Laboration 3: A/D-omvandlaren

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-enheten för att avläsa en analog potentiometer.
- 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

- Potentiometer $10\ k\Omega$.
- Kablar/platta för inkoppling av potentiometer

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 potentiometeravläsning– steg 1.
3. Huvudprogram
4. Drivrutin för potentiometeravläsning – steg 2.
5. Test och färdigställning av system.

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 potentiometern. Från början returneras endast fixa 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 s.k. ”single ended input”, vilket innebär att bara en ingång behövs. Referensspänningen är 5V.

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).

2.2 Examination

2.2.1 Inlämning av rapport och programkod

Instruktioner för inlämning finns beskrivet på inlämningssidan för denna laboration (på It's Learning).

2.2.2 Praktisk och muntlig redovisning

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

- 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 ”Executable” C-projekt i Atmel Studio (instruktioner finns på It's Learning). Därefter ska ni importera vissa kodbibliotek som ni använde i laboration 2. 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 ”lab3”. (Välj typ = ”Executable”, INTE ASF!)

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 inparametern **delay_after_msg** är större än 0.

4 Moment 2: Drivrutin för potentiometeravläsning – steg 1

4.1 Beskrivning

Detta moment går ut på att ni importerar API:t (”halvfärdig” kod) för potentiometeravläsningen. Det 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 (med tomma funktioner) blir det mycket lättare att utveckla den huvudsakliga applikationen i små steg.

4.2 Uppgifter

Uppgift 4.2.1

Ladda ner **Lab3_filer.zip** från laborationens mapp på It's Learning. I denna fil finns filerna ”regulator.h” och ”regulator.c”. Skapa en mapp i projektet och importera dessa filer till denna mapp. Glöm inte att anpassa anropen till init-rutinerna!

