Namn:
Laborationen godkänd:

Digitala system 15 p

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet
LTH Ingenjörshögskolan vid Campus Helsingborg

Datorprojekt, del 2

Projektlaboration 2, skrivning till LCD.

Förberedelser:

Studera databladen för LCD på sid 4, 5, 7, 8, 14, 18, 19, 20 och 23. Datablad finns på kurshemsidan.

Projektets organisation:

Syftet med denna laboration är att skriva drivrutiner för LCD. När vi är klara kommer vi att ha följande tre programpaket:

- Knappavläsning (AVR-MT-128-buttons.c och AVR-MT-128-buttons.h).
- Relästyrning (AVR-MT-128-relay.h).

Nytt i denna laboration:

- **LCD (AVR-MT-128-LCD.c + headerfil)**.

Erik Larsson

Laborationsuppgifter:

För att kunna hantera LCD:n, behöver vi ett antal funktioner:

```
// Skicka en puls på E-signalen:
void E_Pulse(void);

// Testa BF (Busy Flag):
void BF_Test (void);

// Skicka ett kommando till displayen:
void LCD_WriteCommand(unsigned char command);

// Skicka ett tecken till displayen:
void LCD_WriteCharacter(unsigned char character);

// Starta upp displayen:
void LCD_Init (void);

// Skicka ett tecken till displayen till en viss plats:
void LCD_PlaceCharacter(unsigned char position, unsigned char character);

// Skriv en sträng av tecken från en viss position:
void LCD_PlaceString(unsigned char position, unsigned const char *address);

// Skriv värdet av byte läsbart decimalt:
void LCD_PlaceByte(unsigned char position, unsigned char byte);
```

Displayen är inkopplad till port C:

DB7	DB6	DB5	DB4		E	R/W	RS

Databussen, som används till att skriva till och läsa från displayen är ansluten till de fyra högsta bitarna och styrsignalerna till de tre lägsta**.**

Fördröjningar

I en del funktioner kommer vi att behöva fördröjningar av olika slag. Detta beror på att elektroniken i LCD:n inte hänger med i de höga frekvenser som processorn har. Detta beror i sin tur delvis på att man gärna vill att displayen drar så lite ström som möjligt.

I en tidigare laborationerna kunde du se ett sätt att skaffa dig en fördröjning. Dock ska man vara försiktig med att använda en for-loop eller en while-slinga, eftersom kompilatorer ibland vill optimera bort sådan kod. (Kompilatorn betraktar den som meningslös.) Den variabel som används kan deklareras som ”**volatile**” och kommer då inte att bortoptimeras av kompilatorn. Om du gör din egen fördröjning, var noga med att kontrollera vilken kod som skapas och vilken optimering du har ställt in. Du måste ju också kontrollera hur lång tid varje varv tar. I vårt program används fördröjningarna bara i programuppstarten, så det gör inget om vi snurrar runt en förfärlig massa gånger i en slinga.

Ett annat sätt att lösa detta är att använda färdigskrivna rutiner. I delay.h finns fördröjningsrutiner som garanterat fungerar under förutsättning att kompilatorn är inställd på optimering. Här finns **`_delay_us()`** som ger en viss fördröjning i mikrosekunder och **`_delay_ms()`** där man anger tiden i millisekunder. I den fil där någon av dessa används, måsta man i filhuvudet ange:

```
#define F_CPU 16000000UL
#include <util/delay.h>
```

Den första raden anger frekvensen för CPU:n till 16 MHz. Om man inte anger något här, som används frekvensen 1 MHz. I filen delay.h finns en hel del förklaringar till hur det fungerar.

Uppgift 1. E_Pulse

All skrivning till displayen görs med en skrivpuls på signalen E (enable). På sid. 19 och 20 i databladen för LCD, kan du se hur lång denna signal måste vara för att det ska fungera rätt.

Skriv en funktion, vars enda uppgift är att ge denna E-puls. Tänk på att klockfrekvensen för processorn på labkortet är 16 MHz!

Skriv koden i filen AVR-MT-128-LCD.c som finns att ladda ner från hemsidan.

Uppgift 2. BF_Test

I databladet på sidan 18 kan man se att nästan varje skrivning och läsning tar lite tid.

Vilka operationer är det som tar särskilt lång tid? .....

