Laboration 5: Temperaturmätare

1 Inledning

Laborationen ska bidra till en grundläggande förståelse för:

- Hur man hanterar olika ”moder” i ett program, genom att konstruera en tillståndsmaskin.
- Hur man använder ADC (Analog-till-Digitalomvandlaren) med differentiell förstärkning.
- Hur man använder ADC-enheten för att avläsa en analog temperatursensor.
- Hur avbrott hanteras i programmeringsspråket C.
- Hur en applikation är uppbyggd av flera abstraktionsnivåer av programkod.
- Hur man utvecklar och testar ett program i små steg.

1.1 Utrustning

- temperatursensor LM 35.
- trimpotentiometer $10\ k\Omega$ (utöver den befintliga, delas ut innan/under labtillfälle)

2 Beskrivning av laboration

Den här laborationen utgör en fortsättning på programmering av inbyggda system i C. Laborationen återanvänder kod från föregående laboration, men det tillkommer även en del nya saker: tillståndsmaskin, ADC (Analog-till-Digitalomvandlaren) och avbrottshantering. Laborationen består av följande moment:

1. Skapa nytt C-projekt.
2. Drivrutin för temperatursensor – steg 1.
3. Huvudprogram
4. Drivrutin för temperatursensor – steg 2.
5. Drivrutin för temperatursensor – steg 3.
6. Inkoppling av temperatursensor.
7. Test och färdigställning av system.
8. Filtrering av temperatursensorns signal (fördjupande moment – ej obligatoriskt!)
9. Förbättring av temperaturkonvertering (fördjupande moment – ej obligatoriskt!)

2.1 Syfte och mål

I denna laboration skall ni fortsätta att arbeta med mindre kodmoduler med tydligt definierade och avgränsade ”ansvarsuppgifter”. Ni kommer att utveckla huvudprogrammet själva steg för steg. Detta underlättas av ett redan definierat (och befintligt) API för att avläsa temperatursensorn. Från början returneras endast fixerade värden. Det är meningen att ni själva ska vidareutveckla API:t så att den faktiskt returnerar värden från A/D-omvandlaren.

A/D-omvandlingen ska ske med en differentiell förstärkning (10x). Genom att förstärka signalspänningen från temperatursensorn blir det lättare att mäta variationer. Referensspänningen är 5V. Detta har dock vissa nackdelar. Detta ska ni få reflektera över. Om ni har lust kan ni åtgärda problemen som uppstår i moment 8, som är ett fördjupande moment.

Istället för att låsa upp programmet för att vänta på att A/D-omvandlingen blir klar, ska ni använda en avbrottsrutin.

Huvudprogrammet ska konstrueras som en tillståndsmaskin (se Figur 5-1). Ett system kan alltid beskrivas med ett tillståndsdiagram. Hur man sedan implementerar lösningen i form av mjukvara kan däremot variera. I denna laboration ska ni implementera en lösning som är både enkel att förstå och realisera (diskuteras på föreläsningarna).

Tips! För att säkerställa att vissa saker verkligen fungerar (exempelvis avbrottsrutinen) kan ni alltid använda pinne 13 (PC7), som ni kan sätta hög/låg alternativt toggla. Pinnen mäter ni med oscilloskop!

2.2 Examination

2.2.1 Inlämning av rapport och programkod

Instruktioner för inlämning finns beskrivet på inlämningssidan för denna laboration (på Canvas), samt i dokumentet *Allmänna instruktioner för laborationer*.

2.2.2 Praktisk och muntlig redovisning

Den praktiska delen av laborationen examineras efter att laborationens sista moment har avslutats. Följande ska redovisas:

- Uppkopplad krets. Kretsen ska vara snyggt kopplad. Dessutom ska kablarna ha korrekt och enhetlig färgkodning.
- Programkod (med god struktur och kommentarer).
- Demonstration av systemet, fullt fungerande enligt specifikation.

Ni ska även kunna redogöra för funktionalitet avseende krets och programkod.

