Namn:
Laborationen godkänd:

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet
LTH Ingenjörshögskolan vid Campus Helsingborg

Digitala system 15 p

Datorprojekt, del 4

Projektlaboration 4, synkronisering av klockan

Asynkron seriekommunikation.

Förberedelser.

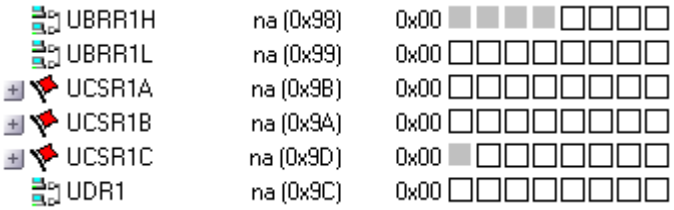
I denna laboration ska vi komplettera vårt digitalur med en synkroniserings mekanism. Med den asynkrona seriekommunikationen ska meddelanden tas emot varje sekund. Hur de ser ut finns beskrivet längre fram i laborationen.

Läs de delar av ***"USART register description" sid 189 – 197 i databoken Atmega128*** som du behöver för att förstå resten av förberedelserna. Placera de nya funktionerna i en fil **AVR-MT-128-USART.c.**

Erik Larsson *30 mars 2016*

USART är den engelska förkortningen för Universal Synchronous and Asynchronous serial Receiver and Transmitter. Den är en kraftfull och användbar periferienhet för seriell kommunikation. Vi ska använda den för att ta emot 8-bitars seriella dataord (kallas ***tecken*** i fortsättningen).

Vad behöver vi känna till för att kunna använda denna enhet? Atmega 128 innehåller två USARTar. Den som finns tillgänglig på labkortet är USART1, och den går att ansluta på två olika kontakter. Två pinnar i port D används: TxD – bit 3 och RxD – bit 2.
Här är de register som vi ser i AVR Studio:



Vi har ett register för inställning av bithastigheten (UBRR1), tre kontroll- och statusregister (UCSR1A, UCSR1B, UCSR1C) och ett register för dataöverföring (UDR1).

UDR1 är dataregistret för både sändning och mottagning av tecken. Egentligen är det två olika register men de har samma adress.
Vi kommer bara att behöva ta emot tecken och alltså läsa i UDR1.

UBRR1H och **UBRR1L** (höga och låga halvan av UBRR1) är register där man ställer in bithastigheten för den seriella överföringen. Man kan tolka dem som ett 12-bitars register UBRR1 där man lägger ett tal som delar ner mikroprocessorns klockfrekvens så att serieöverföringen går i rätt fart. Den frekvens man får, är mottagarens samplingsfrekvens.

Vi använder klockfrekvensen 16 MHz och ska använda överföringshastigheten 2400 baud.

Titta i databladen för ATmega128! Överföringshastigheten påverkas också av biten U2X1. Vilket värde ska läggas i UBRR1 och vad ska U2X1 ställas in på?

UBRR1 = …….. U2X1 = …….

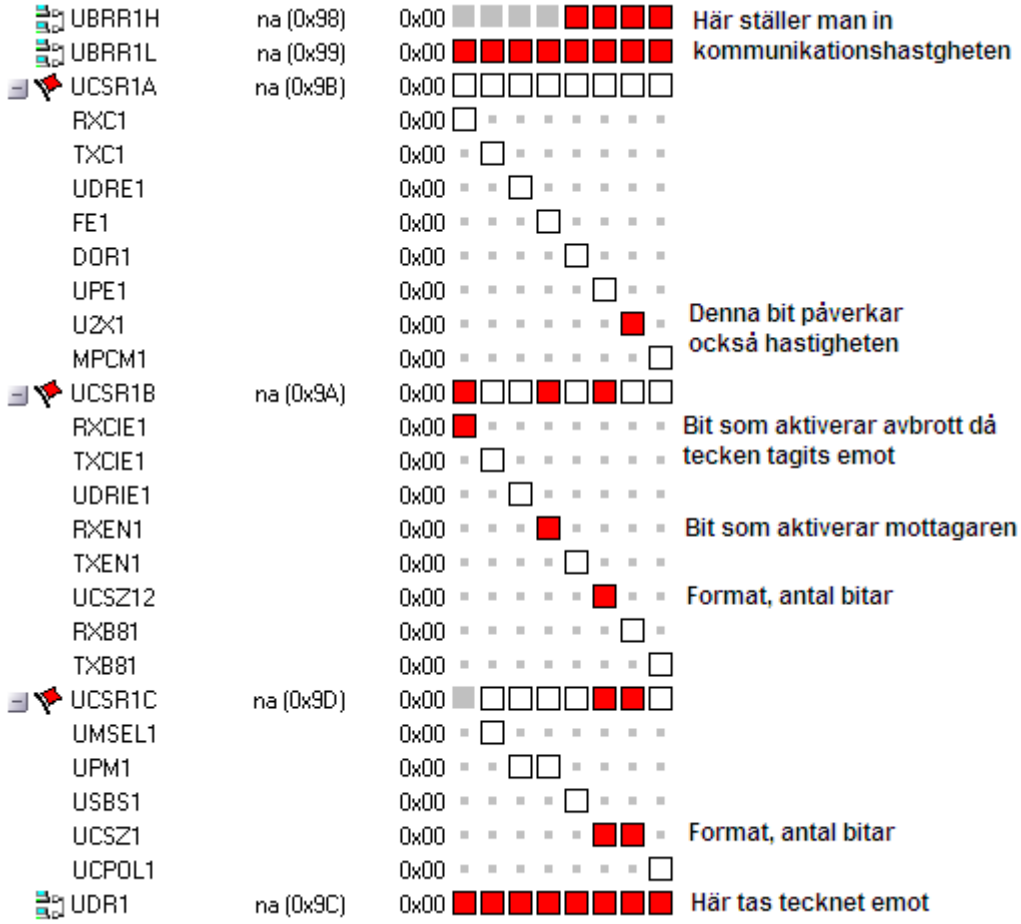
Vår uppgift är alltså att ta emot tecken med 2400 baud. Nu vet vi hur man ställer in hastigheten och var man läser de mottagna tecknen.