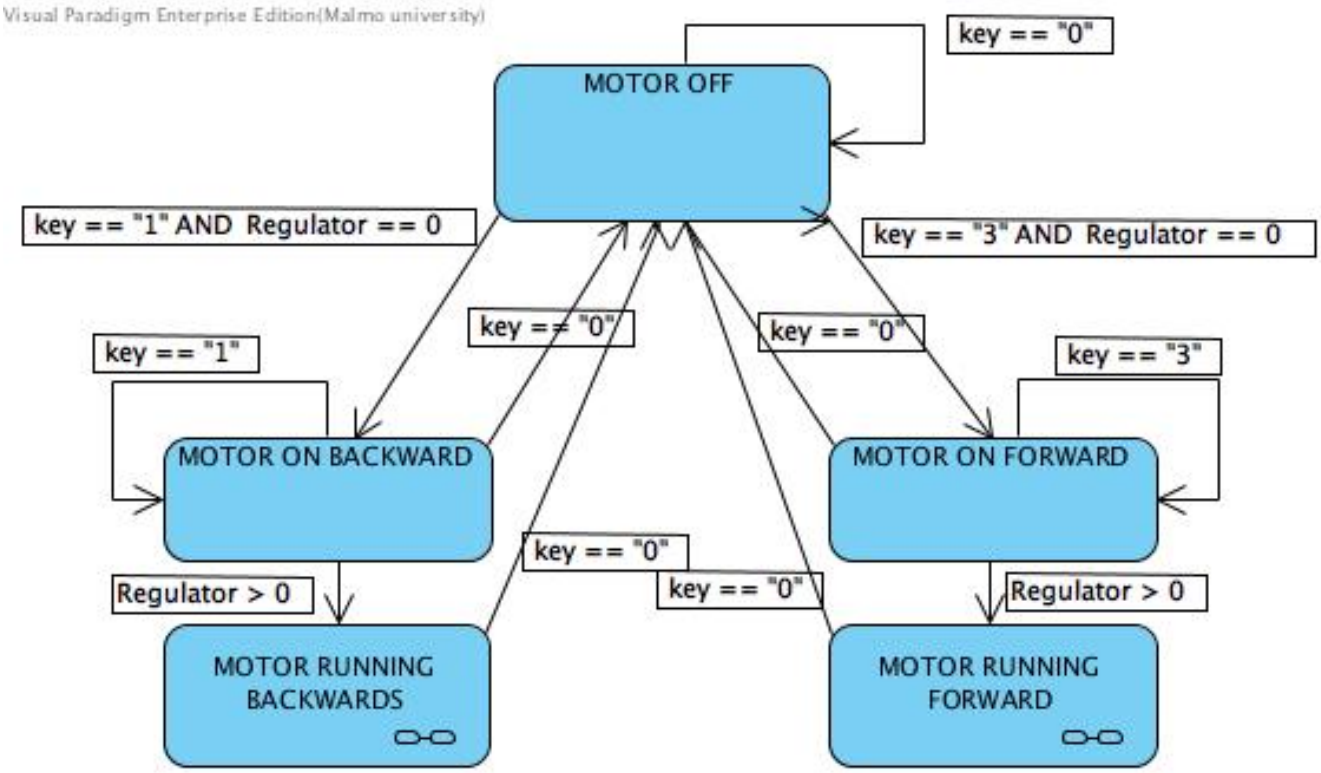
5 Moment 3: Huvudprogram

5.1 Beskrivning

Huvudprogrammet ska konstrueras som en tillståndsmaskin. Det ska finnas 5 tillstånd som utför följande:

- **MOTOR_OFF**
  Visar bara texten ”MOTOR OFF!”
- **MOTOR_ON_FORWARD**
  Visar texten ”FORWARD” på första raden.
- **MOTOR_RUNNING_FORWARD**
  Visar texten ”FORWARD” på första raden. På andra raden ska det visas ”MOTOR SPEED:” följt av ett siffervärde i procent (0-100).
- **MOTOR_ON_BACKWARD**
  Visar texten ”BACKWARD” på första raden.
- **MOTOR_RUNNING_BACKWARD**
  Visar texten ”BACKWARD” på första raden. På andra raden ska det visas ”MOTOR SPEED:” följt av ett siffervärde i procent (0-100).

Man växlar mellan tillstånden genom att trycka på någon av knapparna 1, 3 och 0, se tillståndsdiagram (se Figur 5-1) och vrida på potentiometern. Med andra ord behöver ni använda **numkey_read()**. För att skriva ut text på displayen ska ni använda **output_msg()**. ”MOTOR SPEED”-värdet läser ni via funktionen **regulator_read().**



Figur 5-1: Tillståndsdiagram för huvudprogrammet

Samtliga uppgifter i detta moment ska utföras i filen **lab3.c**.

5.2 Uppgifter

Uppgift 5.2.1

Börja med att inkludera header-filerna **hmi.h**, **numkey.h** och **temp.h**.

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: MOTOR_OFF, 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 (MOTOR_OFF, se tillståndsdiagram i Figur 5-1).

Uppgift 5.2.6

Nu ska ni skapa tillståndsmaskinen. Utveckla huvudprogrammet i små steg. Skapa variablerna (tex ”regulator”), som ges initialvärde. Till en början kan ni nöja er med att skriva ut konstanta textsträngar med valfria siffror i de olika tillstånden. Vänta med att anropa potentiometerfunktionen tills att tillståndsmaskinen är komplett.

*OBS! När ni anropar **output_msg()** ska ni låta den tredje parametern 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 aktuellt potentiometervärde. 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 **regulator_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()**. Tex i tillståndet **MOTOR_RUNNING_FORWARD** ska ni anropa funktionen enligt:
   `sprintf(str, "%u%%      ", regulator);`
4. Variabeln **regulator_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()**!*

Uppgift 5.2.8

Nu ska ni testa (delar av) tillståndsmaskinen. Variera ett defaultvärde på variabeln regulator och se att ni kan komma till tillstånden MOTOR_ON_x om värdet ==0, men inte om det är >0. Ni skall alltid kunna komma direkt till MOTOR_OFF genom att trycka ”0”.

