AVRStudio på tre minuter

Micke Josefsson, 2005

Mycket kort intro till AVRStudio

Utvecklingsmiljön AVRStudio innehåller en editor för att mata in programmet, en simulator för att under kontrollerade former enkelstega genom programmet och slutligen en programmerare som kan användas för att ladda ner den färdiga koden till målsystemet. ”Målsystemet” är alltid en processor men den kan sitta på olika utvecklingskort eller direkt i en brännarenhet.

Detta är en enkel bruksanvisning som steg för steg visar hur man skapar ett enkelt program och simulerar detsamma. Du behöver antagligen bara läsa igenom det en enda gång; AVRStudio är enkelt att hantera.

Från kod till processor

Det hela börjar med att koden matas in i en editor. Denna kod måste sedan kompileras. Kompileringen innehåller flera steg men de viktigaste för användaren är att man får en lista med syntaktiska fel så att man har möjlighet att korrigera sin kod. En korrekt kompilering resulterar i en hex-fil, en fil med all nödvändig programinformation i ett nedladdningsformat.

En hex-fil är inte direkt läsbar. Innehållet är programmets opkoder i hexadecimal form tillsammans med deras adresser. Även om den inte är tänkt att läsas av människor är den inte helt obegriplig:

```
:020000020000FC
:1000000018E011BB22B3212722BB00E40395D1F3F2
:02001000FDCF22
:00000001FF
```

Nja, ok då, det kanske den är. Efter kompileringen kan koden simuleras. Det innebär att hex-filen används som indata till AVRStudio.

Under denna process har några andra filer också tillverkats för AVRStudios interna bruk, bland annat en fil som håller ihop det projekt vi bygger upp, dvs information om vilka olika källkodsfiler (asssembler) som ingår. Man kan alltså låta det slutliga programmet härröra från flera filer om man vill. Det är inget vi bryr oss om här.

Starta AVRStudio

Efter inloggning finns ikonen ”Shortcut to AVRStudio” på skrivbordet. Dubbelklicka på den för att dra igång utvecklingsmiljön.

Du kan nu välja att öppna ett befintligt projekt eller skapa ett nytt. Ett ”projekt” i detta sammanhang är den samling filer och konfigureringar som behövs för AVRStudio. Här har vi inget gammal projekt att öppna. Alltså:

- Välj ”Create New project”.
- Fyll i ”Project Name”.
- Ändra ”Location” till din hemkatalog på vanheden.
  - Klicka på [...], och sedan [↑] tills du kommer till ”gruppXX on ’Samba server (vanheden)’ (Q:)”
  - Välj denna katalog, din hemkatalog. Här kan du nu skapa en egen katalog för projektet om du vill.
  - Avsluta med att klicka på ”Select” och sedan ”Finish”.
- Du har nu ett editorfönster där du direkt kan skriva in kod.

Skriva program

Här visas i kort form vilka mått och steg man måste vidta för att få igång ett enkelt program. Programmet ska bara flippa bit 3 på PORTD. Programmet är inte avsett att vara exempel på det bästa sättet att göra detta, utan bara visa hur assemblerfilen kan se ut med de speciella reserverade ord som kan användas.

AVRStudio ställer några få krav på hur den inmatade koden ska se ut för att kunna kompilera den. Här kommer koden som den ska se ut när den är klar så tar vi detaljerna efteråt:

```
.include "m8def.inc"
.def loopraknare  = r16
.def temp         = r17
.equ start        = 0x40

.org 0x00

       rjmp   main

       ; här finns plats för avbrottsvektorer

.org   0x20
main:
       ldi    temp,0b00001000   ; bit 3 utgång
       out    ddrd,temp         ; konfigurera!

flip:
       in     r18,portd         ; läs av portd
       eor    r18,temp          ; r18 = r18 XOR temp
       out    portd,r18         ; skicka till porten
       ldi    loopraknare,start
loop:
       inc    loopraknare       ; sätter flaggor (Z)
       breq   flip
       rjmp   loop
```

