Laboration 1: Tangentbord och LCD i C

1 Inledning

Laborationen ska bidra till en grundläggande förståelse för:

- Utvecklingsmiljön Atmel Studio
- Hur man programmerar inbyggda system i C.
- Hur man skriver sk ”drivers” för tangentbord och LCD
- Grundläggande bitvisa operationer, med syftet att hantera in- och utenheter.
- Hur man kan skapa och använda enkla makrofunktioner.
- Hur textsträngar är uppbyggda samt hur de kan hanteras i C.
- Hur programkod kan delas upp i flera abstraktionsnivåer.

1.1 Utrustning

Alla komponenter som behövs ingår i lab-lådan.

2 Beskrivning av laboration

Den här laborationen utgör det första tillfället i kursen där ni ska programmera inbyggda system i C. Laborationen innebär att skriva drivrutiner för LCD och tangentbord. Utöver detta tillkommer en del andra saker: bitvisa operationer och hantering av pekare.

Laborationen består av följande moment:

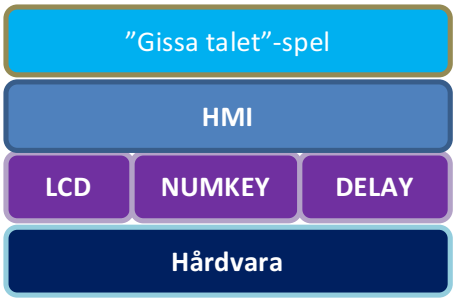
1. Introduktion till Atmel studio
2. Fördröjningsrutiner
3. Bitvisa operationer
4. Drivrutin för LCD
5. Drivrutin för numeriskt tangentbord

I laborationens mapp på It’s Learning finns ett nästintill färdigt Atmel-projekt (**lab1.zip**). Projektet består av flera filer. Laborationens olika moment går ut på att ni ska komplettera programkoden på olika abstraktionsnivåer. I nästa lab skall ni bygga på dessa delar till ett litet spel, ”Gissa talet”, som går ut på att spelaren ska försöka gissa på ett slumpmässigt tal mellan 1-100.

2.1 Syfte och mål

I denna laboration blir detta ännu tydligare med fördelarna av att skapa mindre kodmoduler med tydligt definierade och avgränsade ”ansvarsuppgifter”. Med denna metod skapar man flera

abstraktionsnivåer av programkod (se illustration i Figur 2-1).



Figur 2-1: Illustration av en applikation, baserat på flera abstraktionsnivåer av kod.

Genom att använda flera nivåer (”lager”) så blir det lättare att utveckla, underhålla och vidareutveckla applikationer. Man behöver exempelvis inte programmera om spelet bara för att man ska ”porta” det till en annan mikrokontroller. Dessutom blir det lättare att fördela uppgifterna mellan olika programmerare i ett projekt, då varje person kan fokusera på en avgränsad del av den slutgiltiga applikationen.

”Gissa talet”-spelet har avsikten att exemplifiera hur en applikation kan konstrueras på ett strukturerat vis. I takt med att ni utför laborationens uppgifter, genom att modifiera kod och göra vissa tillägg på olika abstraktionsnivåer, kommer ni att se fördelarna med att skapa avgränsade kodmoduler.

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 programkod.

3 Moment 1: Introduktion till Atmel Studio

3.1 Beskrivning

I kursen använder vi Atmel Studio IDE för programmering av de inbyggda systemen.NI bör känna igen mycket från andra IDE'er som ni använt i andra kurser (tex Eclipse).

3.2 Aktiviteter för at komma igång

Uppgift 3.2.1

På It's learning finns en beskrivning "Introduktion till Atmel Studio". Ladda ner och läs denna. Beskrivningen är gjord för Arduino Leonardo, men vi använder Arduino Mega i denna kurs. Några skillnader är:

- Processorn är en ATmega 2560
- Man behöver inte trycka på "reset" då man skall ladda ner koden till kortet. Kortet kommer hela tiden att synas som en enhet på någon port i Windows.

Prova att bygga ett projekt för C – "Executable" (låt det ligga på den plats som systemet föreslår). Projektet skall inte användas vidare, så allt ni gör i projektet kastas senare automatiskt.

Uppgift 3.2.2

På It's learning finns en zip-fil med ett färdigt projekt för lab 1. Hämta ner zip-filen, packa upp den på ett lämpligt ställe, tex i er M:-katalog (notera att ni måste köra ett skript för att logga in mot MAH innan ni kommer åt er katalog). Om ni lägger filen lokalt på datorn, så finns en risk att ni förlorar data om datorn av någon anledning startar om. Datorn startar nämligen alltid från ett sparat läge.

