Andreas Axelsson
Alexander Göransson
Viktor Mann

JÖNKÖPING UNIVERSITY
School of Engineering

2022-04-09

Enchipsdatorer – Laboration 4
LCD-skärm

Målsättning

- Läsa datablad
- Skriva med objekt-orienterad stil i C.
- Få dioder (inuti skärmen) att lysa.
- Programmera direkt med register.

Förberedelse

Läst på föreläsningarna som tar upp timers och LCD-skärmen.

Examination

Resultat för laborationen presenteras muntligt för en laborationshandledare under laborationstillfället. Svar på frågor i **fet stil** ska besvaras under redovisningen. Krav på kodstil måste följas.

Genomförande

Laborationen har 4 timmar handledd tid, men kan ta mer tid att genomföra, vilket då görs på egen hand.

Det är viktigt att kodningsarbetet sker individuellt. Det är okej att diskutera problemlösning med andra men det är absolut förbjudet att kopiera kod från andra. De lösningar som tas fram skall förses med bra kommentarer, korrekt indentering och bra variabelnamn. Koden ska m.a.o. vara lätt att läsa och förstå.

Hårdvara

- Kopplingskablar
- LCD-skärm (2x16 tecken) med styrkrets PCF8574 för I2C-kommunikation.

Enchipsdatorer – Laboration 4
LCD-skärm

1. C-programmering, klocka

1.1. CubeMX

Gör inställningar i CubeMX så att du kan återanvända koden för din klocka från förra laborationen. Om du vill förenkla din kod (slippa tick två gånger per sekund) så kan du sätta $f_{TIM} = 1\ Hz$.

1.2. Objekt-orienterad C

I C++ (och mängder med andra språk) så skriver man objekt-orienterat med syntaxen

```
my_obj.method(arg1, arg2);
```

I C görs nästan samma sak, men notationen ser ut så här:

```
method(&my_obj, arg1, arg2);
```

Det objekt som du vill operera på blir alltså ett argument (oftast första) och man skickar pekaren till detta objekt. För ”objekt” (i betydelsen ”samling av data”) i C används ordet `struct`. Skriv en `struct` för att hålla din klockas variabler i ditt program. Du vill ha den tidigt i ditt `main.c` eller i `main.h` så att alla funktioner kan använda sig av den.

Ordet `typedef` ska **INTE** användas här. Så här ska det se ut:

```
struct clock_data
{
    /* your clock variables goes here */
};
```

Skriv först en ”setter” för din nya `struct`:

```
void cd_set(struct   clock_data * pcd,
            uint8_t               hrs,
            uint8_t               min,
            uint8_t               sec);
```

Notera att (som med `enum`) så nämns ordet `struct` explicit. Så här skriver du kod som använder cd_set:

```
struct clock_data my_clock; // reserves the memory needed
cd_set(&my_clock, 23, 59, 45);
```

För att klockan ska ticka ordentligt när timerns callback-funktion kört så ska du implementera denna funktion. Om den ska ticka varje hel eller halv sekund beror på hur du konfigurerat din timer.

```
void cd_tick(struct clock_data * pcd);
```

För att testa att tiden tickar ordentligt så skriv en funktion som skriver ut klockan över UART.

```
void uart_print_cd(UART_HandleTypeDef * huart,
                   struct clock_data  * pcd);
```

Gör den så att den ”skriver över” föregående rad. Dvs. att den *inte* skriver ut en ny rad vid varje utskrift. För att få detta att fungera behöver du förstå skillnaden mellan `'\n'` (NL, newline) och `'\r'` (CR, carriage return). Om du använder `sprintf()` så ta reda på hur man fyller på (eng. ”padding”) med nollor vid små siffror. Om t.ex. sekundsiffran är 8 så ska den visas som `”08”`. Inte som `” 8”` eller bara `”8”`.

Kodkrav:

- Funktionerna ska ha de namn och typsignaturer som nämnts ovan.
- `cd_tick()` får ej anropas inifrån callback-funktionen.
- Varje callback ska leda till ett anrop av `cd_tick()`.

2. LCD-skärmen

2.1. Datablad och hårdvara

Skärmen som ska användas är en HD44780. Öppna dess datablad, gå till s. 8 och läs om de signaler som hör till MPU. Läs sedan s. 9-12. Du behöver inte bry dig om CGROM/CGRAM[1].

