

Datorteknik

Kursmateriel samlat i ett pdf-dokument med hyperlänkar

Dokumentet läses med Adobe Reader

Inställning: Menyn Edit -> Preferences -> Documents

"Open cross documents in same Window" skall ej vara förbockat

2016/01/17 09:06:11 +01'00'

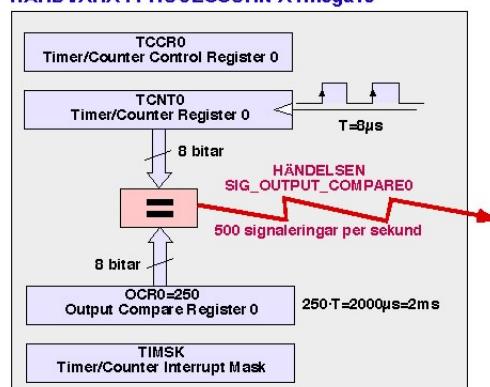
March 2, 2016

MJUKVARA FÖR ATT INITIERA TIMER0

```
int main()
{
    ...
    bTCCR0.cs0 = 3;      //Clock Select för TIMER0
                        //CK=8MHz/64
    bTCCR0.wgm01=1;    //Arbetsmod för TIMER0
                        //Output Compare Match-mode
    bTIMSK.ocie0 =1;   //Tillåt avbrott från TIMER0

    OCR0 = 250;         //Avbrott var 250:e klockpuls
```

HÄRDAVARA I PROCESSORN ATmega16



MJUKVARA-AVBROTTSFUNKTION

```
SIGNAL(SIG_OUTPUT_COMPARE0)
{
    ...
}
```

Händelsignaleringen
triggar exekveringen av
denna funktion 500 gånger
per sekund.

Innehåll

I Kursen	4
1 Kursen	6
2 Veckoplanering	9
II Assemblerprogrammering	18
3 Introduktion till grundläggande begrepp och binär aritmetik	21
4 Introduktion till AVR-processorn och programutvecklingsmiljön	23
5 Laboration 1: Lysdioder & strömtällare	25
6 Assemblerprogrammering av AVR-processor	48
7 Laboration 2: Subrutiner och aritmetik	50
8 Intel _x86 assembler	59
9 C# och CIL (Common Intermediate Language)	60
III Hårdvarunära programmering i C	61
10 Hårdvarunära programmering i C	64
11 Laboration 3: Hårdvarunära programmering i C	66
12 Klassen time	80
13 Tillståndsmodellering	84
14 FSM-implementering	94
IV Objektorienterad programmering i C	100
15 Modellering av inbyggda system med OOP-metodik i C	102
16 Laboration 4: Objektorienterad programmering i C	105
V Introduktion till reeltidsprogrammering	118
17 Avbrottshantering	121
18 Laboration 5: Objektorienterad modellering och avbrottshantering	124
19 Reeltidsprogrammering och konstruktion av små inbyggnadssystem	131
20 Projekt: Styrning och övervakning i en villa	133
VI Seriella bussar	147
21 Tretrådbussen - 3wire	149
22 RS232	159

Innehåll

VII Programvaror, datafiler och dokument	160
23 Programvaror som används i kurserna	162
24 Utvecklingsmiljön och program tillhörande denna	164
25 Böcker, kompendier och artiklar	167
26 Sensorer och andra typer av kretsar	170
27 Intel assemblér	171

Del I.

Kurser

Innehåll

1 Kursen	6
1.1 Kursplan	6
1.2 Kursinnehåll, examination och betygskriterier	6
1.3 Läs- och laborationsmoduler	7
2 Veckoplanering	9
2.1 Vecka 3 - Introduktion till assemblerprogrammering	10
2.2 Vecka 4 - Assemblerprogrammering	11
2.3 Vecka 5 - Hårdvarunära programmering i C	12
2.4 Vecka 6 - Objektorienterad modellering i C/C++ och tillståndsmodellering	13
2.5 Vecka 7 - Avbrottshantering	15
2.6 Vecka 8 - Projekt för betyg 4/5	16
2.7 Vecka 9 - Laborationer	17
2.8 Vecka 10 - Laborationer	17
2.9 Vecka 11 - Laborationer	17

1. Kursen

Innehåll

1.1	Kursplan	6
1.2	Kursinnehåll, examination och betygskriterier	6
1.3	Läs- och laborationsmoduler	7

1.1. Kursplan

Den officiella kursplanen kan här hämtas som ett pdf-dokument. Alternativt kan kursplanen hämtas från Örebro Universitets kursdatabas, länken dit är: [Kursplan](#).

1.2. Kursinnehåll, examination och betygskriterier

För högre betyg krävs att man uppfyller fodringarna för de lägre betygsgraderna.

- Betyg 3
 - Kursinnehåll
 - ◊ Talsystem och binär aritmetik
 - ◊ Assemblerprogrammering
 - ◊ Hårdvarunära programmering i C
 - ◊ Objektorienterad programmering i C
 - ◊ Avbrottsfunktioner
 - Lärandemål
 - ◊ Att förstå hur en dator fungerar på maskinnivån.
 - ◊ Att kunna programmera en inbyggndator i assembler och i högnivåspråket C.
 - ◊ Att förstå och kunna använda avbrottsfunktioner för att lösa olika realtidsprogrammeringsproblem.
 - Betygskriterie

1. Kursen

- ◊ Laboration 1 till 5 skall redovisas i form av ett PDF-dokument gjort i LyX. I dokumentet skall finnas inbäddat en komprimerad 7z-fil innehållande alla projekt. Uppgifterna redovisas med en programlistning och ibland med en bild av uppkopplingen, etc.
 - ◊ Ett antal kontrollskrivningar skall genomföras och lämnas in.
 - ◊ Närvarande under laborations och lektionstid.
- Betyg 4
 - Kursinnehåll
 - ◊ Taskabstraktionen för att bryta ner ett komplext problem till några mindre komplexa problem.
 - ◊ Realtidsprogrammering och statiska exekveringsscheman.
 - Lärandemål
 - ◊ Att kunna använda periodiska och icke periodiska tasks för att lösa realtidsprogrammeringsproblem
 - Betygskriterie
 - ◊ Laboration 6 skall genomföras och redovisas på ett nöjaktigt sätt.
 - ◊ Närvarande under laborations och lektionstid.
- Betyg 5
 - Lärandemål
 - ◊ Att självständigt definiera ett mindre projekt och skriva en kravspecifikation för detta alternativt att genomföra projektuppgiften ”Styrning och övervakning av en villa”,
 - ◊ Att genomföra projektet och tillämpa inhämtade kunskaper om programmering av inbyggnadssystem.
 - Betygskriterie
 - ◊ Ett mindre projekt skall göras och redovisas med en rapport gjord med LyX. Om ni gör ett egendefinierat projekt skall en liten kravspecifikation skrivas och redovisas för examinatörn för ett godkännande.
 - ◊ Närvarande under laborations och lektionstid.

1.3. Läs- och laborationsmoduler

Kursen är praktiskt uppdelad i ett antal läsmoduler som avslutas med ett självtest.

Följande är en lista över i vilken ordning du bör göra saker och ting. Denna lista innehåller läsmoduler, laborationer samt en punkt som anger hur du skall installera din programvara.

1. Kursen

1. Modul 1: Introduktion till grundläggande begrepp och binär aritmetik
2. Modul 2: Introduktion till AVR-processorn och programutvecklingsmiljön
 - a) Installation av programvaror
 - b) Laboration 1: Strömställare och lysdioder
3. Modul 3: Assemblerprogrammering av AVR-processorn
 - a) Laboration 2: Subrutiner och aritmetik
4. Modul 4: Hårdvarunära programmering i C
 - a) Laboration 3: C-programmering
5. Modul 5: Objektorienterad programmering och modellering i ett inbyggnadssystem
 - a) Laboration 4: Objektorienterad programmering i C
6. Modul 6: Avbrottshantering
 - a) Laboration 5: Objektorienterad modellering i C
7. Modul 7: Realtidsprogrammering och konstruktion av små inbyggnadssystem
 - a) Kapitel 11 och 12 läses om du siktar på betyg 4 eller 5.
 - b) Laboration 6: Realtidprogrammering
 - i. Laboration som skall göras om du siktar på betyg 4 eller 5.

Det är viktigt att du följer ovanstående ordning för hur teori och praktiska moment skall genomföras. Varje modul har en självtest som du skall göra när du har gått igenom den teori som hör ihop med modulen. Detta test ger en indikation på om du skall gå vidare eller läsa om teoriavsnittet.

