptemperatur".
Definitionerne starter altid med hvilke eksterne biblioteker vi skal have med. Her bruger vi LCD display, så vi medtager <LiquidCrystal.h>. Husk at sådanne sager er "Case Sensitive", som på dansk betyder: Store bogstaver skal være store og små bogstaver skal være små.

Den næste definition er kryptisk, men er en simpel omgåelse af at vi mangler og ikke mere kan finde biblioteket "WProgram.h" og at det erstattes af biblioteket "Arduino.h".

Så kommer nogle få definitioner af 2 globale konstanter i dette tilfælde. De kunne ligeså godt være indsat som normale tal længere fremme i den Function hvor de bruges, men af hensyn til programmets læsbarhed og venlighed til senere vedligeholdelse er det en kæmpe fordel at bruge ord i stedet for rene tal.
At fremstille programmer med stil i stedet for med spaghettikugler letter læsningen for andre og ikke mindst, så fremmer det programkvaliteten. For resten kører vi jo ikke alle med 2 x 16 linjers display, 4 x 20 er også et populært format.

Som et led i definitionerne bruger vi nu biblioteket LiquidCrystal til at fortælle vores program hvilke Arduino Uno pinde, der anvendes til hvad. Er du nysgerrig: find og læs dokumentationen til "LiquidCrystal" og find dokumentationen til dit display.
Slutteligt definerer vi en enkelt global variabel af type double, "temperatur". Globale variabler er sådanne, som kan benyttes i hele programmet. Alternativet er en variabel, som defineres inde i en Function. Den kan så kun benyttes inde i den Function hvor den er defineret.

Det var definitionerne, så starter 2. halvleg som hedder Setup.
Først skal vi have skrevet en velkomst på vores display med en lokal Function, som ubetinget skal have navnet "Setup". Ordet Void betyder kun at parameteren er tom og det samme betyder de to parenteser efter Function navnet. Vil du vide mere, dykker du ned i C++ lærebogen.

"Lcd.begin" fortæller biblioteket hvilken type display du har sat på. Her kører jeg med et meget enkelt, 2 linjer med hver 16 karakterer. Har du lyst til større display, flere linjer og/eller flere karakterer retter du blot til oppe under konstant definitionerne.

"Lcd.setCursor(0,0);" Peg på den første position du vil til at skrive på, her den først i øverste linie.

"Lcd.print("Chip Temperatur")"; En almindelig ASCII streng på øverste linje
"Lcd.SetCursor (0,1)"; Peg på første position i nederste linje.
"Lcd.print(" Grader C"); : Igen en hel ASCII streng.

Som du ser: Hver gang du skriver på displayet starter din skrivning der hvor cursoren står.

Taktisk ved vi, at når vi befinder os i "Setup", så aner vi ikke hvad der tidligere er skrevet på displayet. Det er derfor tilrådeligt at skrive noget kendt i alle positioner og alle linjer.

Det var "Setup", så starter vi på 3. halvleg "Loop".
Her residerer den overordnede styring af program, afviklingen og aktivering af vores senere liggende arbejdsslaver i 4. halvleg, de kaldes jo for vores "Functions". Bemærk at "Loop" kører uendeligt, medmindre du har programmeret et stop kriterium inde i loopen. Det har jeg ikke her.
Void betyder som sædvanlig en tom parameter, se nærmere i C++ lærebogen. Her i "Loop" ved programudvikling er det meget fornuftigt at starte med at skrive en pseudokode, som i ganske almindeligt sprog beskriver hvilke dele mit programforløb kan opdeles i. Når du så kompilerer får du en masse fejl. Dernæst programmerer du så detaljerne i dine Functions indtil alle fejl er væk. Hvis du er til spaghettiprogrammering mikser du en klump, som du ikke engang selv kan forstå i overmorgen. Hvis du vil frembringe et program der er forståeligt for dig selv og alle andre: Brug navne på dine Functions der beskriver indholdet og del loop op i mange forståelige Functions.

Her har vi såmænd hele 2 "Function;" "henttemperatur", og "skrivtemperatur". Bemærk her, at de to navne ligeså godt kunne være pseudokode tekst. Nu er de så forfremmet til Function.
Sidst kommer der så en ordre: "delay" med mikrosekunderne som kaldeparameter i parentesen.
"delay" er med for at undgå flicker (vaklende blink) på displayet, den kunne let erstattes af en form for midling af de indkommende data.

Det var slut på Loop, nu kommer vi til 4. halvleg: "Functions", hvor vi udfører slavearbejdet.
I dette program har vi kun to Functions, nemlig "henttemperatur" og "skrivtemperatur".
Vores Function "henttemperatur" afvikles således:

Void betyder tom plads, de to parenteser efter navnet er mere parameterplads. Læs i C++ bogen.

"henttemperatur" benytter en meget speciel facilitet i vores C++ compiler. Vi kan nemlig direkte programmere med anvendelse af de benævnelser på registre og deres indhold, som er beskrevet i ATmega328p processorens hardware.
Nu får du brug for beskrivelsen af ATmega328p. Det jeg her beskriver er omtalt i afsnit 23 siderne 250 til 266. En skimning tilrådes.