Din enhet kommer med en I2C-seriedataexpanderare fastmonterad. Istället för att du kontrollerar varje kontakt individuellt så kommer data att skickas till enheten seriellt och expanderaren lägger sedan ut de bitar som skickas parallellt. I2C är ett protokoll som på endast 2 kontakter (SCL och SDA) kan kommunicera med upp till 128[2] enheter.

Detta betyder att du inte kommer åt kontrollportarna (E, RW, RS) direkt. Hur expanderaren är ansluten till skärmen kan du se i Appendix. Kortfattat så används de

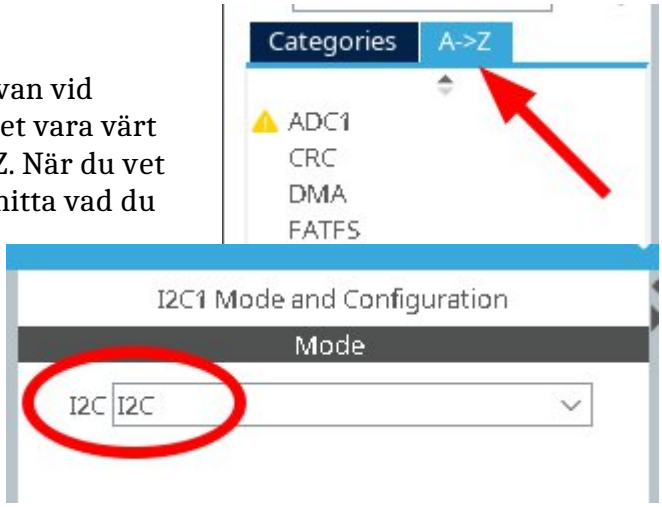
[1] Såtillvida du inte tycker att s.17 är otillräcklig och du tänker ladda över ett eget typsnitt.

[2] Egentligen något lägre p.g.a. reserverade adresser. Detaljer för I2C tas upp i kursen Elektriska Gränssnitt.

fyra översta bitarna för data och de fyra undre för kontroll. Signalerna RS, RW, E samt en bakgrundsbelysningsflagga skickas med vid varje kommunikation till enheten.

På expanderarkortet sitter en potentiometer som kan regleras med en liten skruvmejsel. Denna reglerar skärmens kontrastnivå. Det är ej ovanligt att denna måste justeras.

2.2.CubeMX



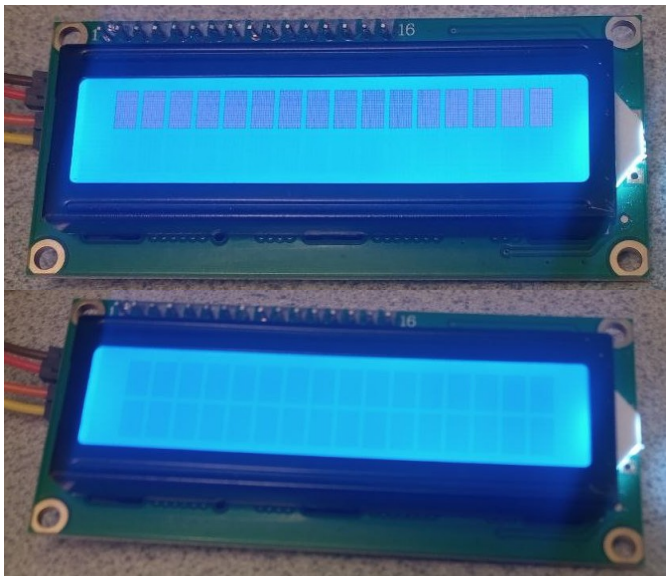
Öppna CubeMX. Nu, när du börjar bli van vid CubeMX och dess olika enheter, kan det vara värt titta på enheterna i listan sorterad A-Z. När du vet vad du vill ha så går det snabbare att hitta vad du behöver.

Aktivera enheten `I2C1` enligt bilden. Inställningarna under ”Parameter Settings” är bra, men en sak ska ändras; benen som används av MCU:n.

När en enhet som använder GPIO-portar aktiveras så sätts portarna som ”Alternate Function” (vad som sätts i `GPIOx->MODER`) av HAL. Slå upp Alternate Function-tabellen i *databladet* för din MCU. Titta i kolumnen AF04 för I2C1. Om du byter portarna SDA och SCL för I2C1 så tar du upp portar som TIM10_CH1, TIM11_CH1, SDIO_D4 och SDIO_D5 kunde använda.