2. Veckoplanering

Innehåll

2.1	Vecka 3 - Introduktion till assemblerprogrammering	10
2.2	Vecka 4 - Assemblerprogrammering	11
2.3	Vecka 5 - Hårdvarunära programmering i C	12
2.4	Vecka 6 - Objektorienterad modellering i C/C++ och tillståndsmodellering	13
2.5	Vecka 7 - Avbrottshantering	15
2.6	Vecka 8 - Projekt för betyg 4/5	16
2.7	Vecka 9 - Laborationer	17
2.8	Vecka 10 - Laborationer	17
2.9	Vecka 11 - Laborationer	17

Preliminärt innehåll för lektioner och laborationer ges för varje kursvecka. Innan en ny kursvecka startar bör man ha tittat igenom denna veckas videofilmer som finns på youtube.

Följande tider gäller för varje kursvecka:

- Tisdag 8-12 - Lektion / Laboration
- Torsdag 8-12 - Lektion / Laboration

2. Veckoplanering

2.1. Vecka 3 - Introduktion till assemblerprogrammering

- Videofilmer
 - [Microcontrollers - A Beginner's Guide - Intro to the AVR Atmega32](#)
 - [Learn Atmel AVR Programming - An Introduction](#)
- Kursintroduktion
- Introduktion till grundläggande begrepp och binär aritmetik
- Introduktion till AVR-processorn och programutvecklingsmiljön
- Laboration 1: Lysdioder & strömväxlar
- Kontrollskrivning 1
 - Skall vara klar i denna vecka.
- Slides
 - [Datorteknik](#)
 - [Mekatroniska system](#)
- Manualer för AVR-processorn
 - Manualerna är vår främsta källa till information om processorns hårdvaruuppgift och programmering av denna både på maskinnivån med assemblerkod och hårdvarunnära programmering i C/C++.
 - [Referensmanual ATmega32](#)
 - [Detaljerad Instruktionsmanual](#)
- Startprojekt för assemblerprogram
 -  [StartAssemblerAtmelStudio7.7z](#)

2. Veckoplanering

2.2. Vecka 4 - Assemblerprogrammering

- Videofilmer
- [Assemblerprogrammering av AVR-processor](#)
- [Laboration 2: Subrutiner och aritmetik](#)
- Kontrollskrivning 2
 - Skall vara klar i denna vecka.
- Slides
 - [Datorteknik](#)
- Exempel: Logiska signaler, detektering av negativ eller positiv flank, toogle-minne för att komma ihåg en negativ flank.
 -  NegativeFlanc.7z
 - Sist i kurskompendiet kan du ladda ner ExempelProgram.7z, där finns bland annat exemplet Start/Stopp Tidur
- Manualer för AVR-processorn
 - [Referensmanual ATmega32](#)
 - [Detaljerad Instruktionsmanual](#)
- Rapportmall i LYX/LATEX
 -  RapportSkrivning.7z

2. Veckoplanering

2.3. Vecka 5 - Hårdvarunära programmering i C

- Videofilmer
- Moment 3: Hårdvarunära programmering i C
 - [Hårdvarunära programmering i C](#)
 - [Laboration 3: Hårdvarunära programmering i C](#)
- Tillståndsmodellering - En programmeringsparadigm
 - [Tillståndsmodellering](#)
 - [FSM-implementering](#)
 - Exempel: Falsk Elektronisk Tärning (uppgift på laboration 3)
- Kontrollskrivning 3
 - Skall vara klar i denna vecka.
- Slides
 - [Datorteknik](#)
 - [Effektiv C-kodning](#)
- GNU C vs GNU C++ på objektkodsnivå (kompilerad kod)
 - [Wikipedia: Name mangling](#)
 - [Name mangling and extern C](#)
- Manualer för AVR-processorn
 - [Referensmanual ATmega32](#)
 - [Detaljerad Instruktionsmanual](#)

2. Veckoplanering

2.4. Vecka 6 - Objektorienterad modellering i C/C++ och tillståndsmodellering

- Videofilmer
 - ANSI-C Object-orientated Development
- Moment 4: Objektorienterad modellering
 - OOP med C respektive C++
 - Exempel: Strukturen som värde- och retur-parameter på maskinnivån (uppgift på laboration 3)
 - ◊ Genomgång av denna uppgift.
 - Laboration 3: Hårdvarunära programmering i C
 - Modellering av inbyggda system med OOP-metodik i C
 - Laboration 4: Objektorienterad programmering i C
 - Genomgång av laborationsuppgiften med sensorn SMT160
- Tillståndsmodellering - En programmeringsparadigm
 - Tillståndsmodellering
 - FSM-implementering
 -  LogicSignal.7z
 - ◊ Klass i C++ för signalbehandling av en logisk signal, genererar negativ och positiva flanksignaler och toggle-signaler.
 - ◊ För att kunna använda

```
enum class
```

skall kompilatorflaggan

```
-std=gnu++11
```

användas
 - <stdio> - printf/sccanf
 - ◊  PRINTF_SCANF.7z
 - ◊ Terminalprogram i Windows: TeraTerm Pro, Putty eller Hercules
 - ◊ [RS232 slides](#)
 - Temperatursensorn SMT160
 - ◊  MeasurePeriodWithPolling.7z

2. Veckoplanering

- Glöm inte kontrollskrivningarna
- Slides
 - [Datorteknik](#)
 - [Tillståndsmaskiner i UML](#)
 - [Effektiv C-kodning](#)
- Manualer för AVR-processorn
 - [Referensmanual ATmega32](#)
 - [Detaljerad Instruktionsmanual](#)

2. Veckoplanering

2.5. Vecka 7 - Avbrottshantering

- Videofilmer
 - AVR Microcontroller (AVR Interrupts) Assembly Language Tutorial 5
- Moment 5 - Avbrottshantering
 - Avbrottshantering
 - Laboration 5: Objektorienterad modellering och avbrottshantering
 - Exempel: Pulsräkning med externa avbrottet INT0
 - Exempel: Klassen time för tidstyrda uppgifter (tasks)
 - ◊  TimeClass.7z
- Glöm inte kontrollskrivningarna
- Slides
 - Datorteknik
 - Effektiv C-kodning
- Manualer för AVR-processorn
 - Referensmanual ATmega32
 - Detaljerad Instruktionsmanual

2. Veckoplanering

2.6. Vecka 8 - Projekt för betyg 4/5

- Presentation av betyg 4/5-projektet
 - Realtidsprogrammering och konstruktion av små inbyggnadssystem
 - Projekt: Styrning och övervakning i en villa
 - Treträdsbussen - 3wire
- Alla kontrollskrivningar bör vara klara
- Slides
 - Realtidsprogrammering
 - ◊ Taskabstraktionen
 - ◊ Statiska exekveringsscheman
 - Kommunikationsbussar
 - ◊ 3wire-bussen

2. Veckoplanering

2.7. Vecka 9 - Laborationer

- Laboration Tisdag och Torsdag

2.8. Vecka 10 - Laborationer

- Laboration Tisdag och Torsdag

2.9. Vecka 11 - Laborationer

- Laboration Tisdag och Torsdag.

Del II.

Assemblerprogrammering

Innehåll

3 Introduktion till grundläggande begrepp och binär aritmetik	21
3.1 Mål	21
3.2 Läsanvisningar	21
3.3 Videos	22
4 Introduktion till AVR-processorn och programutvecklingsmiljön	23
4.1 Mål	23
4.2 Läsanvisningar	24
4.3 Videos	24
5 Laboration 1: Lysdioder & strömställare	25
5.1 Programmet Hello World	26
5.2 Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare	34
5.3 Assemblerprogram för att tända och släcka en lysdiod med en switch	38
5.4 Ringräknare i assembler	40
5.5 Johnsonräknare i assembler	45
5.6 Ännu ett lysdiodsmönster	47
6 Assemblerprogrammering av AVR-processor	48
6.1 Mål	48
6.2 Läsanvisningar	49
6.3 Videos	49

INNEHÅLL

7 Laboration 2: Subrutiner och aritmetik	50
7.1 Villkorliga programsatser i assembler	51
7.2 Elektronisk tärning	53
7.3 Förändringsräknare	56
7.4 Simulerad aritmetisk enhet (Arithmetical Unit - AU)	57
7.5 Några loopar	58
8 Intel _x86 assembler	59
8.1 Mål	59
8.2 Läsanvisningar	59
9 C# och CIL (Common Intermediate Language)	60