OBS! Precis som i tidigare laborationer ska ni alltid redovisa ert bidrag i de filer som ni redigerar eller skapar. Från och med nu förutsätts att ni gör det självmant utan uppmaningar i laborationsmanualen.

3 Moment 1: Skapa nytt C-projekt

3.1 Beskrivning

Innan ni kan börja programmera måste ni göra vissa förberedelser. Till att börja med måste ni skapa ett nytt C-projekt i Atmel Studio (instruktioner finns på Canvas Därefter ska ni importera kodbibliotek som ni använde i laboration 4. Dessutom ska ni göra en liten ändring i en av dessa kodbibliotek.

3.2 Uppgifter

Uppgift 3.2.1

Skapa ett nytt C-projekt med namnet ”lab5”.

Uppgift 3.2.2

I detta projekt ska ni återanvända kod från den förra laborationen. Ni ska importera följande: **common.h** samt mapparna med **delay**, **lcd**, **numkey**, **hmi**. Mappstrukturen ska vara densamma som i föregående laboration.

Uppgift 3.2.3

I **hmi.c** finns en funktion vid namn **output_msg**. Ändra i koden så att funktionen **delay_s** endast exekveras om **delay_after_msg** är större än 0.

4 Moment 2: Drivrutin för temperatursensor – steg 1

4.1 Beskrivning

Detta moment går ut på att ni importerar API:t för temperatursensorn. Den är väldigt grundläggande och returnerar ännu inget faktiskt värde från A/D-omvandling. Detta ska ni däremot åtgärda i kommande moment. Genom att ni har tillgång till API:t blir det mycket lättare att utveckla den huvudsakliga applikationen i små steg.

4.2 Uppgifter

Uppgift 4.2.1

Ladda ner **temp.zip** från laborationens mapp på Canvas. I denna fil finns filerna ”temp.h” och ”temp.c”. Skapa en mapp i projektet och importera dessa filer till denna mapp.

Inkludera modulen i huvudprogrammet.