Vilka andra portar kan dessa signaler använda?

Klicka nu på de alternativa benen i pinout-vyn i CubeMX och byt var I2C1 ska anslutas. Koppla nu in skärmen till ditt kort. VCC ska vara +5V.



2.3.Initiering av display

Lägg till `lcd.h/lcd.c` i ditt projekt.

Den övre bilden är före `TextLCD_Init()` är körd och den undre är efter. Fyrkanterna på övre raden försvinner alltså när funktionen körts.

Skärmen kan drivas i ett 4-bitars-läge vilket är vad `lcd.h/lcd.c` ställer in i initieringsfunktionen.

Öppna lcd.c och titta på typsignaturen för `TextLCD_SendByte()`. Utöver den byte som ska skickas (görs en nibble åt gången) så tar den `RS` som argument. Detta betyder att du har kontroll över en av kontrollsignalerna genom serieexpanderaren.

2.4.Komma igång

För att göra arbetet lättare så börja med att skriva en funktion `wait_for_button_press()`. Ditt programs körning ska stå still på denna funktion tills du tryckt *och* släppt knappen. Så att använda while-satser med tomma kodblock är mycket användbart:

```
while (!the_thing_im_waiting_for)
{
    /* do nothing */
}
```

För att testa den så kan du använda detta kodblock:

```
for (int i = 0; i < 5; i++)
{
    wait_for_button_press();
    HAL_GPIO_TogglePin(LD2_Port, LD2_Pin);
}
```

När du läst resten av problembeskrivningen på nästa sida så titta igenom koden inunder. **Börja med att fokusera på `TextLCD_PutChar()` och använd denna kodsnutt som mainloop** för att se till att funktionen gör vad den ska.

```
char c = 0;
const char ASCII_CAPITAL_OFFSET = 'A';
const char LETTERS_TOTAL = 'Z' - 'A';

while (1)
{
    wait_for_button_press();
    TextLCD_PutChar(&lcd, c + ASCII_CAPITAL_OFFSET);
    c = (c+1) % LETTERS_TOTAL;
}
```

Enchipsdatorer – Laboration 4
LCD-skärm

Slå upp databladet för HD44780 på s. 23 och börja läsa under Outline. Tabellerna som slutar på s.25 innehåller informationen som du behöver för att fylla i de tomma funktionerna i `lcd.c`. P.g.a. serieexpanderaren så kan du inte kontrollera vad som händer på BF (DB7). Detta eftersom läsning från enheten inte går att göra på ett betydelsefullt sätt. Det är därför flaggan RW alltid sätts som låg (Write).

De fördröjningar som skrivs ut i tabellerna ska vara med i dina implementationer. Fördröjningsfunktionen du ska använda är den som är längst upp i `lcd.c`. I sista delen av denna laboration ska du skriva om denna funktion för att korrekt hantera mikrosekunder.

Dina funktioner ska bete sig i enlighet med listan i appendix.

Krav på din färdiga lösning:

- Har implementationer för funktionerna i denna lista. För hjälp med `TextLCD_SetDDRAMAdr()`, och `TextLCD_SetPos()`, se s. 21 i databladet.
  - `TextLCD_Clear      (TextLCDType * hlcd)`
  - `TextLCD_Home       (TextLCDType * hlcd)`
  - `TextLCD_SetDDRAMAdr(TextLCDType * hlcd, uint8_t adr)`
  - `TextLCD_SetPos     (TextLCDType * hlcd, int     col, int row)`
  - `TextLCD_PutChar    (TextLCDType * hlcd, char    c)`
  - `TextLCD_PutStr     (TextLCDType * hlcd, char  * str)`
- Har korrekta (enligt databladet) fördröjningsanrop i all implementerad funktionalitet. De under 1 ms hanteras i laborationens sista del.
- Visar tiden (`HH:MM:SS`) längst till höger på LCD-skärmens nedre rad.