För att göra varje skrivning till displayen helt säker, inleder du med en läsning av en flagga i ett statusregister — **BF**, Busy Flag. Detta är enda tillfället då du behöver läsa från displayen. Läsningen styrs av signalerna R/W och E. Båda måste vara ettställda för att man ska kunna läsa ut något från kretsen.

I denna uppgift ska du göra en funktion som testar BF och inte återvänder förrän BF blivit = 0.

Funktionen ska heta **BF_Test**.

Följande ska göras:

1. Ställ om LCD-databussen till ingång.
2. Sätt RS = 0 (Instruction Register) och R/W = 1 (läsning).
3. Sätt E = 1 (Enable).
4. Läs av BF (bit 7 i PORTC) och vänta här tills den blivit = 0.
5. Sätt E = 0 (Disable).
6. Anropa E_Pulse en extra gång. 4-bitars kommunikation kräver dubbel läsning och skrivning (se sid 23 i databladen).

Studera timingdiagrammet på sidan 20 i databladen. och svara på frågorna:.

Hur lång tid måste det gå från det att man ställt in rätt värden på RS och R/W tills det att man får lägga E till en etta?

.....

Hur lång tid från detta, dröjer det innan Busyflaggans status kan ses på databussens signalledningar?

.....

Hur länge måste de rätta värdena på RS och R/W finnas kvar efter det att man åter har lagt E till en nolla?

.....

Skriv funktionen BF_Test!

Uppgift 3. LCD_WriteCommand

Denna funktion ska skicka ett kommando till displayen. Funktionen ska ha en 8-bitars parameter som utgörs av själva kommandot.

På vårt laborationskort har man sparat på pinnarna på processorn och kopplat in LCD:n så att man bara kan kommunicera med 4 bitar istället för 8. Alla signaler till och från LCD:n ryms därför i port C. Hårdvarumässigt tjänar man alltså på det men vår skrivrutin blir lite mer komplicerad.

Detta ska göras:

1. Ställ om LCD-databussen till utgång.
2. Sätt RS = 0 (Instruction Register) och R/W = 0 (skrivning).
3. Placera den översta halvan av parametern ”`command`” i den övre halvan av port C.
4. Anropa E_Pulse.
5. Placera den undre halvan av parametern ”`command`” i den övre halvan av port C.
6. Anropa E_Pulse.
7. Anropa BF_Test.

```
void LCD_WriteCommand(unsigned char c){
     ????????????????????????????????????
     ????????????????????????????????????
     E_Pulse();
     ????????????????????????????????????
     ????????????????????????????????????
     E_Pulse();
     BF_Test();
}
```

Skriv resten av funktionen LCD_WriteCommand!

Uppgift 4. LCD_WriteCharacter

Funktionen är identisk med LCD_WriteCommand med undantag av att signalen RS här ska vara =1.

Skriv funktionen!

Uppgift 5. LCD_Init

Displayen ska initieras, dvs. ställas in så att den fungerar på önskat sätt. Med utgångspunkt från flödesplanen på sid. 14 i databladen får man följande initieringsfunktion:

```
//============================================================
//=== Uppstart av LCD till 4-bitars, 2-radig, ================
//=== blinkande markör.                       ================
//============================================================
void LCD_Init(){
    DDRC = DDRC | 0b11110111;   // Databussen ut
    _delay_ms(15);

    PORTC = 0b00110000;         // 8-bit interface, R/W och RS = 0
    E_Pulse();
    _delay_ms(5);

    PORTC = 0b00110000;         // 8-bit interface
    E_Pulse();
    _delay_us(100);

    PORTC = 0b00110000;         // 8-bit interface
    E_Pulse();
    _delay_us(40);

    PORTC = 0b00100000;         // 4-bit interface
    E_Pulse();
    _delay_us(40);

    LCD_WriteCommand(0x28);     // 4-bit, 2-lines

    LCD_WriteCommand(0x01);     // Display clear

    LCD_WriteCommand(0x06);     // Cursor increment

    LCD_WriteCommand(0x0F);     // Disp on, cursor on, cursor blink
}
```

Du ser att den första delen av funktionen är lite mer primitivt uppbyggd. Det beror på att Busy-flaggan inte alltid går att testa förrän inställningarna är gjorda.

Skriv in funktionen LCD_Init och anropa den i huvudprogrammets initieringsdel.