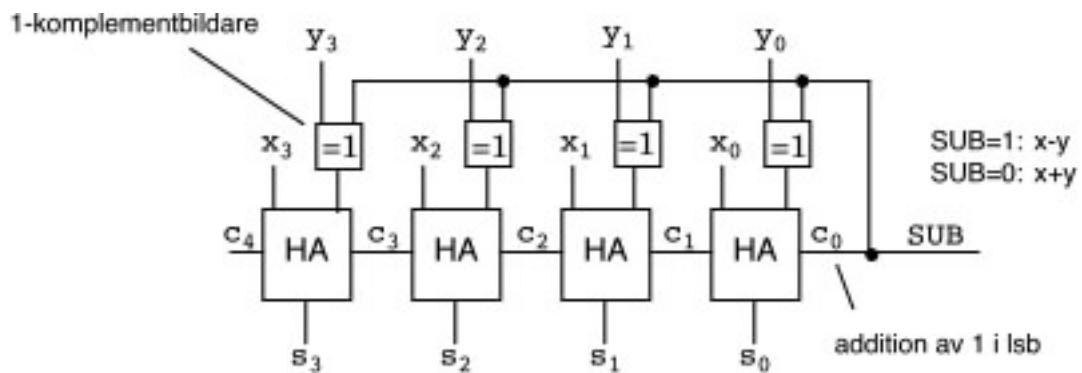
3. Introduktion till grundläggande begrepp och binär aritmetik

Innehåll

3.1 Mål	21
3.2 Läsanvisningar	21
3.3 Videos	22

3.1. Mål

Att introducera grundläggande datortekniska begrepp på maskinnivån. Att få färdighet i att arbeta med binära och hexdecimala tal och ge en introduktion till binär aritmetik.



Figur 3.1.1.: Fyra bitars adderare

3.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- Inbyggda system med AVR RISC-processorer
 - Kapitel 1: NÅGRA DATORTEKNISKA BEGREPP
 - ◊ Kapitlet behandlar grundläggande begrepp på maskinnivån för datorer.
 - ◊ Gå igenom alla uppgifter.

3. Introduktion till grundläggande begrepp och binär aritmetik

- Kapitel 2: GRUNDLÄGGANDE DIGITALTEKNIK
 - ◊ Gå igenom alla uppgifter.
- Kapitel 3: TALSYSTEM, KODER OCH BINÄR ARITMETIK
 - ◊ Kapitlet är viktigt då det tar upp grunderna för binär aritmetik och hur data bearbetas i en CPU.
 - ◊ Gå igenom alla uppgifter.
- Slides
 - [Datorteknik](#)

3.3. Videos

Figur 3.3.1.: Video - Binary, Decimal and Hexadecimal Number Systems

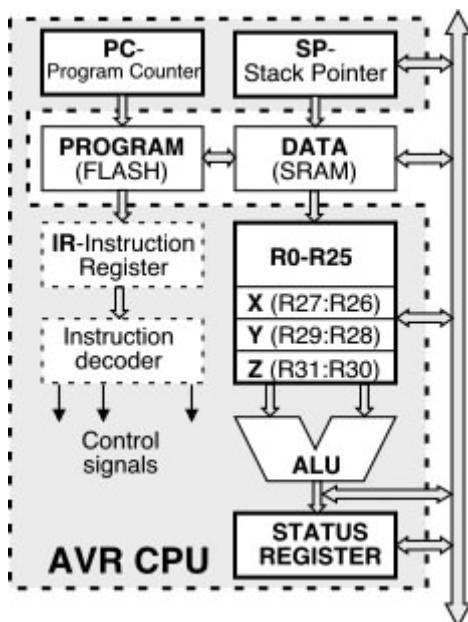
4. Introduktion till AVR-processorn och programutvecklingsmiljön

Innehåll

4.1	Mål	23
4.2	Läsanvisningar	24
4.3	Videos	24

4.1. Mål

Att få en förståelse och känsla för hur beräkningar sker på maskinnivå i en processor, samt att presentera programmerarens modell av AVR-processorn på maskinnivån.



Figur 4.1.1.: Programmerarens modell av AVR-processorn på maskinnivån

4. Introduktion till AVR-processorn och programutvecklingsmiljön

4.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- [Inbyggda system med AVR RISC-processorer](#)
 - Kapitel 4: AVR 8 BITARS DATORER
 - ◊ Kapitlet behandlar datorarkitektur och speciellt datorarkitekturen för AVR-processorn.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 5: INSTRUKTIONER SOM AVR-PROCESSORN KAN UTFÖRA
 - ◊ Grundläggande programmering av processorn på maskinnivå behandlas i detta kapitel. Olika typer av instruktioner som processorn kan utföra behandlas.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 6: DIGITALA IN- OCH UTSIGNALER
 - ◊ Olika typer av portar finns som gör att CPU:n kan stå i kontakt med en omvärld. De mest grundläggande portarna är de för digitala in- och utsignaler. I kapitlet behandlas också hur lysdioder och tryckknappar kan anslutas till digitala portar.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 7: GNU ASSEMBLATOR/LÄNKARE
 - ◊ Ett assemblerprogram skrivit för en processor på maskinnivån byggs upp med hjälp av assemblerdirektiv och instruktioner. I kapitlet går igenom det mest grundläggande för assemblatorn GNU as.
 - ◊ Gå igenom alla uppgifter
- [Referensmanual ATmega16](#)
- [Detaljerad Instruktionsmanual](#)
- Slides
 - [Datorteknik](#)

4.3. Videos

- [Microcontroller Tutorial - A Beginners Guide](#)
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

5. Laboration 1: Lysdioder & strömställare

Innehåll

5.1	Programmet Hello World	26
5.2	Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare	34
5.3	Assemblerprogram för att tända och släcka en lysdiod med en switch	38
5.4	Ringräknare i assembler	40
5.5	Johnsonräknare i assembler	45
5.6	Ännu ett lysdiodsmönster	47

Att kontrollera : Om du använder en portpinne som Digital Utgång (DO) shall motsvarande bit i datariktningsregistret (DDR) vara satt till 1

Mål

- Att introducera utvecklingsmiljön baserad på GNU assembler, AVRstudio4, hårdvarukortet STK500 bestyckad med en AVR RISC-processor av typen ATmega16/32.
- Att få en första insikt i hur en given processor och dess registerarkitektur kan utnyttjas vid maskinnära programmering.
- Att få en förståelse för cyklisk programexekvering för att avläsa strömställare och att uppdatera lysdioder. Den cykliska programexekveringen kan sägas vara en av grundstenarna i realtidsprogrammeringstekniken.

Läsanvisningar, tips och uppgifter att göra

- Kompendiet: [Inbyggda system med AVR RISC-processorer](#)
 - Kapitel 5: NÅGRA DATORTEKNISKA BEGREPP INSTRUKTIONER SOM AVR-PROCESSORN KAN UTFÖRA
 - Kapitel 6: DIGITALA IN- OCH UTSIGNALER
 - Kapitel 7: GNU ASSEMBLATOR/LÄNKARE
- Startfiler

5. Laboration 1: Lysdioder & strömställare

-  main.c
-  Subrutiner.S
-  wait_ms.S

5.1. Programmet Hello World

I det följande vill visa på arbetsgången för att skapa ett nytt programprojekt, koda programmet, kompilera och länka programmet och sedan sist ladda ner programmet till STK500-kortet (bestyckat med en ATmega16/32). Arbetsgången är följande:

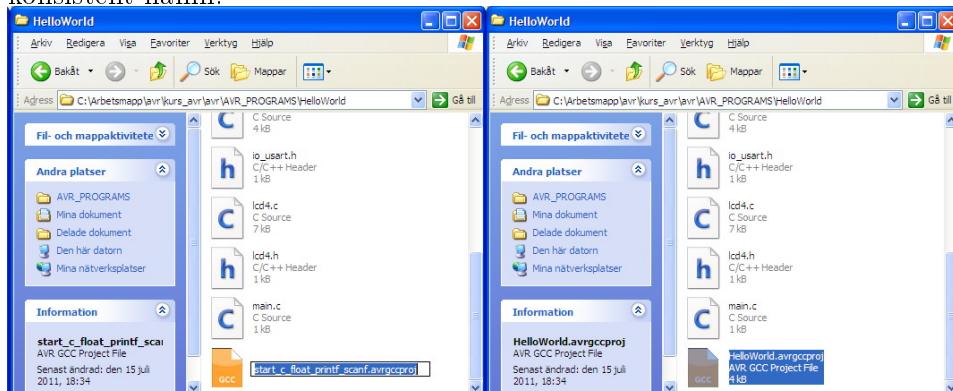
1. Börja med att ladda ner den zippade filen som innehåller katalogen med olika startprojekt i C respektive assembler

Startprojekt och exempelprogram

Tag en kopia av katalogen "**start_c_float_printf_scanf**" döp om denna till "HelloWorld".