Vi skal lige genopfriske din erindring at lokale variable kun kan benyttes indenfor den "Function" den er defineret i. Først definerer vi en lokal variabel ved navn "beggeregistre" og af type double. Forklaringen er enkel, vi skal hente 10 bit fra konverteren med en 8 bits processor, altså bruger vi 2 gange 8 bit i forlængelse og dem henter vi ind i en tilstrækkelig stor/lang variabel. Her bruger jeg type Double.

Først skal vi have valgt klokfrekvens til A/D konverteren. Vi har ikke travlt, så en klok på 125 kHz vil være passende for at opnå at alle 10 bit er faldet til ro. Med en hurtigere klok mister vi opløsning svarende til færre bit i konverteringen. Programmeringen gør vi ved til registeret "ADCSRA" at skrive 3 bit: ADPS2, ADPS1 og ADPS0. Så får vi aktiveret vores prescaler, som dermed deler de 16 MHz fra CPU klokken ned til 125 kHz.
Du finder mere i afsnit 23.4

Dernæst skal vi have valgt A/D konverterens referencespænding og dermed dens måleområde.
De 10 bit ligger jo fra 0 V og op til konverterens referencespænding. Det gør vi ved at skrive til registeret ADMUX - A/D konverterens input multiplekser -de to bit REFS1 og REFS0 for valg af referencespænding, samt ikke mindst at skrive MUX3.
Ordren MUX3 betyder, at A/D konverterens input er den interne temperaturføler. I højniveausproget alene kan man vælge 7 andre indgange med ordren Analogread(PIN), MEN man kan ikke vælge den interne temperaturføler i højniveausproget. Det gør vi så på en anden måde.
Konverterens andre indgange kan vælges med MUX0, MUX1 og MUX2. Her bruger vi MUX3, multiplekserens ottende indgang. Som du allerede ved, så indeholder alle cpu'er en temperaturmåler. Her har vi så valgt en i samme kategori, men noget speciel i forhold til andre cpu'er.
Du finder mere i afsnittene 23.5 og 23.9.15

Figur 2. Chiptemperaturen er 35,6 grader.

Næste trin i processen er at få enablet hele A/D konverteren. Det gør vi ved til registeret ADCSRA at skrive ADEN, A/D enable. Afsnit 23.9.2
Kvikt derefter starter vi konverteringen ved til registeret ADCSRA at skrive ADCSC, A/D konverter "Start conversion".
Så kommer det sjove: Nu står vi bare og venter, fordi vi har netop sat bit ADCSC sand, og det forbliver sandt, så længe konverteringen varer. Når dette bit bliver falsk er konverteringen færdig og vi kan hente resultatet. Det gøres med en elegant While sætning.
Resultatet af konverteringen hentes som en double ved navn "beggeregistre" hvor man skubber de øverste 8 bit 8 positioner til venstre. Sådan sikrer man sig samhørende værdier af ADCH (High) og ADCL (Low)
Du skal lige bemærke at når du læser de to registre i A/D konverteren på denne måde, så er du sikker på at få de to registres indhold fra den samme A/D konvertering. Du kan sagtens læse de to registre hver for sig, men du risikere så at få ikke sammenhængende værdier, som stammer fra 2 forskellige konverteringer.
Nu kommer det morsomste: Vi skal have korrigeret "beggeregistre" for startpunkt og stejlhed, her vist som - 342,2 som start og 1,06154 som stejlhed. Se AVR 122.
Stejlheden kan du roligt betragte som fast. Derimod er startpunktet meget varierende fra chip til chip. Der omtales flere steder en tolerance fra chip til chip på 10° C.
Den nemmeste metode at løse problemet på:

1. Mål din rumtemperatur, Arduino har stået slukket en times tid.
2. Start din Arduino og aflæs den første temperaturvisning.
3. Ret startpunktet så Arduinos første aflæsning ville have svaret til din rumtemperatur.

Anden del af 4. halvleg hedder skrivtemperatur. Finesserne er allerede omtalt i "Setup", så dem repeterer vi der.

Nu er der plads til dine forbedringer eller ændringer af programmet. Det er Open Source, så nu er du på din egen boldgade og dit eget ansvar. Hvis du googler ChipTemp får du mange hits som kan tjene til din egen inspiration.

Jeg har her anvendt konverteren i single ended mode og i single conversion mode. I senere artikler kommer vi ind på interrupt styring og continous mode, samt det at vi luller processoren i søvn mens vi konverterer. Så bliver den enkelte konvertering udført i et mere støjsvagt miljø. Men det er jo ikke altid ønskeligt, specielt hvis når jeg vil oversample. Herom i senere artikler.

Litteratur:

Atmel: ATmega328p beskrivelse.
Arduino Playground, vores sandkasse for voksne som vi alle leger i.
Atmel document: "AVR 126: ADC of megaAVR in single ended mode"
Atmel document: "AVR 120; Characterization and Calibration of the ADC on an AVR"
Atmel document : "AVR122: Calibration of the AVR's internal temperature reference"
Albert van Dalen: http://www.avdweb.nl/arduino/hardware-interfacing/temperature-measurement.html
"iwi" om offset og gain :
http://netquote.it/nqmain/2011/04/arduino-nano-v3-internal-temperature-sensor/
The internal temperature sensor can be used as seed for random numbers:
http://www.arduino.cc/cgi-bin/Yabb2/YaBB.pl?num=1294239019

OZ