Testa! Om allt fungerar som det ska, så kommer displayens båda rader att se lika gråa ut och en svart ruta kommer att blinka uppe till vänster.

Vi ska nu komplettera koden med några användbara funktioner:

1. En funktion som skriver ett tecken på en bestämd plats på displayen. Denna funktion ska användas då man vill skriva en speciell bokstav eller siffra på en speciell plats.
2. En funktion som skriver en textsträng på en bestämd plats. Vi kommer att använda denna funktion för att bl.a. skriva ut förklarande ledtexter.
3. En funktion som skriver det decimala värdet av ett 8-bitarstal på en bestämd plats. Det kan alltså bli upp till 3 st. siffror.

Uppgift 6. LCD_PlaceCharacter

Skriv en funktion som placerar ett tecken på displayen på en bestämd plats. Två parametrar ska funktionen ta emot:

- **position** (1:a parametern) bestämmer var på displayen tecknet ska skrivas. Vi ska alltså skicka ut kommandot "DD RAM Address Set" enligt innehållet i position. Position ska anges enligt följande:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

  Observera att numreringen inte är exakt som databladet anger (sid 5). När vår 1:a parameter, position, är över 15, så måste vi alltså korrigera den genom att lägga till ett lämpligt värde.
- **character** (2:a parametern) är tecknet som skickas. Det gör du lämpligen med LCD_WriteCharacter. Eftersom vi har ställt in displayen så att varje skrivning automatiskt ökar adressen till den plats där markören står (Cursor increment i LCD_init), så behöver vi inte positionera varje tecken.

Testa funktionen genom att skriva ut ett tecken på plats nr 16 t.ex.!

Uppgift 7. LCD_PlaceString

Funktionen ska skriva en teckensträng på en bestämd plats. Den påminner mycket om föregående funktion men med den skillnaden att adressen till en teckensträng (fält eller tabell med tecken) ska tas emot, och utskrift ska ske tills värdet 0 påträffas i strängen.

Funktionen ska ta emot två parametrar:

1. **position** bestämmer var på displayen strängen ska skrivas. Vi ska alltså skicka ut samma kommando som i uppgift 6.
2. ***address** är pekare till en sträng av konstanter.

```
void LCD_PlaceString (char position, const char *address)
{
    ........................
    ........................
    ........................
}
```

Tips:

Såhär kan en sträng definieras:

```
const char string [] = "ABC";
```
eller
```
const char string [] = {0x41, 'B', 67, 0};
```

Dessa två skrivsätt ger exakt samma resultat. Om man skriver ”ABC”, hamnar värdet noll automatiskt som en slutmarkering.

När man refererar till en sträng, kan man antingen skriva:

```
LCD_PlaceString (16, &string[0]);
```
eller enklare
```
LCD_PlaceString (16, string);
```

I båda fallen består parametern av adressen till det första värdet.
Ett tredje sätt är att skriva strängen direkt som en av parametrarna:

```
LCD_PlaceString (16, ”ABC”);
```
fast då knorrar kompilatorn!

Testa funktionen genom att skriva ut texten ”Hejsan!” på position 19.

Uppgift 8. LCD_PlaceByte

Funktionen ska skriva ut ett 8-bitarstal på en bestämd plats.

Se externdeklarationerna i laborationens inledning!

Inparametern **position** ska i likhet med funktionen LCD_PlaceCharacter bestämma var på displayen den första siffran ska skrivas.

Inparametern **byte** är det 8-bitarsvärde som ska skrivas ut.

Om byte har värdet 0xFF (255) så ska alltså de tre siffrorna 2 5 5 skrivas ut tätt efter varandra.

1. Låt funktionen först positionera markören på samma sätt som i LCD_PlaceCharacter och LCD_PlaceString.
2. Låt den sedan beräkna hundratalssiffran, tiotalssiffran och entalssiffran.
3. Skapa ASCII-koden för var och en av dessa siffror och mata ut var och en med hjälp av funktionen LCD_WriteCharacter.
4. Om hundratalssiffran är = 0, så ska den ***inte*** skrivas ut! Anledningen till detta är att funktionen ska användas till digitaluret i nästa laboration. Och det ser snyggast ut då.

Funktionens skal:

```
void LCD_PlaceByte (char position, char byte)
{
    char hundratal, tiotal, ental;
    ..........................
    ..........................
    ..........................
}
```

Testa funktionen med några olika värden.