Bonus:

- Implementera resten av funktionaliteten på s. 24-25. Bl.a. blinkande markör och scrollande text är intressant.
- Använd alternativa typsnittsuppsättningen som finns på s. 18.
- *(svår)* Ladda över egen data in i CGRAM. Videon länkad nedan är ett exempel på vad som kan göras.

https://www.youtube.com/watch?v=TkhVqe8Z2lY

3. Fördröjningsfunktion, registerprogrammering

3.1. Timer

Du ska nu implementera en fördröjningsfunktion som är på *mikrosekundsnivå*. För att göra detta måste en lämplig timer väljas. Timern ska ställas in så att den räknar upp varje mikrosekund.

Om räknaren är 16-bitars, hur många millisekunder kan du som mest ha en fördröjning?

Om räknaren är 32-bitars, hur många minuter kan du som mest ha en fördröjning?

Titta i *databladet* för MCU:n. Du kommer att behöva ha en timer vars räknare är 32 bitar. **Vilka timers kan du använda?**

Välj en lämplig timer och aktivera den i CubeMX. Ställ in dess Prescaler så att den tickar en gång per mikrosekund.

För lösningen nedan behövs ingen interrupt. Du behöver ej heller lägga till någon anrop som sätter igång din valda timer. Detta ska göras genom registren.

3.2. Lösning

Du ska nu skriva om funktionen `My_Delay()` i `lcd.c`. Din lösning ska stanna i funktionen tills önskat antal mikrosekunder har passerat. Om du lägger till dessa två rader allra först i `TextLCD_Init()` så kan du enkelt testa om du gjort rätt med en breakpoint och Step Over-knappen i debuggern:

```
uint32_t dly = 5 * 1000 * 1000; // 5 seconds
My_Delay(dly);
```

Öppna *referensmanualens* kapitel TIM2 to TIM5 registers. Hitta den bit i kontrollregistren (CR1, CR2) som slår av/på räknaren.

För att skriva över en bit så är det viktigt att du inte skriver över några andra bitar i registret. Exemplen nedan jobbar med CCDS-biten i CR2 för TIM3 utan att några andra bitar påverkas:

Enchipsdatorer – Laboration 4
LCD-skärm

```
uint16_t cr2 = TIM3->CR2;    // little endian is best endian <3
cr2          = cr2 | 0x0008; // same as (0x0001 << 3)
TIM3->CR2    = cr2;

uint16_t cr2 = TIM3->CR2;
cr2          = cr2 & ~(0x0001 << 3); // ~ is cool
TIM3->CR2    = cr2;
```

I kapitlet TIM2 to TIM5 functional description finns ett underkapitel, Time-base unit. Läs det kapitlet. Fokusera särskilt på de tre register som nämns i början samt läs om dem i listan av register.

Det står i att du *kan* skriva till dessa tre registren samtidigt som timern är igång, men för denna lösning är det inte tillåtet. Din lösning ska stoppa timern, skriva i nödvändiga rester och sedan starta den igen.

Lösningshjälp finns längst ner på denna sida. Försök gärna utan att att läsa den.

Vill du ha en ytterligare ledtråd så öppna upp (med CTRL+klick) källkoden för `HAL_Delay()`.

Bonusövning 1 (lätt): Kopiera din implementation och skriv om den till att använda de makron som genererats i HAL. Börja med att skriva ”__HAL_TIM_SET” för att sedan trycka på autokomplettering (vanligtvis Ctrl+mellanslag).

Bonusövning 2: Skriv en variant mer lik `HAL_Delay()` som inte nollställer någon räknare.

Bonusövning 3: Skriv en variant som får en interrupt när tiden är ute.

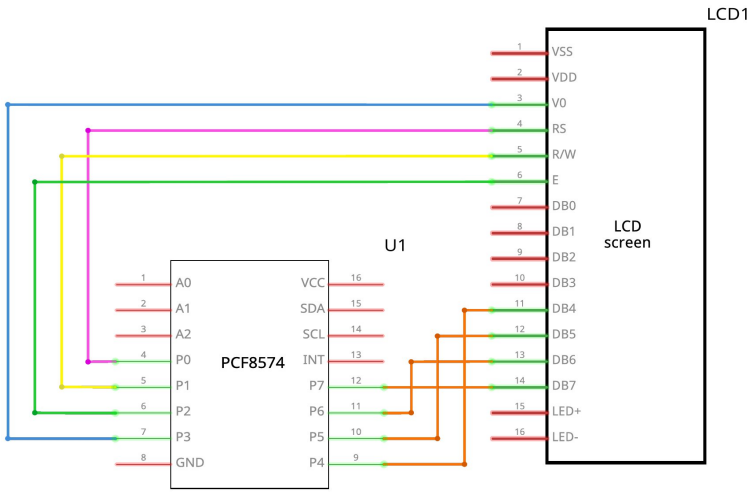
1. Stoppa timern.
2. Nollställ räknaren.
3. Starta timern.
4. Vänta tills räknaren nått upp till fördröjningstiden.