Kompliera/bygg programmet (notera att man måste ha "huvudfilen" – Lab1.c i detta fall – i fokus då man bygger!) som det är (bör gå bra) och ladda ner till kortet (bör också gå bra – programmet kommer inte att visa något på skärmen, men man bör kunna kika under tangentbordet och se hur två lysdioder blinkar under nerladdningen). Vissa rutiner i kedjan är känsliga för långa filnamn!

För att slippa byta port i kommandoraden varje gång kod skall laddas ner till kortet, så kan man välja "External tools…", välja Arduino Mega i listan och sedan scrolla Arguments åt höger tills man ser porten tex COM20. Ändra till aktuell port (hittas om Device Manager i Windows startas).

4 Moment 2: Fördröjningsrutiner

4.1 Beskrivning

Fördröjningsrutiner är viktiga i system som inte använder s.k. timers. Fördröjningsrutiner är kod som bara skall – ta tid! Ni får koden till den första funktionen, men skall skriva kod till de övriga. Ett tips är att låta de olika rutinerna bygga på varandra (tänk dock på att ett 8-bitars tal inte kan vara större än 255!).

4.2 Kopiering av befintliga fördröjningsfunktioner

Uppgift 4.2.1 (redovisas i programkod)

I projektet finns en befintlig fil vid namn **delay.c** I denna fil ska ni implementera era fördröjningsrutiner. Alla rutinerna har en parameter, som talar om hur många mikrosekunder/millisekunder/sekunder som rutinen skall fördröja. Tänk på att inparametern har 8

bitar (största talet som kan hanteras är 255) och att era rutiner också bör använda variabler som är 8 bitar (samma begränsning!). Det är viktigt att subrutinerna har följande namn:

- **delay_1_micros**
- **delay_micros**
- **delay_ms**
- **delay_s**

Uppgift 4.2.2 (redovisas i programkod)
Avsluta redigeringen genom att:

Redogöra syftet med filen och vad den innehåller för funktionalitet i kommentarsblocket i början av filen. Glöm inte att ange era namn och datum!

5 Moment 3: Bitvisa operationer

5.1 Beskrivning

För att kunna sätta specifika nivåer på vissa pinnar, så behöver man använda bitvisa operationer. I **common.h** finns ”skelett” till några makrofunktioner, som ska användas till dessa ändamål. Faktum är att drivrutinerna för både tangentbord och LCD är beroende av dessa. För att få dessa att fungera måste ni färdigställa makrofunktionerna.

5.2 Färdigställning av makrofunktioner

Uppgift 5.2.1 (redovisas i programkod)
Ändra **SET_BIT(reg, pos)** så att en bit på en viss position (angiven av **pos**) ändras i ett register (**reg**) eller port. Biten ifråga ska bli HÖG (logisk etta).

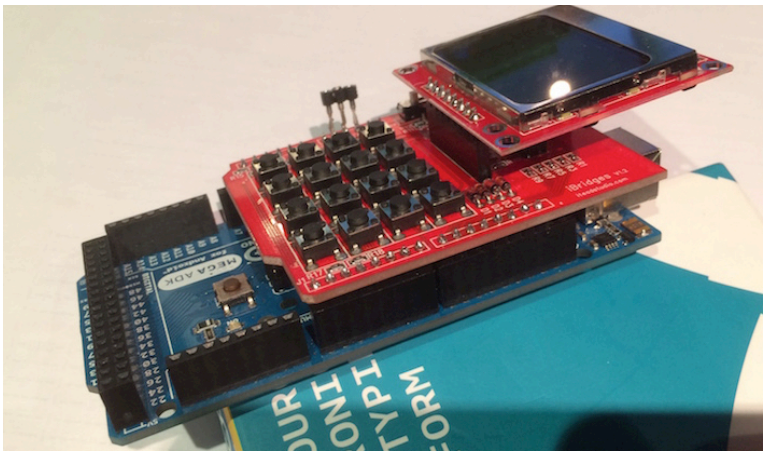
Uppgift 5.2.2 (redovisas i programkod)
Ändra **CLR_BIT(reg, pos)** så att en bit på en viss position (angiven av **pos**) ändras i ett register (**reg**) eller port. Biten ifråga ska bli LÅG (logisk nolla).

Uppgift 5.2.3 (redovisas i programkod)
Avsluta redigeringen av **common.h** genom att redogöra syftet med filen och vad den innehåller för funktionalitet i kommentarsblocket i början av filen. Glöm inte att ange era namn och datum!

6 Moment 4: Drivrutin för LCD

6.1 Uppkoppling av krets

Uppgift 6.3.1
Om det inte redan är gjort, placera LCD-displayen på Arduino Mega-kortet, enligt figuren nedan.



