Laboration 4: ”Gissa talet”

1 Inledning

Laborationen ska bidra till en grundläggande förståelse för:

- Hur man programmerar inbyggda system i C.
- Hur man kan använda assemblykod i ett C-program.
- Grundläggande bitvisa operationer, med syftet att hantera in- och utenheter.
- Hur man kan skapa och använda enkla makrofunktioner.
- Hur man använder pekare och adressreferenser till variabler.
- Hur textsträngar är uppbyggda samt hur de kan hanteras i C.
- Hur programkod kan delas upp i flera abstraktionsnivåer.

1.1 Utrustning

Det behövs inga nya komponenter.

2 Beskrivning av laboration

Den här laborationen utgör det första tillfället där ni ska programmera i C. Laborationen består av en hel del som ni redan är bekanta med: fördröjningsrutiner, drivrutiner för LCD och tangentbord. Utöver detta tillkommer en del nya saker: bitvisa operationer och hantering av pekare.

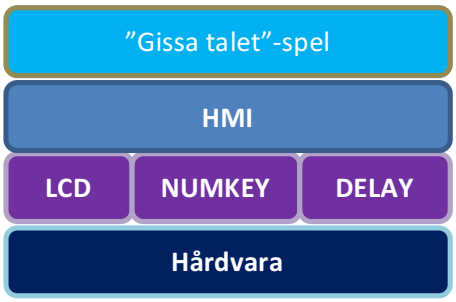
Laborationen består av följande moment:

1. Fördröjningsrutiner
2. Bitvisa operationer
3. Drivrutin för LCD
4. Drivrutin för numeriskt tangentbord
5. Interaktion med användaren
6. Färdigställning av ”Gissa talet”

I laborationens mapp på It’s Learning finns ett nästintill färdigt Atmel-projekt (**lab4.zip**). Projektet består av flera filer som tillsammans utgör grunden för enkelt spel, ”Gissa talet”, som går ut på att spelaren ska försöka gissa på ett slumpmässigt tal mellan 1-100. Laborationens olika moment går ut på att ni ska komplettera programkoden på olika abstraktionsnivåer. När ni sedan är klara kan ni ”slappna av” och testa spelet.

2.1 Syfte och mål

I föregående laboration var det en del fokus på att skapa en applikation baserat på kod med tydliga gränssnitt (API:er). 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), 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 – samma som från laboration 2. 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.

3 Moment 1: Fördröjningsrutiner

3.1 Beskrivning

Fördröjningsrutinerna som ni har skapat i tidigare laborationer ska ni nu importera till C-projektet. Istället för att skriva nya rutiner så ska ni få återanvända dessa.

3.2 Kopiering av befintliga fördröjningsfunktioner

Uppgift 3.2.1 (redovisas i programkod)

I projektet finns en befintlig fil vid namn **delay_asm.s**. I denna fil ska ni kopiera in era befintliga fördröjningsrutiner. Det är viktigt att subrutinerna har följande namn:

- **delay_1_micros**
- **delay_micros**
- **delay_ms**
- **delay_s** (om ni inte redan har skapat en subrutin för sekundlånga fördröjningar, gör det!)

Uppgift 3.2.2 (redovisas i programkod)

I **delay_asm.s** behöver ni även lägga till några direktiv som möjliggör att subrutinerna blir ”synliga” för C-kompilatorn. Detta gör att man sedan kan kalla på assembly-rutinerna från C-koden.

Uppgift 3.2.3 (redovisas i programkod)

Avsluta redigeringen av **delay_asm.s** genom att:

- Säkerställa att endast registren R18, R19 och R24 används!
- Redogöra syftet med filen och vad den innehåller för funktionalitet i kommentarsblocket i början av filen. Glöm inte att ange era namn och datum!

Uppgift 3.2.4 (redovisas i rapport)

R24 används som ”in-parameter”. Men vilka övriga register kan användas för samma syfte? Varför kan man i detta fall inte använda något annat register istället för R24?

4 Moment 2: Bitvisa operationer

4.1 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.

4.2 Färdigställning av makrofunktioner

Uppgift 4.2.1 (redovisas i programkod)

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

Uppgift 4.2.2 (redovisas i programkod)

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

Uppgift 4.2.3 (redovisas i programkod)
Ändra **SET_BIT_LEVELS(reg, bit_mask, bit_data)** så att den fungerar enligt beskrivningen i kommentarerna i källkodsfilen.

Det är väldigt viktigt att denna macro fungerar korrekt! Inte så mycket i denna labb, men i lab 5 och lab 6 är det kritiskt att det blir rätt! Jämför er implementation med den fil som finns på itslearning.

Uppgift 4.2.4 (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!

5 Moment 3: Drivrutin för LCD

5.1 Beskrivning

Nu är det dags att kontrollera att drivrutinen för displayen fungerar! Lämpligtvis gör ni tester på anvisad plats i **lab4.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.

5.2 Grundläggande tester och komplettering av drivrutin

Uppgift 5.2.1 (redovisas i programkod)
I **lcd.h** finns något som deklareras som ”`enum lcd_cursor`”. Dessa värden används av funktionen **lcd_set_cursor_mode()**. I denna funktion används variabeln **mode** i en bitvis operation, vilket i sin tur utgör en instruktion till displayen. Denna instruktion anger hur markören ska se ut. Er uppgift blir att modifiera ”`enum lcd_cursor`” så att markören inte bara kan vara osynlig, men även synlig och blinkande.