4. Appendix

4.1.LCD-skärmens I/O-expanderare

När en byte skickas över I2C till PCF8574 så läggs MSB på P7 och LSB på P0.

V0: Bakgrundsbelysning
RS: Register Select
R/W: Read/Write (1 = Read, 0 = Write)
E: Enable, signalen som berättar för skärmen att det är dags att göra en läsning.



4.2.Metodbeskrivningar, TextLCD

- `void TextLCD_Init( TextLCDType *lcd, I2C_HandleTypeDef *hi2c, uint8_t dev_address )`

  Initierar LCD-displayen. Startar upp displayen och ser till att den initieras med rätt värde så att det passar till displayen som används. (Se manual om hur man ska initiera sin LCD-display.)
  Ex. om du har en variabel av typen TextLCDType, en handle till I2C-periferien (i detta fall enheten I2C1) den är kopplad till samt dess I2C-adress (i detta fall `0x4E`) så körs initieringen så här:

  ```
  TextLCD_Init(&lcd, &hi2c1, 0x4E);
  ```

- `void TextLCD_Home(TextLCDType *lcd)`

  Markören(Cursor) återvänder till första positionen.
  Ex. om du har en variabel av typen TextLCDType med variabelnamn lcd:

  ```
  TextLCD_Home(&lcd);
  ```

- `void TextLCD_Clear(TextLCDType *lcd)`

  Tar bort allt i skärmminnet/på skärmen och markören återvänder till första positionen.
  Ex. om du har en variabel av typen TextLCDType med variabelnamn lcd:

  ```
  TextLCD_Clear(&lcd);
  ```

- `void TextLCD_Position(TextLCDType *lcd, int x, int y)`
  Sätter markören i en viss position (x,y)-läge.
  Ex. om du har en variabel av typen TextLCDType med variabelnamn lcd och vill att markören ska vara på andra raden och första kolumn (hur många rader och kolumner som finns beror på din LCD-display) i detta fall blir x = 0, y = 1. (0,1):

  ```
  TextLCD_Position(&lcd, 0, 1);
  ```

- `void TextLCD_Putchar(TextLCDType *lcd, uint8_t data)`
  Skriver ut ett tecken på LCD-displayen.
  Ex. om du har en variabel av typen TextLCDType med variabelnamn lcd och vill skriva ut bokstaven M:

  ```
  TextLCD_Putchar(&lcd, 'M');
  ```

- `void TextLCD_Puts(TextLCDType *lcd, char *string)`
  Skriver ut en sträng på LCD-displayen.
  Ex. om du har en variabel av typen TextLCDType med variabelnamn lcd och vill skriva ut Hejsan!:

  ```
  TextLCD_Puts(&lcd, "Hejsan!");
  ```

LCD-funktioner – Interna.

- `void MyDelay(uint32_t mysec)`
  Skall utföra en fördröjning på mysec antal mikrosekunder.

- `static void TextLCD_SendNibbleWithPulseOnE(TextLCDType *lcd, uint8_t data)`
  Pga. Hur IO-expanderaren är kopplad så måste skärmen drivas i 4-bitars-läge. Ordet ”nibble” betyder just 4 bitar, dvs. en halv byte. Skärmen ignorerar vad som finns på data-ingångarna tills något händer på E. Därför skrivs varje nibble tre gånger: en gång med E låg, sedan E hög (så skärmen läser) och sedan E låg igen. Endast de övre 4 bitarna av data kommer att skickas. De lägre fyra kontrollerar skärmens tillstånd. Se kommentarer i header-filen för ytterligare detaljer.

  ```
  TextLCD_SendNibbleWithPulseOnE(&lcd, 0x38);
  ```

- `static void TextLCD_SendByte(TextLCDType *lcd, uint8_t data):`
  Delar upp datan och använder sig av SendNibble() två gånger.

  ```
  TextLCD_Data(&lcd, 0x55);
  ```