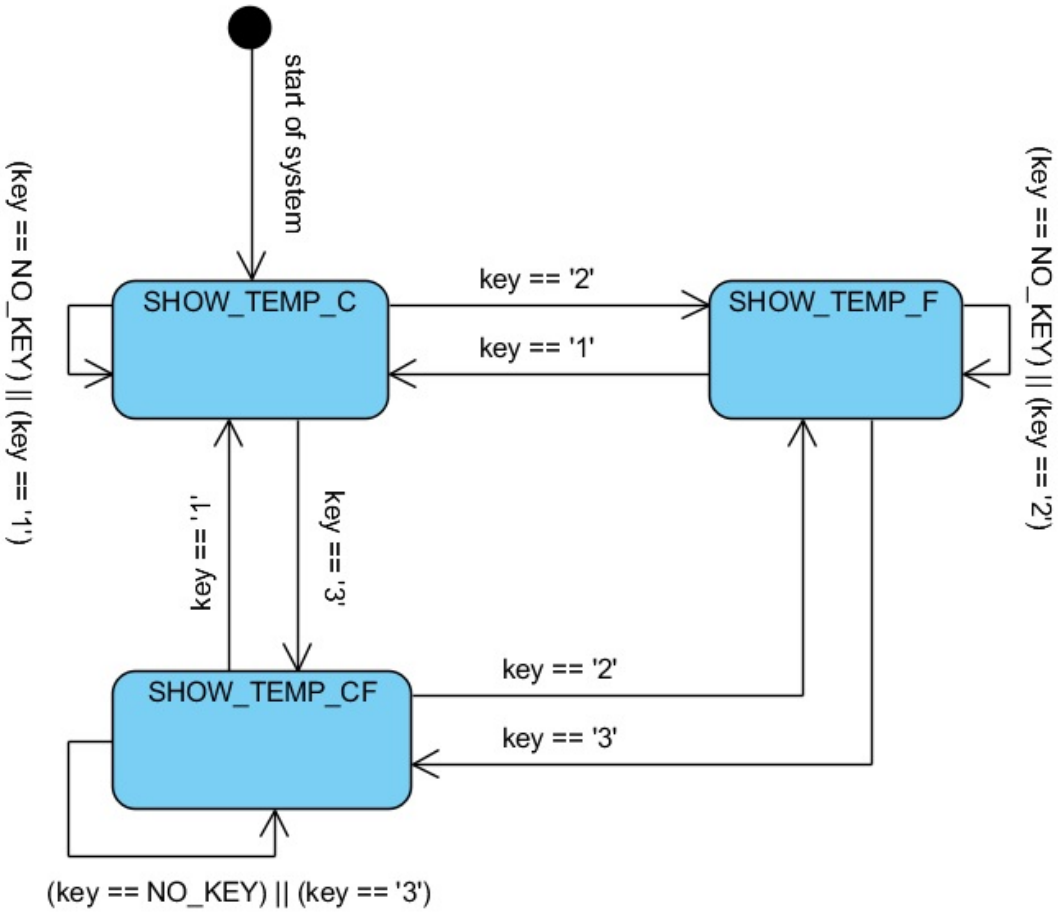
5 Moment 3: Huvudprogram

5.1 Beskrivning

Huvudprogrammet ska konstrueras som en tillståndsmaskin. Det ska finnas tre tillstånd som utför följande:

- **SHOW_TEMP_C**
  Visar temperaturen i Celsius. På första raden ska det visas ”TEMPERATURE:”. På den andra raden ska det visas ”20 °C” (om temperaturen exempelvis är 20 grader).
- **SHOW_TEMP_F**
  Visar temperaturen i Fahrenheit. På första raden ska det visas ”TEMPERATURE:”. På den andra raden ska det visas ”68 °F”.
- **SHOW_TEMP_CF**
  Visar temperaturen i både Celsius och Fahrenheit. På första raden ska det visas ”TEMPERATURE:”. På den andra raden ska det visas ”20 °C, 68 °F”.

Man växlar mellan tillstånden genom att trycka på någon av knapparna 1-3, se tillståndsdiagram (Figur 5-1). Med andra ord behöver ni använda **numkey_read()**. För att skriva ut text på displayen ska ni använda **output_msg()**. ”Grader-tecknet” har teckenkoden 0xDF.
Temperaturen läser ni via funktionerna **temp_read_celcius()** och **temp_read_fahrenheit()**.



Figur 5-1: Tillståndsdiagram för huvudprogrammet

Samtliga uppgifter i detta moment ska utföras i **lab5.c**.

5.2 Uppgifter

Uppgift 5.2.1
Börja med att se till att header-filerna **hmi.h**, **numkey.h** och **temp.h** är inkluderade och att relevanta init-rutiner anropas.

Uppgift 5.2.2
För att deklarera tillstånden på ett strukturerat sätt ska ni skapa en *enumerated type,* med namnet ”state”. I denna placerar ni namnen på tillstånden: SHOW_TEMP_C, etc.
*Tips! Om ni är osäkra på hur ni gör detta, kolla i filen **lcd.h**!*

Uppgift 5.2.3
Det som ni precis har skapat, **enum state**, ska ni nu typdefiniera till en egen datatyp. Det gör ni genom att skriva: `typedef enum state state_t;`
Nu har ni skapat en egen datatyp med namnet **state_t**.

Uppgift 5.2.4
Nu ska ni börja skriva kod i **main**-funktionen. Börja med att deklarera en variabel för att lagra tecken som läses från **numkey_read()**, kalla den för **key**.

Uppgift 5.2.5
Deklarera variabler av typen **state_t**, med namnen **current_state** och **next_state**. Dessa variabler ska initialt tilldelas värdet som motsvarar det allra första tillståndet (se tillståndsdiagram i Figur 5-1).