- `.def` definierar ett annat namn på ett register. Det underlättar **betydligt** om man använder förståndiga namn på sina register.
- `.equ` används för att sätta en konstant till något värde. Här är konstanten angiven i hexadecimal form men det går att ange den decimalt eller binärt också.
- Första radens `.INCLUDE` laddar en (rätt stor) fil med många fördefinierade namn på register och konstanter. Det gör att vi kan skriva `out portd,r18` till exempel. `portd` är definierat med `.equ` i den filen. `m8def.inc` används för ATMega8 och `m16def.inc` för ATMega16.
- `.org`, *origin*, anger på vilken adress i minnet den följande koden skall placeras. Processorn hämtar sin första instruktion efter spänningspåslag från adress 0. En del av de lägre adresserna är fördefinierade avbrottsvektorer varför man ofta hoppar över dessa, därav `rjmp main`.
- Alla *lables*, symboliska adresser, måste börja i första kolumnen och avslutas med :.

- Opkoderna börjar åtminstone ett tabb-stopp in. Det finns inget krav att operanderna behöver skiljas med tab-stopp.
- Kommentarer inleds med semikolon.
- Filen behöver inte avslutas med `END` e dyl.

Simulera programmet

För att kunna simulera projektet måste det först kompileras och en hexfil skapas. För att göra detta:

- Under ”Project”-menyn, välj ”Build”. Eller använd tangenten F7 direkt. Detta kan ge felmeddelanden i det nedre fönstret. Klicka på respektive felmeddelande så kommer du till den felande kodraden. Ändra och kompilera om till allt gått igenom.
- Under ”Debug”-menyn väljer du nu ”Start Debugging”.
- Nu dyker ett fönster (kallat *workspace*) upp till vänster med processorns register, I/O-portar mm. Här visas all processorstatus. Du kan högerklicka på raderna för att få mer information.
- Under ”Debug”-menyn kan du nu välja ”Step Into” eller trycka F11 eller klicka direkt på symbolen ”{¬}”. Effekten är i vart fall att du startat exekveringen med att gå en programrad i programmet. I *workspace*-fönstret kan du inspektera de berörda registren osv.

Hur fort går det?

I *workspace*-fönstret finns alternativet ”Processor”. Under denna finns programräknarens värde men också en praktisk instruktionsräknare ”Cycle Counter”, och ett stoppur du kan använda för att ta tid på fördröjningsloopar osv.

Simulatorn antar att processorn kör med klockfrekvensen 4 MHz men detta värde går att ändra under ”Debug->AVR Simulator Options”.

Bränna koden till processorn

Det finns en hel uppsjö med olika metoder att bränna in koden i processorn. Här visas hur det går till om man använder Atmels utvärderingskit STK500. Använder du JTAG-pogrammering finns detaljerade inkopplings- och användarinstruktioner i on-line-hjälpen till AVRStudio.

Med ett program som är kompilerat slår man först på spänningen till STK500.[1] På själva kretskortet sitter en liten strömbrytare upp till vänster. När kortet är spänningssatt lyser det från en lysdiod. Man kan inte missta sig.

Välj sedan från menyn i AVRStudio ”Tools->’STK500/...’->’STK500’ ” för att få fram programmeringsmenyn.

Här kan man välja vilken krets man avser att bränna. Kortet STK500 kan acceptera åtskilliga typer men i vårt fall är det bara ATMega8 som är aktuellt.

Se till att ”ISP” (In-Circuit Serial Programming) är valt som ”Programming mode”.

Kontrollera också under rubriken ”Fuses” att:

- Kryssrutan för *watch-dog* är rätt inställd
- Rätt klockalternativ används. Antingen *Ext Clock*, yttre klocka, eller *Int RC Osc*, intern klocka i form av en RC-oscillator med rätt frekvens, måste väljas.
- Kryssrutan för *Brown-out detection* är ikryssad.

Programmeringen i sig sker genom att trycka på knappen ”Program” i rutan för ”Flash memory”, och genomförs på några sekunder. Därefter resetas processorn och programmet startas. Om programmet skvallrar sin funktion på en utgående port kan denna kopplas till en av lysdioderna och man får omedelbart en bekräftelse på att programmet körs.

[1] Den vill ha 10-15 volt och bryr sig inte om polariteten så det är bara att koppla in rätt spänningsnivå.