6 Moment 5: Drivrutin för potentiometeravläsning – steg 2

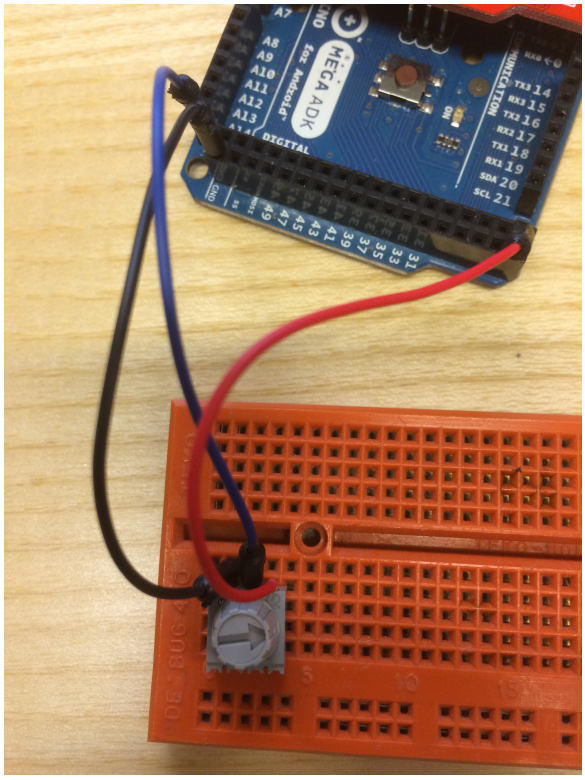
6.1 Beskrivning

Potentiometern skall läsas av med en avbrottsrutin, som ni skall börja konfigurera i detta avsnitt. Först skapar ni en variabel. Sedan 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 funktionen **regulator_read()** erhålls det nya värdet.

Nu ska ni ansluta en potentiometer. Potentiometern ska lämna ifrån sig spänningar mellan 0-5V. Anslutningen skall göras enligt bilden nedan. Koppla:

- Potentiometern har 3 ben. Placera den så att alla ben kommer i ”var sin rad” – dvs 5 hål som går från kanten in mot mitten på kopplingsplattan.
- Ena sidan av potentiometern kopplas (svart kabel i bilden) till GND (i den 2-radiga kontakten på Mega-kortet, alldeles bredvid där det står ”A15”.

- Den andra sidan av potentiometern kopplas (röd kabel i bilden) till 5V (andra änden av den 2-radiga kontakten).
- Potentiometerns mittanslutning (blå kabel i bilden) kopplas till A15.



Figur 2 Inkoppling av potentiometer

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

6.2 En variabel att skriva i avbrottsrutinen

Uppgift 6.2.1 (redovisas i programkod)

Skapa en deklaration:

```
static volatile uint8_t adc = 0;
```

Uppgift 6.2.2 (redovisas i rapport)

Beträffande deklarationen som gjordes i föregående steg: Vad innebär ”static”? Vad innebär ”volatile”?

6.3 Inkoppling av potentiometern

Uppgift 6.3.1

Koppla in potentiometern till ingång ADC15 (A15, längst ner på kortet) enligt bild och beskrivning ovan.

6.4 Konfigurering av ADC

Nu ska ni konfigurera ADC-enheten och avbrottsrutinen i funktionen **regulator_init()**.

Uppgift 6.4.1

Programmera avbrottsrutinen. Själva ”skelettet” finns också tillgänglig i samma fil. Det som återstår att göra ser ni i instruktionerna bland kommentarerna. Föreläsningsbilder och datablad för ATmega2560 kan vara bra att titta i…

7 Moment 7: Test och färdigställning av system

7.1 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.

7.2 Funktionstest

Uppgift 7.2.1 (redovisas i rapport)

Nu ska ni testa så att allt fungerar enligt specifikation. Gör ett testprotokoll med olika kombinationer av knapptryckningar och värden och verifiera dessa. Använd tex potentiometern och ställ in ett värde (som bör visas på displayen då ni är i ”rätt” tillstånd i tillståndsmaskinen). Då potentiometerns värde är >0 skall ni inte kunna byta tillstånd från MOTOR_OFF. I övrigt bestämmer ni själva vad som ska testas.

7.3 Färdigställning av program

Uppgift 7.3.1

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.

7.4 Slutgiltigt funktionstest

Uppgift 7.4.1

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

7.5 Reflektion

Uppgift 7.5.1 (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?