Uppgift 5.2.6
Nu ska ni skapa tillståndsmaskinen. Utveckla huvudprogrammet i små steg. Till en början kan ni nöja er med att skriva ut konstanta textsträngar med valfria siffror i de olika tillstånden. Vänta med att anropa temperaturfunktionerna tills att tillståndsmaskinen är komplett.
*OBS! När ni anropar **output_msg()** ska ni låta den tredje parameterna vara 0 (noll)! Det ska inte vara någon paus efter att texten har skrivit på displayen!*

Uppgift 5.2.7
Nu ska ni färdigställa respektive tillstånd. Det är ju önskvärt att skriva ut annat än konstanta textsträngar om man vill ha aktuell temperatur. Med andra ord behöver man skapa strängar som är varierande. I Java är det ganska lätt att kombinera strängar och tal, men i C är det lite jobbigare.

1. Börja med att inkludera **stdio.h** genom att skriva: `#include <stdio.h>`
2. Därefter behöver ni skapa en variabel för att lagra den ”variabla” textsträngen i. Kalla variabeln **temp_str**. Variabeln måste vara en array av typen **char**.
   OBS! Arrayen ska ha plats för 16 ”synliga” tecken! Hur många tecken ska det vara totalt?
3. I varje tillstånd ska det genereras en textsträng med funktionen **sprintf()**. I tillståndet **SHOW_TEMP_C** ska ni anropa funktionen enligt:
   `sprintf(temp_str, "%u%cC", temp_read_celsius(), 0xDF);`
4. Variabeln **temp_str** ska ni sedan skriva ut på displayens andra rad.

*OBS! Ni kommer att bli tvungna att söka information om hur man använder funktionen **sprintf()**!*

6 Moment 4: Drivrutin för temperatursensor – steg 2

6.1 Beskrivning

Temperatursensorn LM35 representerar temperaturen enligt $10\ mV/°C$. Med differentiell förstärkning 10 ggr, kommer exempelvis $30°C$ representeras av spänningen $3.00\ V$ (innan A/D-omvandlingen). Det finns däremot ett litet problem vid A/D-omvandlingen (9-bitars upplösning). Spänningen $3.00\ V$ kommer att konverteras enligt:

$$\frac{3.00\ V}{5.00\ V} \cdot 512 = 307.2 \approx 307$$

Det hade varit lite mer praktiskt om man kunde få värdet 300 (som sedan divideras med 10). Om man jämför spänningar för temperaturerna $10 - 50\ °C$ med ”önskat” och motsvarande värde efter A/D-omvandling, ser man att värdena är felaktiga med en genomsnittlig faktor 0.98, exempelvis:

$$\frac{300}{307} \approx 0.98$$

Man behöver alltså korrigera värdet från A/D-omvandlingen med denna faktor. Däremot har inte mikrokontrollern inget stöd för flyttal (decimaltal). Det går visserligen att utföra flyttalsberäkningar, men detta tar mycket längre tid jämfört med heltalsberäkningar. Lösningen blir att man ”skalar upp” faktorerna under beräkningen, för att sedan ”skala ner” resultatet.
OBS! Tänk på att eventuella decimaler ”klipps bort” vid heltalsberäkningar!

Samtliga uppgifter i detta moment utgår från **temp.c**.

6.2 Uppgifter

Uppgift 6.2.1 (redovisas i rapport)
Analysera beräkningen, steg för steg, som utförs i **temp_read_celsius().** Utgå från att **adc** har värdet 221.

Uppgift 6.2.2 (redovisas i rapport)
I **temp.c** finns en deklaration av en variabel, **adc**: (dvs. utanför funktionerna) enligt:

```
static volatile uint16_t adc = 221;
```

Vad innebär ”static”? Vad innebär ”volatile”?

Uppgift 6.2.3 (redovisas i rapport)

Analysera beräkningen, steg för steg, som utförs i **temp_read_fahrenheit().** Utgå från att **temp_read_celsius()** har värdet 27. Beräkningen utförs genom att ”skala upp” och ”skala ner” heltalsvärden istället för att använda flyttalsberäkning.

Uppgift 6.2.4 (redovisas i rapport)
Vad hade resultatet av beräkningen blivit om koden hade sett ut så här:

```
uint8_t temp_read_fahrenheit(void)
{
          return ((temp_read_celsius() * (9 / 5)) + 32);
}
```

Antag ovan att funktionen **temp_read_celsius()** returnerar värdet 27.

