FYS3240 Lab 1

Til denne oppgaven har dere fått utlevert litt ferdig kode for å hjelpe dere med strukturen i programmet deres. Prosjektet lages i AVRStudio 5. Opprett ett prosjekt, og velg kortet XMEGA128B1.

Før dere starter anbefales dere å gå igjennom dokumentet ”XMEGA-B1 Xplained Hardware User Guide” for å sette dere litt inn i oppbygningen av utviklingskortet. Ellers finnes det meste av informasjon i databladet for XMEGA B.

Mikrokontrolleren kommer levert med en ferdig bootloader som lar oss laste opp kode over en virtuell COM-port på USB. For å laste opp kode til denne bruker vi programmet Atmel FLIP 3.4.5 som er installert på maskinene. Dette gir oss muligheten til å klare oss uten fordyrende verktøy.
En beskrivelse av hvordan FLIP brukes kan finnes i dokumentet ”XMEGA-B1 Xplained Getting Started Guide”. Pass på å programmere til FLASH.

Siden bootloaderen vil gi kontrollen til programmet som lastes opp helt til det er ferdig med å eksekvere er det lurt å lage en mekanisme etter hvert som sørger for at programmet avsluttes. Hvis ikke må utviklingskortet skrus av og på hver gang det skal programmeres.
For alle oppgavene til mikrokontrolleren blir det utdelt ferdige kodeskall i header-filer, da noen av rutinene skal brukes videre i senere oppgaver.

Siden det er begrenset lagringsplass i en mikrokontroller bør plassbesparende typer som uint8_t og uint16_t, etter hvor lang variabel som trengs, brukes.
I XMEGA-serien sine headere er alle periferier lagt opp som struct’er, hvor kontroll- og statusregistrene er ordnet under disse.
For å gjøre koden mer portabel bør de predefinerte bitmaskene for de forskjellige innstillingene i registrene, som finnes i io-headeren, brukes. For eksempel *PORT_INVEN_bm* som er definert på linje 5296 i *iox128b1.h* for å invertere ut-/inngangssignalet på portene. AVR Studio 5 har autofullføring av disse navnene når headerfilen er inkludert i kildefilen man jobber med.
For eksempel for alle portene finnes følgende struktur:

```
typedef struct PORT_struct
{
register8_t DIR; /* I/O Port Data Direction */
register8_t DIRSET; /* I/O Port Data Direction Set */
register8_t DIRCLR; /* I/O Port Data Direction Clear */
register8_t DIRTGL; /* I/O Port Data Direction Toggle */
register8_t OUT; /* I/O Port Output */
register8_t OUTSET; /* I/O Port Output Set */
register8_t OUTCLR; /* I/O Port Output Clear */
register8_t OUTTGL; /* I/O Port Output Toggle */
register8_t IN; /* I/O port Input */
register8_t INTCTRL; /* Interrupt Control Register */
register8_t INT0MASK; /* Port Interrupt 0 Mask */
register8_t INT1MASK; /* Port Interrupt 1 Mask */
register8_t INTFLAGS; /* Interrupt Flag Register */
register8_t reserved_0x0D;
register8_t REMAP; /* I/O Port Pin Remap Register */
register8_t reserved_0x0F;
register8_t PIN0CTRL; /* Pin 0 Control Register */
register8_t PIN1CTRL; /* Pin 1 Control Register */
register8_t PIN2CTRL; /* Pin 2 Control Register */
register8_t PIN3CTRL; /* Pin 3 Control Register */
register8_t PIN4CTRL; /* Pin 4 Control Register */
register8_t PIN5CTRL; /* Pin 5 Control Register */
register8_t PIN6CTRL; /* Pin 6 Control Register */
register8_t PIN7CTRL; /* Pin 7 Control Register */
} PORT_t;
```

Bent Furevik/ Snorre Bjørnstad, revidert 02.02.2015

Et register i en struct kan aksesseres som for eksempel *PORTB.DIR = 0b11110000* for å sette pinne 4-7 som utganger. Et register kan sees på som en rekke bits, hvor de for eksempel i DIR-registeret korresponderer til en pinne på den fysiske porten.
Alle registernavnene finnes i databladet, og også i io-headeren til mikrokontrolleren; iox128b1.h. Denne finnes på PCen under *Programfiler/Atmel/AVR Studio 5.0/AVR Toolchain/avr/include/avr/iox128b1.h* og er verdt en kikk.

Den første oppgaven er å få LED-lysene på kortet til å lyse opp når korresponderende bryter trykkes. Som dere har lest i dokumentasjonen til Xplained-kortet er bryterne og LED-lysene aktivt lave. Derfor er det en fordel å konfigurere portene invertert. Det forenkler logikken i koden, og gjør det lettere når det skal gjenbrukes i senere oppgaver. (Hint: side 142 i databladet)

Eksempelkoden som er levert med oppgaven inneholder en «Header»-fil og en C-fil. I Header-filen skal det ligge prototyper for alle funksjoner. Denne prototypen forteller kompilatoren, hva funksjonen heter, hvile innganger den har, og hvilke utganger den har. Dersom du lager flere funksjoner enn de som er gitt, må du også lage en prototype som stemmer over ens med funksjonen. Makroer skal også defineres i Header-filen. Makroer kan være enkle definisjoner av konstante verdier, eller enkle funksjoner. Det er ikke alltid nødvendig å benytte makroer i disse oppgavene, men de kan være nyttige og øke lesbarheten av koden. Makroer kan f.eks se slik ut:

```
#define F_CPU 32000000UL
#define turnOnLed3() PORTB_OUT |= 0b10000000
```

I C-filen skal funksjonene implementeres, her ligger alle funksjons-kall og logikk som ikke er makroer.