Figur 6-1: Placering av display på tangentbordet.

6.2 Beskrivning

Om bit-operationerna (i moment 3) gjorts på rätt sätt, så bör displayen nu kunna fungera. För att kunna skriva ut strängar och för att kunna rensa skärmen måste dock några funktioner färdigställas. Sedan är det dags att kontrollera att hela drivrutinen för displayen fungerar! Lämpligtvis gör ni tester på anvisad plats i **lab1.c**. Testerna kan bestå av att genomföra vissa operationer, exempelvis kan ni skriva ut några tecken, pausa en stund, rensa skärmen och skriva nya tecken i en oändlig loop. I takt med att ni kompletterar drivrutiner enligt nedanstående uppgifter, så bör ni också utöka era tester.

6.3 Grundläggande tester och komplettering av drivrutin

Uppgift 6.3.1 (redovisas i rapport)

Det finns en funktion som påverkar markören på displayen. Med funktionen **lcd_set_cursor_pos()** anger man var markören ska placeras. Förklara hur funktionen **lcd_set_cursor_pos()** fungerar. Ge gärna exempel på resultatet för en valfri rad och kolumn, exempelvis rad 1 (andra raden) och kolumn 2 (tredje kolumnen).

Uppgift 6.3.2 (redovisas i programkod)

Använd **lcd_write ()** för att testa att skriva ut ett tecken i taget. Skriv testkod (i huvudprogrammet), som skriver ut lite olika ord på några olika ställen på displayen (använd funktionerna som nämndes ovan).

Uppgift 6.3.3 (redovisas i programkod)

I **lcd.c** finns en funktion för att skriva ut strängar: **lcd_write_str()**. I nuvarande utförande har den dock ingen funktionalitet. Er uppgift blir att färdigställa den. Ni måste använda så kallad ”pekararitmetik” (*pointer arithmetic*) för att slutföra uppgiften. När ni är klara bör ni testa funktionen!

Uppgift 6.3.4 (redovisas i programkod)

I **lcd.c** finns även en funktion **lcd_clear ()** som också saknar innehåll. Se även till att denna fungerar som avsett (raderar skärmen helt och ställer cursorn på första positionen). Testa funktionen (tex genom att skriva ut lite text, rensa skärmen och sedan skriva lite text igen – den sista texten skall då komma längst upp till vänster på skärmen)!

Uppgift 6.3.5 (redovisas i rapport)
Beskriv hur funktionen **lcd_write_str()** fungerar, genom att referera till hur man använder pekaren för att stega igenom teckensträngen.

Uppgift 6.3.6 (redovisas i programkod)
Komplettera kommentarsblocket i början av **lcd.c** genom att ange era namn. *OBS! Det hör till god ton att låta den ursprungliga programmerarens namn vara kvar!*

7 Moment 5: Drivrutin för numeriskt tangentbord

7.1 Beskrivning

Drivrutinen är till viss del redan given, men ni måste komplettera den. När ni är klara bör ni utföra några tester. Dessa utför ni på anvisad plats i filen **lab1.c**, lämpligtvis tillsammans med drivrutinerna för displayen. På så sätt kan ni lätt få reda på om ni har tryckt på en tangent samt vilken tangent det faktiskt var.

7.2 Komplettering av drivrutin

Uppgift 7.2.1 (redovisas i programkod)
Komplettera funktionen **numkey_read()** så att den kan returnera ett tecken som representerar nertryckt tangent. Tanken är alltså att funktionen skall returnera ett ASCII-tecken, som motsvarar tangenten. Detta tecken kan då skrivas ut direkt på LCD'n.

Uppgift 7.2.2 (redovisas i programkod)
Komplettera huvudprogrammet så att alla tecken som representerar nertryckt tangent skrivs ut på LCD'n. Tänk på att hantera returvärdet "NO_KEY" (det skall förstås testas/verifieras att värdet kommer, men skall normalt inte skrivas ut…) och att man inte skall repetera samma tangent om den hålls nertryckt länge (använd INTE delay!)…..

Uppgift 7.2.3 (redovisas i programkod)
Testa att ert program, förutom att man skall kunna skriva ut alla tecken, även kan skriva ut samma tecken flera gånger (avsiktligt – om ni har problem med knappstuds, så öka delay i read_numkey) och att man skall kunna först trycka en tangent (tex 7), sedan trycka en annan (tex 3) –utan att släppa den första. Slutligen släpper man den första tangenten. Bägge siffrorna skall skrivas ut!

Uppgift 7.2.4 (redovisas i programkod)
Redovisa ert bidrag i koden genom att komplettera kommentarsblocket i början av filen. Därefter kan ni redovisa för labhandledare.

Uppgift 7.2.5 (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?