7 Moment 5: Drivrutin för temperatursensor – steg 3

7.1 Beskrivning

Nu är det dags att konfigurera ADC-enheten och avbrottsrutinen, så att ett avbrott genereras varje gång ADC:n är klar. Då kan resultatet från omvandlingen lagras i variabeln **adc**. Nästa gång huvudprogrammet anropar någon av temperaturfunktionerna erhålls det nya värdet.
Innan ni ansluter temperatursensorn ska ni **istället ansluta en potentiometer**. Anledningen till detta är för att ni först ska se om temperaturapplikationen kan hantera hela intervallet av möjliga spänningar. Potentiometern ska lämna ifrån sig spänningar mellan 0-0.5V, så ni blir tvungna att koppla potentiometern tillsammans med en spänningsdelare. Anledningen till att den maximala gränsen är 0.5V beror på att signalen förstärks 10ggr samt att referensspänningen är 5V.

Konfigureringen av ADC-enhet och avbrottsrutin gör ni i filen **temp.c**.

7.2 Inkoppling av potentiometern

Uppgift 7.2.1
Koppla upp spänningsdelare och potentiometern. Kontrollera att potentiometern lämnar maximalt 0.5V!

Uppgift 7.2.2 (redovisas i rapport)
Rita ett kopplingsschema som föreställer spänningsdelningen och potentiometern. Bifoga detta som bild.

Uppgift 7.2.3
Koppla potentiometerns utgång till Analog Input 4 (ADC1) och koppla 0V (GND) till Analog Input 5 (ADC0).

7.3 Konfigurering av ADC

Uppgift 7.3.1
Nu ska ni konfigurera ADC-enheten och avbrottsrutinen i funktionen **temp_init()**. Instruktioner för detta ser ni i filen.

Uppgift 7.3.2
Nu ska ni programmera avbrottsrutinen. Själva ”skelettet” finns också tillgänglig i samma fil. Det som återstår att göra ser ni i instruktionerna bland kommentarerna.

8 Moment 6: Inkoppling av temperatursensor

8.1 Beskrivning

Förutsatt att A/D-omvandlingen fungerar som den ska, så är det nu dags att ersätta testkretsen med potentiometern och spänningsdelaren. **(Ni bör behålla potentiometern, så att ni snabbt kan byta till den vid redovisningen!)** Men innan ni kopplar in temperatursensorn (LM35) så ska ni bland annat kontrollmäta för att säkerställa att sensorn verkligen fungerar som den ska. Ni får själva kolla upp i databladet hur denna ska kopplas.

8.2 Kontrollmätning

Uppgift 8.2.1

Koppla matningspänning och jord till temperatursensorn. Vänta med att ansluta signalspänningen till den analoga ingången. Mät signalspänningen med ett oscilloskop i DC-läge. Förvissa er om att den visar en rimlig spänning. Spänningen borde dessutom öka om ni värmer den genom att hålla fingrar på den.

Uppgift 8.2.2 (redovisas i rapport)

Växla över till oscilloskopskanalens AC-läge. Ändra vertikal upplösning till någonstans mellan $10 - 50\ mV$. Finns det några störningar på signalen? Hur stora är de i så fall? Ange amplitud.

Uppgift 8.2.3 (redovisas i rapport)

Fotografera oscilloskopsbilden. Bilden ska visa oscilloskopsbilden med en vettig upplösning och stabil bild (tänk på triggningen).

Uppgift 8.2.4 (redovisas i rapport)

A/D-omvandlingen sker med 10-bitars upplösning. Ange antal volt per steg som denna upplösning klarar av.

Uppgift 8.2.5 (redovisas i rapport)

Nu när ni känner till upplösningen och tar hänsyn till att även störningarna förstärks, kan detta påverka mätningarna? Ert svar ska redogöras noggrant och detaljerat!