Del 1: Hello World!
1. LED-lysene er koblet på PORTB pinne 4-7, så det første skrittet blir å konfigurere PORTB som en utgang. Dette gjøres ved å sette PORTB.DIR høy for de aktuelle bitene.

2. De kapasitive bryterene på kortet er koblet til PORTE pinne 0-3, så retningen på denne må også settes. Vi må konfigurere denne porten som inngang slik at vi kan lese av en eksternt påtrykt verdi.

3. Disse bryterne må kobles til pull-up motstander og det er viktig å slå disse på tidlig i initialiseringen. Hvorfor dette må gjøres er beskrevet i databladet for utviklingskortet. For å gjøre koden penere kan flere pinner konfigureres samtidig, ved å sette riktig verdi i *Multi-pin Configuration Mask Register* i *Port Configuration*. Verdien til dette registeret sier hvilke pinner som skal konfigureres samtidig og det nullstilles etter én skriveoperasjon til et pinnekontrollregister som hører til den oppsatte verdien. Dette er et globalt register, og det gjør ingen forskjell på hvilken port det brukes til å konfigurere.

4. En annen ting som kan vise seg å være nøvendig er å bitmaske PORTE, slik at kun de fire interessante bit'ene plukkes ut. (Hint: &)

5. For å sette en verdi på utgangspinnene må vi sette en verdi til utgangsregisteret som hører til PORTB pinne 4-7. Det blir da naturlig sette denne lik inngangsverdien for PORTE pinne 0-3. De aktuelle registernavnene finnes i databladet.

6. Deretter må koden kompileres. Dette kan enten gjøres med den utleverte batch-filen, eller med AVR Studio
5. For å kompilere til en hex-fil, som vi kan laste opp med FLIP, må vi først gjøre noen innstillinger.

7. I AVR Studio må vi gå inn på "Project –> Properties og krysse av i boksen for .hex-fil. Deretter trykker vi på Build -> Build for å kompilere.

8. For å laste opp koden på mikrokontrolleren skal dere bruke FLIP.

Bent Furevik/ Snorre Bjørnstad, revidert 02.02.2015

Del 2: Binærteller med LEDs

1. Før vi gjør noe mer skal vi sette opp systemet til å bruke den interne 32MHz klokka i stedet for den på 2MHz. Registrene vi skal skrive til er beskyttede IO-registere, og vi må sette riktig verdi i CCP-registeret (Configuration Change Protection, side 14) før vi setter en ny verdi til et slikt register.

2. Først må dere aktivere riktig oscillator (s. 89), og vente til statusregisteret dens sier at den er klar (Hint: while-løkke) før den kan velges som systemklokke. Deretter velges denne oscillatoren som kilde for systemklokka. Se databladet.

3. For å teste at 32MHz klokken fungerer kan dere bruke delay-funksjonene fra include-fila «util/delay.h», f.eks. _delay_ms().

4. Vi bruker de samme portene som i del 1, men vi skal kun bruke CS1 (Capacitive Switch 1) til å telle. Siden QTouch-kontrolleren som styrer de kapasitive bryterne CS1-4 er konfigurert som standard til å kun registrere den første bryteren som blir trykket om flere blir trykket samtidig blir testingen av tastetrykk svært enkel.

5. Deretter er det bare å sette opp en enkel test for å inkrementere og dekrementere en teller. Denne telleren skal vises på LED-lysene. CS1 skal inkrementere og CS2 skal dekrementere. For å vise verdien på LED-lysene må verdien bitshiftes 4 bit mot venstre, siden LED'ene er koblet på PORTB[4-7].

6. Programmet skal settes opp slik at rutinen fra del 2 kjøres først, og blir avsluttet ved å trykke på CS3. Deretter skal programmet gå inn i rutinen fra del 1, som også skal avsluttes ved å trykke på CS3. LED4 skal lyse opp for å fortelle brukeren at programmet er avsluttet.

(Hint: to while-løkker)
Bruk av CCP-registeret er tidskritisk og dere kan bli nødt til å endre optimaliseringsnivået i kompilatoren for å unngå problemer med dette. Det gjøres ved å endre –O1 flagget til –Os i batch-fila, eller i optimaliseringsnivået i AVR Studio 5. Vanligvis er dette ikke noe problem riktignok.

Innlevering:
Kommentert kildekode
Hex-fil

Bent Furevik/ Snorre Bjørnstad, revidert 02.02.2015