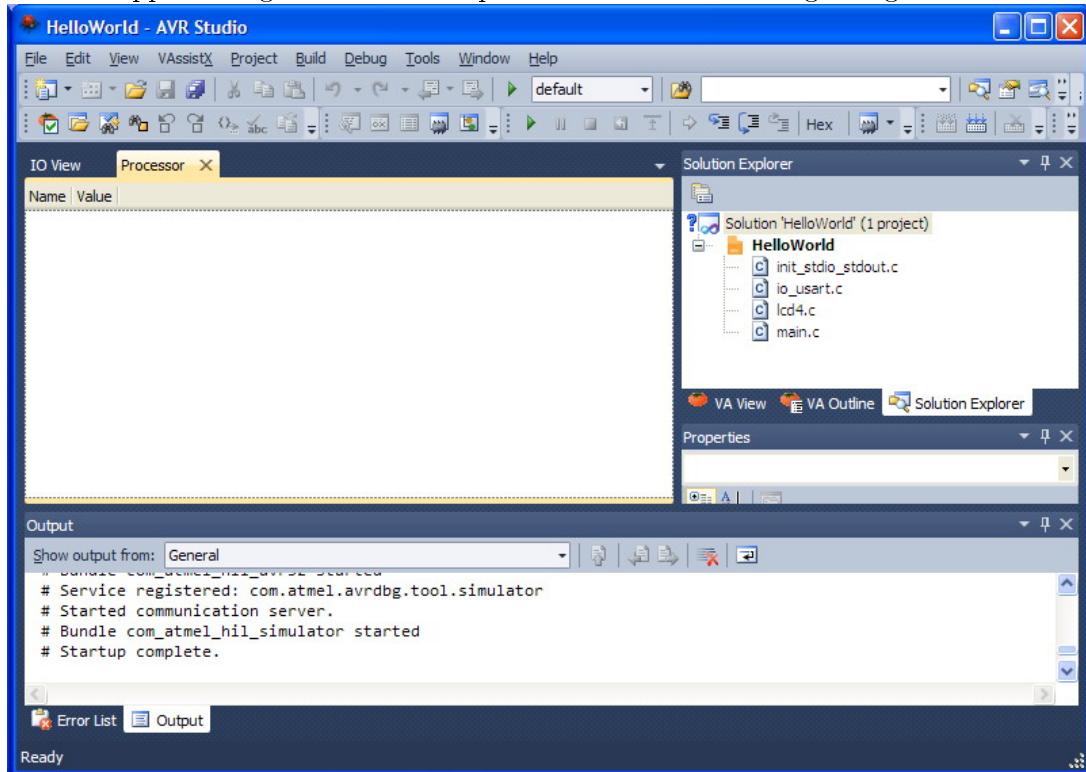


2. Öppna katalogen som nu skall heta HelloWorld och döp om projektfilen från "start_c_float_printf_scanf.avrgccproj" till "HelloWorld.avrgccproj". Detta är nödvändigt för att projektet i AVRstudio6 skall representeras med ett konsistent namn.



5. Laboration 1: Lysdioder & strömställare

3. Öppna projektet genom att dubbelklicka på den omdöpta projektfilen "HelloWorld.avrgccproj" eller att öppna den genom att klicka på ikonen med musens högertangent



AVRstudio5 bör nu ha startats upp.

5. Laboration 1: Lysdioder & strömställare

4. Mata in källkoden för att skriva ut "Hello World" på några olika språk. Dubbelklicka på "main.c" i subfönstret "Solution Explorer" för att öppna denna källkodsfil för redigering. Tag bort all kod som finns ursprungligen, behåll bara den oändliga while-loopen. Lägg in programkod för att skriva ut "Hello World" på några olika sätt.

The screenshot shows the AVR Studio interface. The main window displays the code for `main.c` in the `main.while` tab:

```
#include "lcd4.h"

int main()
{
    while ( 1 )
    {
        printf("Hello World");
        printf("Hej, Verden")
        printf("Saluton mondo")
    }

    return 0;
}
```

The `Solution Explorer` window on the right shows the project structure:

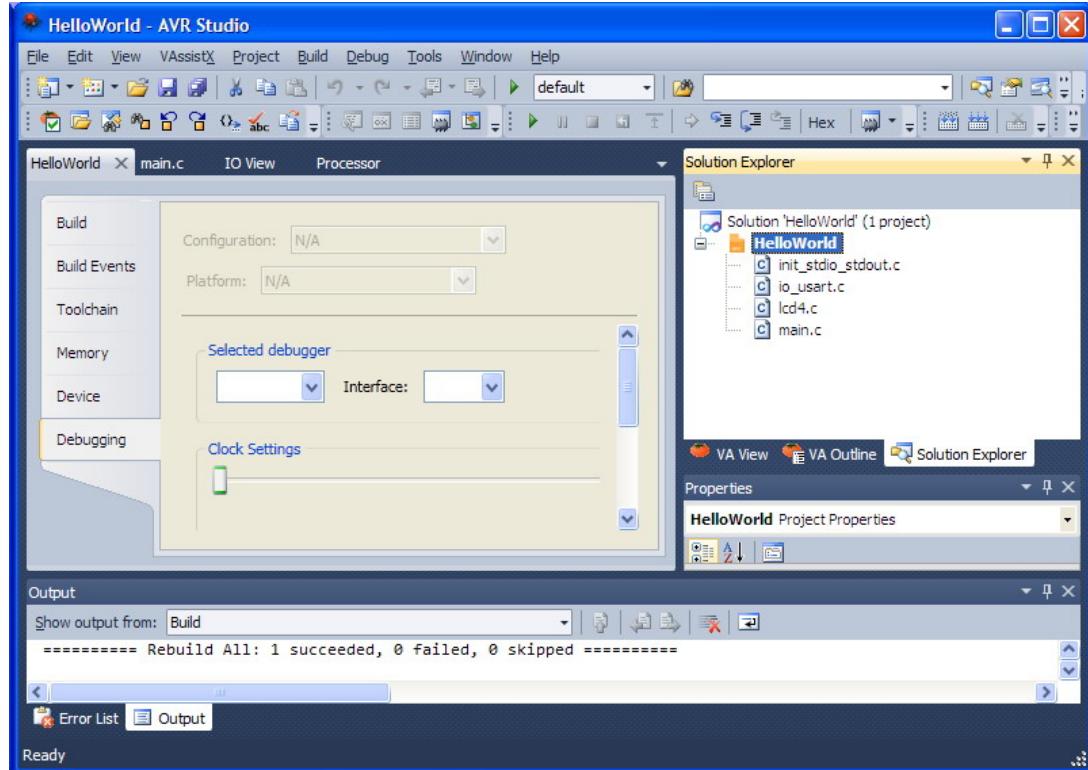
- Solution 'HelloWorld' (1 project)
 - HelloWorld
 - init_stdio_stdout.c
 - io_usart.c
 - lcd4.c
 - main.c

The `Output` window at the bottom shows the build log:

```
# Building com.atmel.111_avr32_debug
# Service registered: com.atmel.avrdbg.tool.simulator
# Started communication server
```

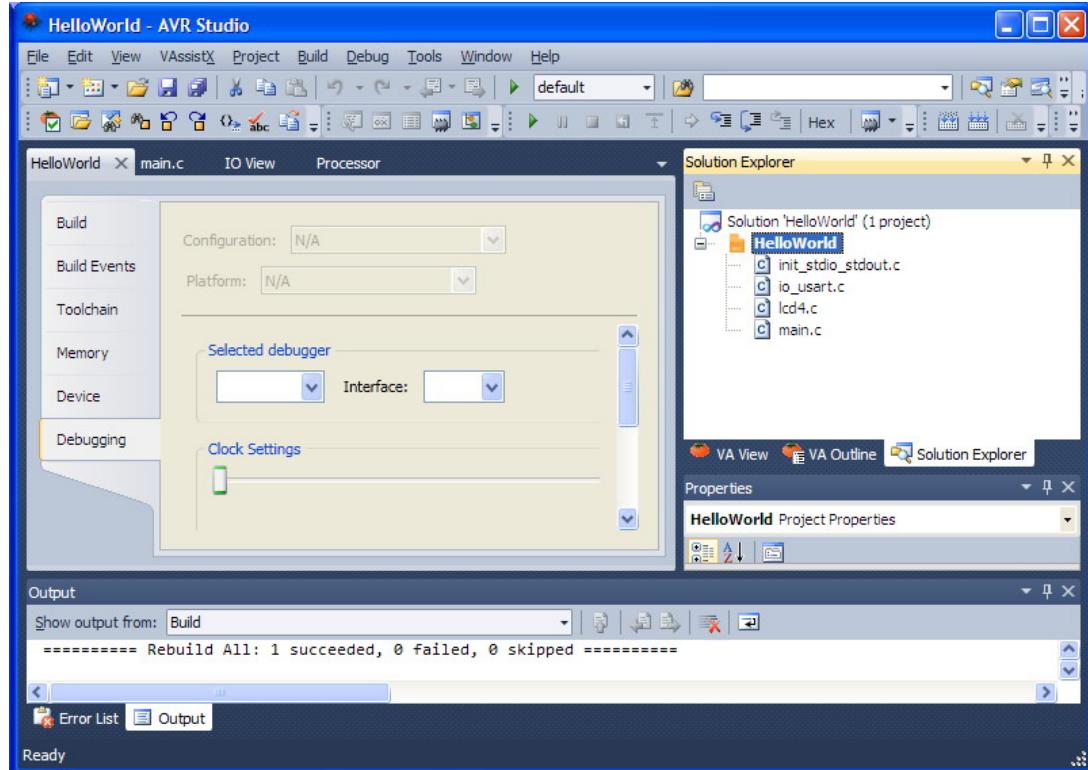
5. Laboration 1: Lysdioder & strömställare