Men vi har några inställningar kvar:

- ❒ Vi ska ta emot teckenmeddelanden på 8 bitar (man kan välja 5, 6, 7, 8 eller 9).
- ❒ Vi måste starta igång USARTen.
- ❒ Vi vill också att ett speciellt avbrott ska ske varje gång vi får ett nytt tecken.

Bitar vi då behöver känna till:

UCSZ1: (3 bitar) Character Size.
RXEN1: Receiver Enable.
RXCIE1: Receive Complete Interrupt Enable.



De röda bitarna ska du känna till funktionen på.
Hur ska dessa bitar ställas in? Kan alla ställas in? Kontrollera i databoken.

Uppgift 1. initUSART1. Initiering av seriekanalen.

Skriv funktionen **initUSART1** som gör följande:

1. Mottagaren ska kopplas på.
2. Den ska vara inställd på 8 bitar och en stoppbit.
3. Hastigheten ska vara 2400 baud (vid 16 MHz klockfrekvens)
4. Avbrott ska ske när ett tecken kommit.

```
void initUSART1 (void){

// Uppgift 1

}
```

Uppgift 2. Avbrott för mottagning av tecken.

Funktion för mottagning av tecken.
Tecken som kommer in ska läggas i en global variabel **character_in**.

Avbrottsrutinens namn är förutbestämt.

```
//============================================================
//=========== Seriell mottagning. ============================
//============================================================
unsigned char character_in;
ISR (SIG_USART1_RECV){
    character_in = UDR1;

// Uppgift 3

}
```

Du kan testa denna avbrottsfunktion genom att skicka iväg ett tecken och se om du får tillbaka det igen. Hur gör man för att skicka iväg ett tecken?

.....

Du bör få ett avbrott och om du sätter en brytpunkt i den nya avbrottsfunktionen kan du se om det fungerar.

Feldetektering.

Uppgift 3. Synkroniseringen.

Det finns ett AVR-MT-128 – kort som är programmerat så att det sänder ut en tidssignal varje sekund. Det sker i en funktion sendClock som du ser här nedanför. Du behöver inte skriva in den; den finns här för att du ska kunna förstå hur mottagningsfunktionen ska se ut.
Funktionen borde kanske kommenteras, men personen som skrivit den orkade inte. Tänk själv ut vad den utför och besvara frågorna.

```
  ////////////////////////////////////
 // Styrklockans sändningsfunktion. //
////////////////////////////////////

void sendClock(void){
    unsigned char sum=0;
    sendCharacter(255);
    sendCharacter(seconds);
    sum = sum + seconds;
    sendCharacter(minutes);
    sum = sum + minutes;
    sendCharacter(hours);
    sum = sum + hours;
    sendCharacter(~sum);
}
```

1. Hur många värden skickar funktionen åt gången? .....
2. Varför skickas värdet 255 ut i början? .....
3. Vad skickas på slutet? .....
4. Vilken typ av feldetektering används alltså? .....

Gör ett besök i anteckningarna från kursen Datorkommunikation!

Det sista värdet som du tar emot ska alltså användas för att kontrollera att hela meddelandet är rätt.
För att kunna ta emot tidssignalen behöver du en liten buffert för seriemeddelandet. Istället för att bara ha en plats, character_in, ska du nu komplettera med en buffert med fyra platser där du kan ta emot enkla små meddelanden. Så fort bufferten är full, sätts en variabel, new_message = 1. Här får du koden till detta; den ska in i avbrottsrutinen för USART. Koden är skriven i språket C-venska. :o)

```
// Bufferten:

char new_message;            // Ettställs om ett nytt meddelande kommit
char serial_buffer[4];       // Bufferten med fyra platser
char serial_buffer_pointer;  // Buffertpekare

// Koden som skriver i bufferten:

    om (inläst värde är 0xFF) Nollställ buffertpekaren;

    annars om (bufferten ännu inte full) {
        Lägg character_in i bufferten;
        Öka buffertpekaren;
        om (buffertpekaren är 4) new_message=1;
    }
```

Skriv in koden (översätt till C)!

Uppgift 4. Uppdatering av klockan.

Med hjälp av funktionerna ovan ska du nu kunna skriva en mottagarfunktion, **getClock**, som plockar in tidssignalen från bufferten varje gång new_message = 1.

Använd samma princip som när new_time läses av och kvitteras.

Tänk på: Uppdatera ***inte*** timmar, minuter och sekunder om meddelandet är felaktigt. Det kommer sådana meddelanden också.

När allt fungerar stabilt, är du klar.