HiST/EDT, RKS

Mikroprosessorsystemer

Labøving 3 – Eeprom lesing og skriving. Eksterne avbrudd.

Oppgave 1 - Eeprom.

Lag et program i C som inneholder en funksjon for skriving til eeprom og en for lesing. Det er forslag til funsjoner i databladet for AVRen. Bruk så funksjonene til å skrive verdier til adresse 0x01 til 0x10 i eeprom'en og lese ut en av verdiene fra EEPROM'en. Simuler programmet og se på EEPROM i Memory view.

EEprom'en er beskrevet i databladet.

Funksjonene for skriving og lesing kan for eksempel defineres slik:

```
void EEPROM_write(uint16_t address, uint8_t data)
uint8_t EEPROM_read(uint16_t address)
```

Ekstra til oppgave 1 – valgfritt.
Bruk de innebyggede funksjonene i AVRGCC/AVR-libc for å skrive og lese bytes til eeprom. Du finner dokumentasjon under `avr/eeprom.h` i avr libc manualen, samt en tutorial under AVR articles/tutorials i It's learning.

Oppgave 2 - Avbrudd.

Avbrudd (interrupt) er et signal til en mikroprosessor om at en hendelse har skjedd. Dette kan brukes til å reagere hurtig på hendelsen. Da hopper prosessoren ut av programsekvensen den holder på å gjøre til en avbruddsrutine før den hopper tilbake til koden der den ble avbrutt.

En avbruddsrutine er en funksjon som blir kalt av hardware og kan derfor ikke returnere data som en vanlig C funsjon. Avbrudd er dermed ikke definert i standard C og syntaksen for avbrudd er derfor spesifikk for hver enkel kompilator. AVRGCC avbrudd er beskrevet i avr-libc dokumentasjonen:
http://www.nongnu.org/avr-libc/user-manual/index.html
Avbrudd i ATmega2560 er beskrevet i databladet.

I denne oppgaven skal vi skrive et program som bruker det eksterne avbruddet INT0 for å detektere lave pulser (negativ flanke) på PD0 og toggle (skifte verdi på) PB0 for hvert trykk.

Debug og verifiser at programmet virker som forventet. For å få avbrudd til å trigge ved «stepping» av programmet ved debugging/simulering, må man endre en konfigurasjon i Atmel Studio:
«Tools» menyen -> «Options and Settings» -> «Tools» -> «Tool settings» -> «Mask interrupts while stepping» settes til «False»

INT0 avbruddet kan trigges ved å sette PIND0 bit'et høyt i I/O-view og kjøre/simulere minst en klokkesykel, for så å sette PIND0 lav og kjøre/simulere minst en klokkesykel.

Forslag til programstruktur:

```
/*      Header describing program   */

#include <avr/interrupt.h>

int main (void)
{
        // PORTB0 output
        // configure INT0 to falling flank
        // enable INT0 interrupt
        // enable global interrupts

        do {  // Eternal loop. Code can be inserted here that run independently of PORTB0 toggling.
                asm("nop");  // Hack for debugging. Remove before programming.
        } while (1);
}

// Interrupt handler routine for INT0. Toggles PB0 pin.
ISR(__vect)
{
        // Toggle PB0
}
```

Ekstra til oppgave 2 – valgfritt.
Sjekk hvor mange klokkesykluser det tar å kjøre avbruddet og noter resultatet som en kommentar i filen. Sjekk så hvorfor det tar så mange klokkesykler ved hjelp av disassembly view. Har du noen forslag til endringer som kan gjøre det raskere?

Test så programmet på ATmega2560 i STK600.
Lever oversiktlig og kommentert c-fil via It's Learning.