5. Bygg ihop ett körbart program genom att gå till menyn Build och göra Rebuild HelloWorld. Kontrollera i subfönstret "Output" att kompileringen och länkningen av projektet medförde en exekverbar fil som utdata. Kolla av att du ej fick några syntaxfel som måste rättas. Om fel har uppstått kollar du i subfönstret "Error list".

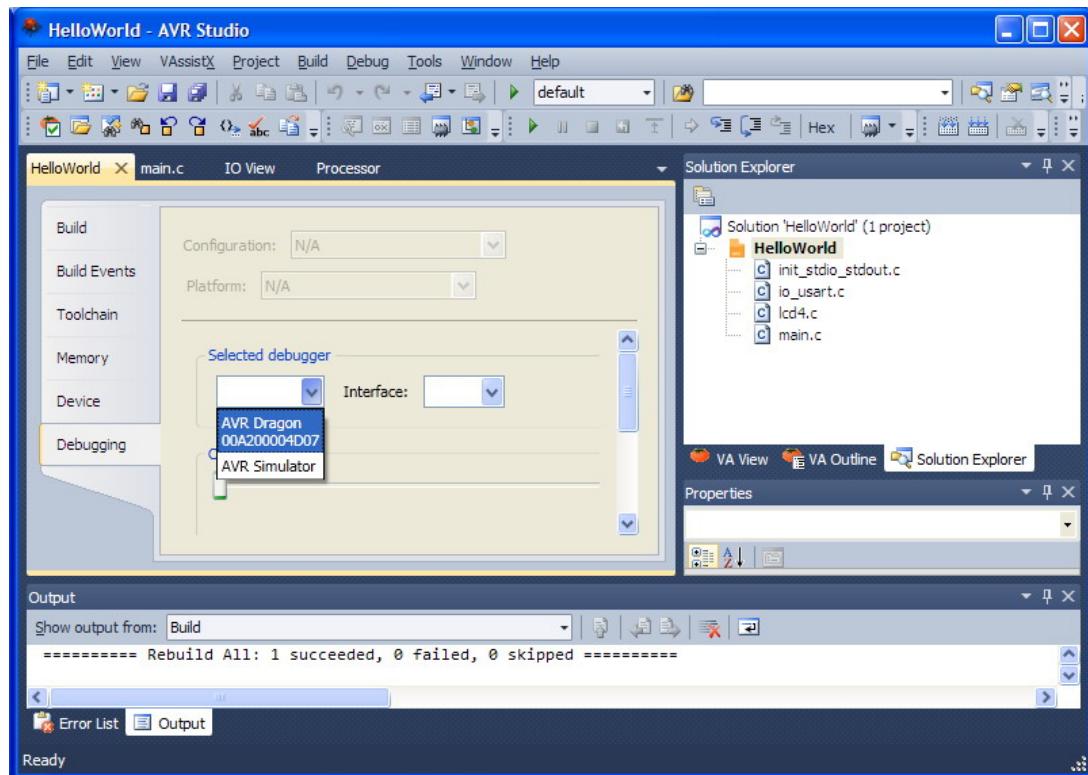


5. Laboration 1: Lysdioder & strömställare

6. Öppna projektets egenskapsfönster genom att i "Solution Explorer" med musenpekaren över projektnamnet "HelloWorld" använda högertangenten och ta fram kontextmeny och göra menyvalet Properties.

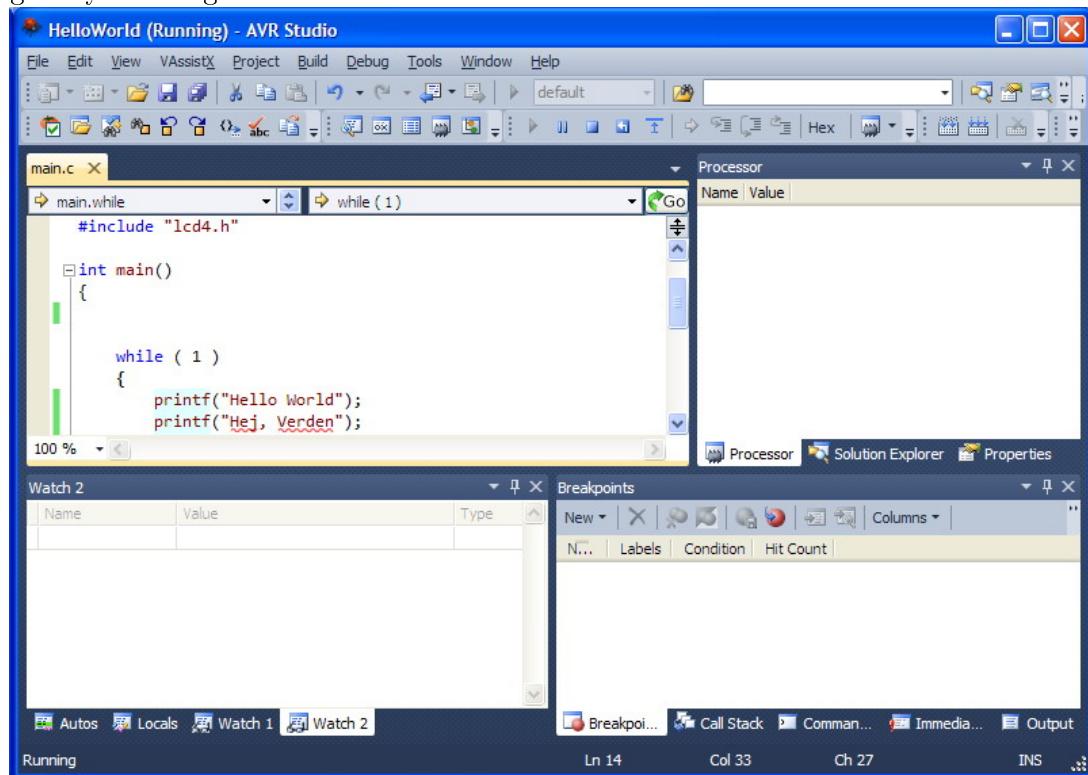


5. Laboration 1: Lysdioder & strömställare



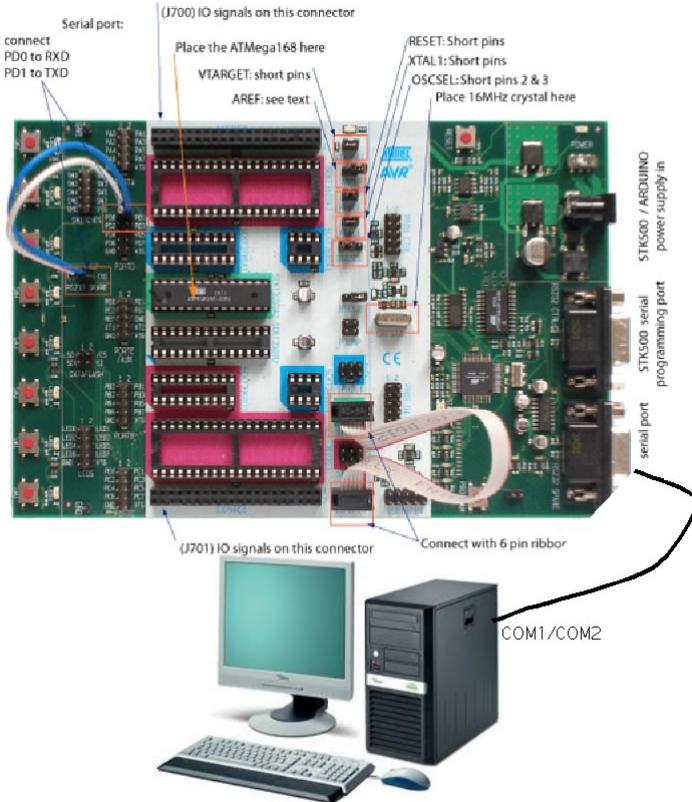
5. Laboration 1: Lysdioder & strömställare

7. Ladda nu ner programmet till datorkortet genom att trycka på ikonen i form av en grön fylld triangel.



5. Laboration 1: Lysdioder & strömställare

8. För att du skall kunna se något skall COM1/COM2-porten på din persondator vara ansluten till STK500 RS232SPARE-kontakten. Förutom detta skall du starta upp ett terminalprogram (TeraTerm Pro eller Terminal) i din persondator och se till att kommunikationsinställningarna är 9600 baud, 1 startbit, ingen paritetsbit och 1 stoppbit.