Uppgift 5.2.2 (redovisas i rapport)
Det finns även en annan funktion som påverkar markören på displayen. Med funktionen **lcd_set_cursor_pos()** anger man var markören ska placeras. Med en kombination av bitvisa operationer skapar man instruktionen som skickas till displayen. Förklara hur de bitvisa operationerna i funktionen **lcd_set_cursor_pos()** fungerar. Ge gärna exempel på resultatet av bitoperationerna för en valfri rad och kolumn, exempelvis rad 1 (andra raden) och kolumn 2 (tredje kolumnen).

Uppgift 5.2.3 (redovisas i programkod)
I den förra laborationen fick ni bekanta er med att skriva ut hela strängar av tecken, istället för att skriva ett tecken i taget. 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 5.2.4 (redovisas i rapport)
Beskriv hur funktionen **lcd_write_str()** fungerar, genom att referera till hur man använder pekaren för att stega igenom teckensträngen.

Uppgift 5.2.5 (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!*

6 Moment 4: Drivrutin för numeriskt tangentbord

6.1 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 **lab4.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.

6.2 Komplettering av drivrutin

Uppgift 6.2.1 (redovisas i programkod)
Komplettera funktionen **numkey_read()** så att den kan returnera ett tecken som representerar nertryckt tangent.

Uppgift 6.2.2 (redovisas i programkod)
Redovisa ert bidrag i koden genom att komplettera kommentarsblocket i början av filen.

7 Moment 5: Interaktion med användaren

7.1 Beskrivning

Med hjälp av grundläggande drivrutiner för att skriva text på displayen, hantera tangenttryckningar och även skapa fördröjningar, kan man skapa ytterligare en abstraktionsnivå i programkoden. På denna nivå finns funktioner för att erbjuda interaktion med användaren. Dessa funktioner finns i filen **hmi.c**. HMI står för *Human-Machine-Interaction*, dvs. interaktion mellan människa (användaren) och maskin (ert system).
I detta moment ska ni komplettera funktionen **input_int()**, som är en funktion för att låta användaren mata in ett tresiffrigt tal. Funktionen fungerar som en ”dialogruta”, vilket kan jämföras med något ni har sett i en GUI-miljö. Detta innebär att samtidigt som ni trycker på en tangent, så ska den visas på displayen. När man trycker på någon av tangenterna 0-9 så ska dessa visas på displayen. Tangenten med fyrkant (#) används för att avsluta och bekräfta inmatat tal. Tangenten med stjärna (*) används för att radera inmatade tecken, ett i taget.

7.2 Komplettering av inmatningsfunktion

Uppgift 7.2.1 (redovisas i programkod)
Efter att en tangent har tryckts ner och dess funktion har utförts, så ska programmet ”loopa” tills att tangenten har släppts upp. Komplettera funktionen så att detta sker.

Uppgift 7.2.2 (redovisas i programkod)
Komplettera funktionen så att ett tecken kan raderas från displayen när man trycker på knappen med stjärna (*).
OBS! Ni kommer att behöva använda en viss LCD-instruktion för att stega åt vänster! Läs i databladet om detta!

Uppgift 7.2.3 (redovisas i rapport)
I början av funktionen deklareras en tecken-array (**numbers**), som används för att skapa en sträng av siffertecken. Funktionen tillåter endast att maximalt tre siffror kan matas in. Som ni ser så dimensioneras arrayen till att rymma fyra tecken. Vad används den sista positionen till?

Uppgift 7.2.4 (redovisas i programkod)
Redovisa ert bidrag i koden genom att komplettera kommentarsblocket i början av filen.

8 Moment 6: Färdigställning av ”Gissa talet”-spelet

8.1 Beskrivning
Spelet går ut på att ett tal slumpas fram, som spelaren ska försöka lista ut. Om man inte gissar rätt, så får man en ledtråd om talet var för lågt eller för högt. Spelet är nästan färdigt för att demonstreras! Det enda som kvarstår är några mindre modifieringar av två funktioner i filen **guess_nr.c**.

8.2 Färdigställning av funktioner
Uppgift 8.2.1
Ta bort alternativt kommentera ut er testkod i filen **lab4.c**. Avkommentera den del av huvudprogrammet som utgör ”Gissa talet”-spelet. Kompilera programmet. Det bör inte bli uppstå några fel eller varningar vid kompileringen.

Uppgift 8.2.2 (redovisas i programkod)
I funktionen **get_nr()**, gör så att **input_int()** anropas. Den första parametern är en textsträng, som ska vara ”Enter number:”. Den andra parametern ska vara en adressreferens till variabeln **guessed_nr**, vilket möjliggör att **input_int()** kan modifiera denna variabel. Returvärdet från **input_int()** ska lagras i variabeln **input_length**. Testkör programmet för att se att det fungerar som det ska!

Uppgift 8.2.3 (redovisas i programkod)
När spelaren har gissat rätt tal ska antalet gissningar visas på displayen. För varje gissning ska en variabel ökas med 1. I nuvarande versionen fungerar detta inte alls. För att detta ska fungera måste ni göra en mindre utökning i funktionen **playing_game()**. Testkör programmet för att se att det verkligen fungerar!

Uppgift 8.2.4 (redovisas i programkod)
När spelet fungerar som det ska så är det dags att avsluta genom att redovisa ert bidrag i koden, dvs. genom komplettering av kommentarsblocket i början av filen **guess_nr.c**. Därefter kan ni redovisa för labhandledare.

Uppgift 8.2.5 (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?