Uppgift 8.2.6 (redovisas i rapport)

Om störningarna behöver hanteras, hur skulle man kunna göra detta på ett enkelt sätt? Ge förslag på minst en lösning. Ert svar behöver inte vara extremt detaljerad, men sammanfatta er på 3-5 rader.
Tips! Sök på Internet om ni inte kommer på något!

8.3 Inkoppling till analog ingång

Uppgift 8.3.1

Anslut temperatursensorns signalspänning till Analog Input 4 (ADC1). 0V (GND) skall fortfarande vara kopplad till Analog Input 5 (ADC0). Nu bör aktuell temperatur visas på displayen. Temperaturen bör även öka om man värmer den med fingrarna.

9 Moment 7: Test och färdigställning av system

9.1 Beskrivning

Vid det här laget bör allting fungera som den ska. Men för säkerhets skull ska ni göra slutgiltiga tester. Ni bör även snygga till koden, fixa kommentarer, etc.

9.2 Funktionstest

Uppgift 9.2.1 (redovisas i rapport)

Nu ska ni testa så att allt fungerar enligt specifikation. Skriv ett enkelt testprotokoll där ni även anger era föregående tester med potentiometern. I övrigt bestämmer ni vad som ska testas.

Uppgift 9.2.2

Utför testerna enligt ert testprotokoll.

Funkar det inte? Får ni konstiga värden? Kolla att ni gjort macrot: SET_BIT_LEVELS korrekt (Jämför med filen som finns på Canvas lab 4 och lab5)

9.3 Färdigställning av program

Uppgift 9.3.1

Snygga till koden – se till att den har en formatering som är konsekvent med majoriteten av koden. Är ni osäkra – kolla exempelvis i **lcd.c**.
Se till att kommentarerna är vettiga och ange ert bidrag i filerna. Uppdatera datum.

9.4 Slutgiltigt funktionstest

Uppgift 9.4.1

För att verkligen säkerställa att programmet fungerar efter er att ni modifierat koden en sista gång, testa systemet ytterligare en gång (enligt ert testprotokoll)!

9.5 Reflektion

Uppgift 9.5.1 (redovisas i rapport)

Redogör för era erfarenheter och kunskaper från denna laboration (minst en halv A4-sida):

- Vad har ni lärt er?
- Om ni får välja en sak, upplevde ni något som var intressant/givande?
- Fanns det något som upplevdes som svårt?
- Gick allting bra eller stötte ni på problem? Om allting gick bra, vad var i så fall anledningen detta? Om ni stötte på problem, hur löste ni i så fall dem?

Fördjupande moment

10 Moment 8: Filtrering av temperatursensorns signal

10.1 Beskrivning

I föregående moment har ni säkert kommit fram till att signalen skulle kunna vara i behov av en filtrering. Dessutom har ni kanske sett hur temperaturen ”fladdrar” lite mellan två värden.

Eftersom störningarna är högfrekventa i förhållande till signalspänningen, som är en likspänning som varierar relativt långsamt (under normala förhållanden), kan man filtrera signalen med ett lågpassfilter. Detta kan implementeras både i form av mjukvara och hårdvara. Det enklaste är att göra en mjukvaruimplementering, eftersom det inte kräver några extra komponenter.

10.2 Implementering

En enkel metod för att skapa ett mjukvarubaserat lågpassfilter är att spara några värden från A/D-omvandlingarna (3-5 värden räcker antagligen) i en array. Värdena summeras och därefter beräknar man medelvärdet, som sedan används för temperaturfunktionerna. Smidigast blir att tillämpa en cirkulär array (*circular array*).

11 Moment 9: Förbättring av temperaturkonvertering

11.1 Beskrivning

Om ni kollar lite mer noggrant på temperaturkonverteringen till Fahrenheit, så ser ni att temperaturen i Celsius redan är avrundad, vilket gör att vissa temperaturer inte blir helt korrekta.

11.2 Implementering

I funktionen **temp_read_fahrenheit()**, beräkna temperaturen i Celsius utan att anropa **temp_read_celsius()**. Tänk på att ha korrekt ”skalning” på faktorerna.