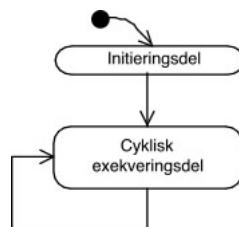
5. Laboration 1: Lysdioder & strömställare

5.2. Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare

I uppgiften som följer skall vi gå igenom alla moment för att skriva ett program för AVR-processorn, kompilera/assemblera och länka programmet och sist ladda ner programmet till STK500-kortet.

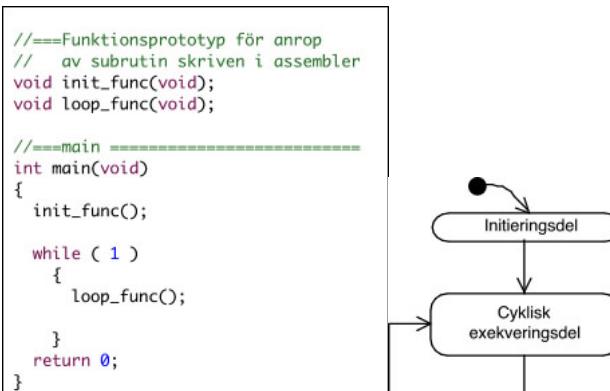
Strukturen på våra assemblerprogram

Ett typiskt program i en mikrostyrenhet (micro controller unit) består av en initieringsdel och en cyklig del. I initieringsdelen initierar vi variabler och olika I/O-enheter som används. I den cykliska delen har vi en oändlig loop där vi exekverar tillämpningens programkod cyklistiskt, upprepning av exekveringen med tidsintervallet T .



Figur 5.2.1.: Våra program är små realtidssystem med en enkel cyklig loop

I de två första laborationerna (totalt 6 laborationer) skall vi programmera processorn med program skrivna i assembler. Vi kommer att starta upp assemblerprogrammet från main-funktionen i C, detta sker via två subrutiner som skall skrivas helt i assembler. Subrutinerna kan vi anropa från C som två funktioner. Vår main-funktion kommer alltid att ha följande utseende:



Figur 5.2.2.: Subrutiner/funktioner i assembler startas upp via en main-funktion i C

Programmet i main-funktionen består i att först anropa en funktion/subrutin med namnet `init_func()`. I denna subrutin sker initieringen av vår applikation, denna består

5. Laboration 1: Lysdioder & strömställare

normalt i att vi skall initiera datariktningsregister för digitala in- och utsignaler samt att vi initierar diverse variabler.

Vårt assemblerprogram kommer att finnas i en källkodsfil som heter "subrutiner.s". I denna fil skriver vi vårt assemblerprogram i form av två subrutiner: init_func och loop_func I filen finns också makrokonstanter för PORTA till PORTD definierade.

```
;---- I/O-adresser for Port D ---
#define PIND 0x10
#define DDRD 0x11
#define PORTD 0x12
;---- I/O-adresser for Port C ---
#define PINC 0x13
#define DDRC 0x14
#define PORTC 0x15
;---- I/O-adresser for Port B ---
#define PINB 0x16
#define DDRB 0x17
#define PORTB 0x18
;---- I/O-adresser for Port A ---
#define PINA 0x19
#define DDRA 0x1A
#define PORTA 0x1B

;---- subrutin init_func ---
.global init_func

.data

.text

init_func:

RET

;---- subrutin loop_func ---
.global loop_func
.text
loop_func:

RET
```

Figur 5.2.3.: Källkodsfilen i assembler - "subrutiner.S"

När du skall skriva ett nytt assemblerprogram börjar du alltid med att kopiera startkatalogen för assemblerprogram: "**Start assembler**". Ändra sedan namn på denna.

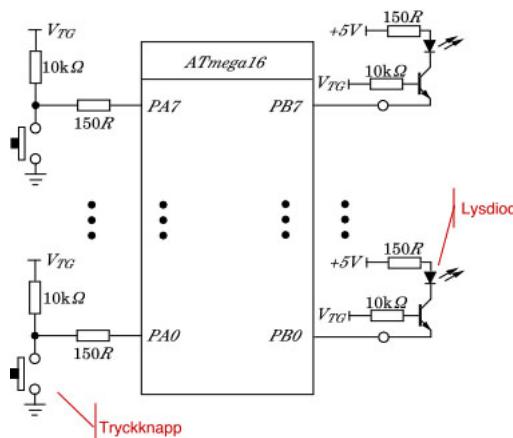
5. Laboration 1: Lysdioder & strömställare

Förslagsvis enligt följande nomenklatur där vi anger vilken laboration och uppgift vi gör. Antag som exempel att vi håller på med laboration 1 och uppgift 2, namnet på programkatalogen skall då vara "Lab_1_2".

Problemförformulering

Följande hårdvaruuppkoppling gäller: Åtta tryckknappar är anslutna till PORTA och åtta lysdioder till PORTB. Uppgiften är att låta var och en av tryckknapparna styra om korresponderande lysdiod skall vara tänd eller släckt. På STK500-kortet finns 8 tryckknappar märkta SW7,..,SW0 och 8 lysdioder märkta LED7,..,LED0. Tryckknapparna kopplas ihop med PORTA via en flatkabel, på liknande sätt kopplas lysdiodeerna ihop med PORTB.

Problemet i denna uppgift består i att skriva ett program som kopierar switcharnas värden på PORTA och lägger ut dessa på lysdiodeerna som finns på PORTB.



Figur 5.2.4.: Uppkoppling för PORTA och PORTB hos ATmega16/32-processorn

Vilka register kan användas i processorn?

Använd registrena R18-R27 helt fritt i dina subrutiner som du skriver i assembler, om vi gör på detta sättet kommer detta att innebära att vi följer GNU C-kompilatorns konventioner för användning av register i en funktion/subrutin. Detta innebär i slutändan att våra subrutiner kan anropas från en C-funktion som en vanlig C-funktion.

De flesta av våra uppgifter är så pass små att vi klarar oss mycket väl på dessa 10 register.

Några frågor att besvara

1. När vi trycker på en tryckknapp, vilket logiskt värde (0/1) har en aktiv nedtryckning?
2. För vilket logiskt värde 0 (0V) eller 1 (5V) tänds en lysdiod?

5. Laboration 1: Lysdioder & strömställare

Tips för att lösa problemet

- Pseudo-kodslösning i C

```
void init_func()
{
    DDRA=0x00; //PA7-PA0 are inputs
    DDRB=0xFF; //PB7-PB0 are outputs
}

void loop_func()
{
    //Copy PINA (Port INput A) to PORTB
    PORTB = PINA;
}
```

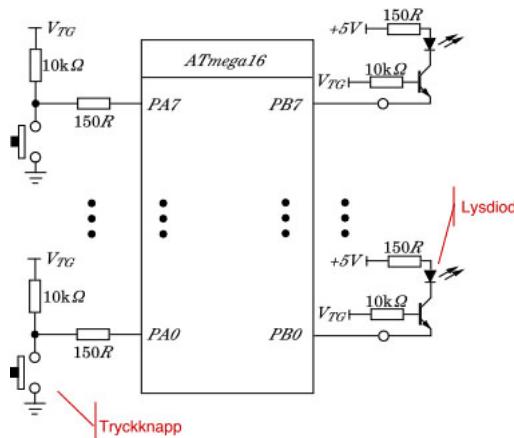
- En tilldelningsats i C blir en tvåstegsoperation i assembler
 - 1. Ladda ett register (R18-R27) med en konstant (LDI) eller genom att läsa ett IO-register (IN)
 - 2. Spara värdet i registret (R18-R27) till ett IO-register (OUT)
- Använd I/O-instruktionerna IN, OUT samt instruktionen LDI.

5. Laboration 1: Lysdioder & strömställare

5.3. Assemblerprogram för att tända och släcka en lysdiod med en switch

Problemformulering

Skriv ett program som kopierar switchen SW3:s (PA3) värde till lysdioden LED0 (PB0), övriga lysdioder LED1 till LED7 skall alltid vara släckta.



Figur 5.3.1.: Uppkoppling för PORTA och PORTB hos ATmega16/32-processorn

Vilka register kan användas i processorn?

Använd registrena R18-R27 helt fritt i dina subrutiner som du skriver i assembler, om vi gör på detta sättet kommer detta att innehålla att vi följer GNU C-kompilatorns konventioner för användning av register i en funktion/subrutin. Detta innebär i slutändan att våra subrutiner kan anropas från en C-funktion som en vanlig C-funktion.

De flesta av våra uppgifter är så pass små att vi klarar oss mycket väl på dessa 10 register.

Tips för att lösa problemet

Prova gärna att lösa detta problem på flera olika sätt, det ger insikt och kunskap i hur processorn fungerar.

1. Följande pseudokodslösning i C är en lösning på problemet. Försök att översätta detta till assemblerkod.

5. Laboration 1: Lysdioder & strömställare

```
void init_func()
{
    DDRA = 0x00; //PA7-PA0 are inputs
    DDRB = 0xFF; //PB7-PB0 are outputs
    PORTB = 0xFF; //The LEDs are off
}

void loop_func()
{
    // The program as one C-statement
    //     PORTB = ((PIN & 0x08) >> 3) | (PORTB & 0xFE);

    char R20; //Register R20 is used as a local variable
    char R21; //Register R20 is used as a local variable

    R20 = PIN & 0x08; //Mask out bit PA3
    R20 = R20 >> 3; //Shift right three bits
    R21 = PORTB & 0xFE;
    R20 = R20 | R21;
    PORTB = R20;
}
```

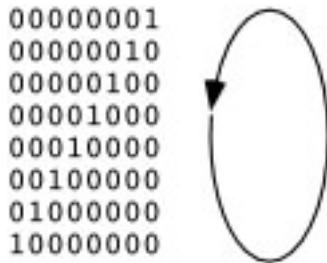
- a) Använd instruktionerna IN, ANDI, LSR, OR och OUT.
2. Ett alternativ lösning kan göras med bit-instruktionerna SBIC, SBI, SBIS och CBI.
3. En annan lösning kan göras med BST- och BLD-instruktionerna.

5.4. Ringräknare i assembler

Denna uppgift skall lösas med hjälp av ett assemblerprogram.

Problemformulering

Skriv ett program som implementerar en 8 bitars ringräknare. Ringräknarens värde skall visas på 8 lysdioder anslutna till PORTB.



Figur 5.4.1.: Bitmönster för en ringräknare

Tidsfördröjning med hjälp av en vänteloop

Vi har ofta behov av att kunna utföra uppgifter periodiskt i vår mikroprocessor. Detta innebär att vi vill exekvera samma programkod med jämna tidsmellanrum ett typexempel på detta är då vi blinkar en lysdiod. Frågan är hur åstadkommer vi detta på enklast sätt?

En lösning på problemet är att utnyttja att varje varje instruktion tar en viss tid att utföra. Om vi klockar vår processor med 8MHz så är periodtiden för denna 125 nanosekunder, vi brukar säga att en maskincykel är på 125 nanosekunder. En instruktion tar 1 till 4 maskincykler att utföra, de flesta instruktionerna tar en maskincykel att utföra. I figuren som följer visas på en subrutin med namnet wait_ms som realiseras fördröjningar i intervallet 1 till 65000 millisekunder, subrutinen arbetar med en vänteloopsfördröjning. Passningen av fördröjningstiden sker i registerparet R25:R24 innan anropet sker.

5. Laboration 1: Lysdioder & strömställare

```
.text
;;;;=Subroutine wait_milliseconds=====
;; C-prototype: void wait_milliseconds(int milliseconds);
;; Parameter 1 is passed in registerpair R24:R25
.global wait(milliseconds)
wait(milliseconds:
    wait(milliseconds_loop:
        RCALL    wait1ms
        SBIW    R24,1           ;Subtract R25:R24=R25:R24-1
        BRNE    wait(milliseconds)_loop
    wait(milliseconds_end:
        RET
    );
    ;=====Subroutine wait1ms=====
    ; C-prototype: void wait1ms(void);
    .global wait1ms
wait1ms:
    RCALL    wait500microseconds
    RCALL    wait500microseconds
    RET
    ;=====Subroutine wait250microseconds=====
wait250microseconds:   ; 16MHz clock <-> 1 machine cycle = 62.5 ns
wait500microseconds:  ; 8MHz clock <-> 1 machine cycle = 125 ns
    LDI     R18, 210
    ; Register R25-R18 can be used as scratch pad register
    ; due to the GNU C calling conventions.
wait_loop:
    LD      R19,X
    DEC    R18
    BRNE    wait_loop
    RET
```

Figur 5.4.2.: Subrutinen wait_milliseconds

5. Laboration 1: Lysdioder & strömställare

I ett assemblerprogram kan nu denna subrutin anropas. Antag att vi vill ha en tidsfördröjning på 500 millisekunder, vi skriver om detta på hexadecimal form som 0x01F4.

```
LDI R24,0xF4 ;Minst signifikanta byten till R24
LDI R25,0x01 ;Mest signifikanta byten till R25
CALL wait_ms
```

Subrutinen kan också anropas från en C-funktion som en vanlig funktion, prototypen för funktionen blir då:

```
void wait_ms(unsigned int ms);
```

Ett anrop från C ser då ut på följande sätt om vi vill fördröja 500 millisekunder:

```
wait_ms(500);
```

Tips

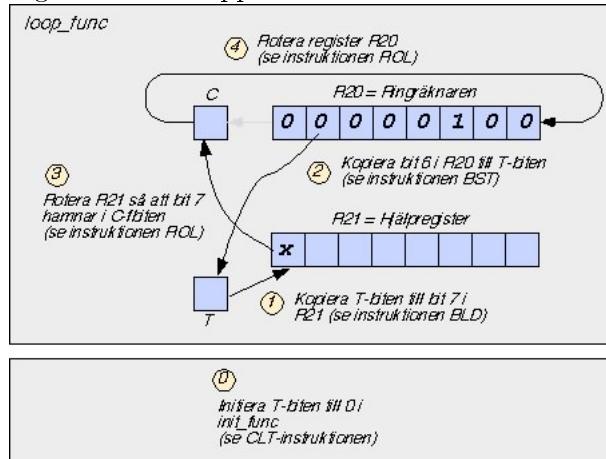
Tre typer av lösning är tänkbara:

1. Lösning 1: Använd processorns register på något listigt sätt

- a) Utnyttja processorns register på något smart sätt. Idealt vore att kunna ha ringräknaren i ett av processorns register och sedan bara rotera detta register.



- b) Tyvärr så finns ingen instruktion som utför det ovanstående figur visar. Där emot så kan statusregistret SREG utnyttjas tillsammans med ett beräkningsregister för att uppnå samma sak:



Lösningen bygger på att utföra ovanstående 3 operationer varje gång som loop_func exekveras. T-biten i statusregistret kan användas för att lagra en bit temporärt.

5. Laboration 1: Lysdioder & strömställare

2. Lösning 2: Lägg upp en mönstertabell

```
char RING1_LEDS[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };

void init_func()
{
    i=0;

    DDRB=0xFF; //PB7-PB0 are programmed as outputs
}

void loop_func()
{
    PORTB = ~ RING1_LEDS[i];

    i++;
    i= i & 0x07;

    wait_milliseconds(500);

}
```

- a) Studera följande pseudokodslösning i C, försök sedan att implementera denna i assembler.
- b) I C finns ett antal bit- och skiftoperatorer som används i pseudo-koden:

i. Bitand-operatorn &

```
char x = 0b11110000;
x = x & 0b00111100; //variabeln x har värdet 0b00110000 efter tilldelningssatsen.
```

ii. BitOr-operatorn |

```
char x = 0b11110000;
x = x | 0b00111100; //variabeln x har värdet 0b11111100 efter tilldelningssatsen.
```

iii. BitInvertering ~

```
~0b00001111 är detsamma som 0b11110000
~0xAA är detsamma som 0x55
```

iv. Vänsterskiftoperatorn <<

```
char x=0b00001100;
x = x << 1; //Vänsterskift ett steg, har värdet 0b00011000 efter tilldelningssatsen.
```

5. Laboration 1: Lysdioder & strömställare

v. Högerskiftooperatorn $>>$

```
char x=0b00001100;
x = x >>1; //Högerskift ett steg, har värdet 0b00000110 efter
tilldelningssatsen.
```

3. Lösning 3: If-else-sats

- a) Skifta runt en 1:a, bivillkor är att när du skiftat ut 1:an måste du initiera om till 0x01. Denna lösning innehåller en if-sats för att initiera om räknaren.

Test av ett program i simulatorn

Ett alternativ vid programutveckling är att provköra sitt program i en simulator, d.v.s. ett program som simulerar en AVR ATmega16/32-processor. Fördelarna med detta är att vi kan felsöka och upptäcka logiska fel i vårt program på ett mer flexibelt sätt än genom att ladda ner programmet och kontrollera att det fungerar.

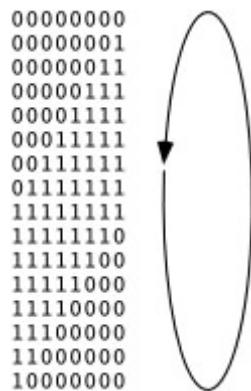
5. Laboration 1: Lysdioder & strömställare

5.5. Johnsonräknare i assembler

Denna uppgift skall lösas med hjälp av ett assemblerprogram.

Problemformulering

Skriv ett program som implementerar en 8 bitars Johnsonräknare. Johnsonräknare värde skall visas på 8 lysdioder anslutna till PORTB.



Figur 5.5.1.: Bitmönster för en johnsonräknare

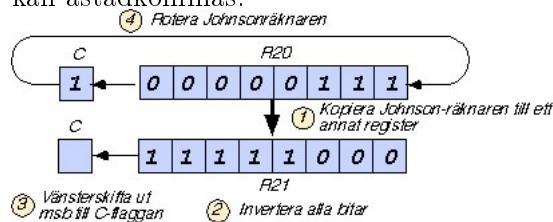
Studera mönstret. När mest signifikanta biten är 0 skiftas en 1:a in och om mest signifikanta biten är 1 skiftas en 0:a in.

Tips

- Lösning 1: Använd processorns register på något listigt sätt
 - Principen för en Johnsonräknare visas i figuren som följer:



- Hur implementerar vi inverteringen av mest signifikanta biten för inskiftning till den minst signifikanta positionen? Figuren som följer visar i 4 steg hur det kan åstadkommas.



- Lösning 2: Lägg upp en mönstertabell

5. Laboration 1: Lysdioder & strömställare

- Lösning 3: If-else-sats

Studera följande pseudokodslösning i C, försök sedan att implementera denna i assembler. Denna lösning kräver att du använder villkorliga hopp som behandlas i nästa assemblerlaboration".

```
char vjohn;
```

```
void init_func()
{
    DDRB=0xFF; //PB7-PB0 are programmed as outputs
    vjohn=0x00; //R20 is used as vjohn
}

void loop_func()
{
    if ((vjohn & 0x80)==0)
        vjohn=(vjohn<<1)+1;
    else
        vjohn=vjohn<<1;

    PORTB=~vjohn;

    wait_milliseconds(500);
}
```

5. Laboration 1: Lysdioder & strömställare

5.6. Ännu ett lysdiodsmönster

Skriv ett assemblerprogram som genererar följande mönster:

```
00000000  
10000001  
11000011  
11100111  
11111111  
11100111  
11000011  
10000001
```

6. Assemblerprogrammering av AVR-processor

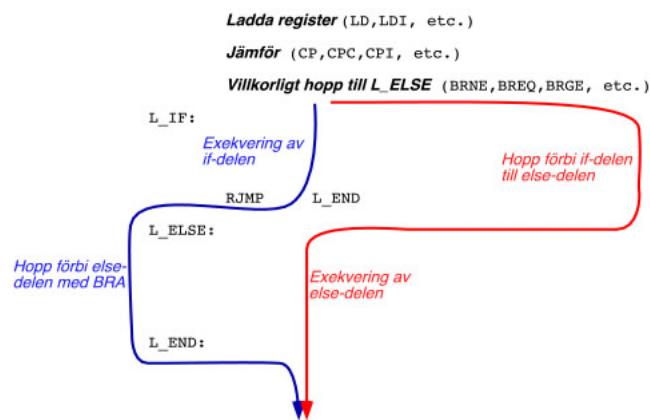
Innehåll

6.1	Mål	48
6.2	Läsanvisningar	49
6.3	Videos	49

6.1. Mål

Några mål i denna modul:

- Att få insikt om hur en högnivåkompilator som GNU gcc utnyttjar processorns register för olika ändamål.
- Att förstå hur högnivåstrukturer som if-else, while-loopar och funktionsanrop implementeras på maskinnivån.
- Att förstå hur parameteröverföring till en funktion/subrutin sker på maskinnivån.



Figur 6.1.1.: Implementering av if-then-else på maskinnivån

6. Assemblerprogrammering av AVR-processor

6.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- [Inbyggda system med AVR RISC-processorer](#)
 - Kapitel 8: C OCH ASSEMBLER
 - ◊ Hur utnyttjar en högnivåkompilator en given datorarkitektur för kunna realisera en exekvering av högnivåprogrammet på en processor? I kapitlet tas det upp lite om GNU C kompilatorns konventioner för att använda processorns 32 register.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 9: VILLKORLIGA OCH REPETETIVA PROGRAMSATSER I ASSEMBLER
 - ◊ Hur realiseras if-else-satser och while-satser på maskinnivå, i detta kapitel gårs detta igenom.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 10: MODULÄR PROGRAMMERING MED SUBRUTINER
 - ◊ I detta kapitel behandlas funktioner (subrutiner) och parameteröverföring och hur detta görs det på maskinnivå?
 - ◊ Gå igenom alla uppgifter.

6.3. Videos

- [Microcontroller Tutorial - A Beginners Guide](#)
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

7. Laboration 2: Subrutiner och aritmetik

Innehåll

7.1	Villkorliga programsatser i assembler	51
7.2	Elektronisk tärning	53
7.3	Förändringsräknare	56
7.4	Simulerad aritmetisk enhet (Arithmetical Unit - AU)	57
7.5	Några loopar	58

Att kontrollera : Om du använder en portpinne som Digital Utgång (DO) shall motsvarande bit i datariktningregistret (DDR) vara satt till 1

Mål

- Att få en förståelse för den grundläggande heltalsaritmetiken i en mikroprocessor.
- Att kunna modularisera assemblerprogrammet med hjälp av subrutiner och att kunna använda globala variabler för parameteröverföring
- Att få en förståelse för hur processorns register och stacken används för parameteröverföring till subrutiner/funktioner och hur processorns register och stacken används för lokala variabler.

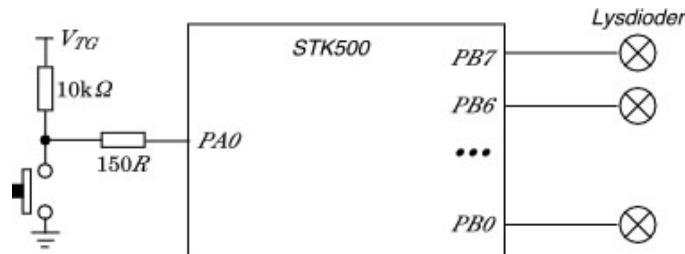
Läsanvisningar, tips och uppgifter att göra

- Kompendiet [Inbyggda system med AVR RISC-processorer](#)
 - Kapitel 8: C OCH ASSEMBLER
 - Kapitel 9: VILLKORLIGA OCH REPETETIVA PROGRAMSATSER I ASSEMBLER
 - Kapitel 10: MODULÄR PROGRAMMERING MED SUBRUTINER
- [Referensmanual ATmega16](#)
- [Detaljerad Instruktionsmanual](#)

7.1. Villkorliga programsatser i assembler

Problemformulering

Skriv ett program där man med hjälp av en switch kan välja mellan att blinka 8 lysdioder antingen i form av en ringräknare eller i form av en Johnsonräknare. Använd switch SW0 (PA0) för att växla mellan de två räknarna.



Figur 7.1.1.: Koppling av tryckknapp och lysdioder till portar

Vilka register kan användas i processorn?

Använd registrena R18-R27 helt fritt i dina subrutiner som du skriver i assembler, om vi gör på detta sättet kommer detta att innebära att vi följer GNU C-kompilatorns konventioner för användning av register i en funktion/subrutin. Detta innebär i slutändan att våra subrutiner kan anropas från en C-funktion som en vanlig C-funktion.

De flesta av våra uppgifter är så pass små att vi klarar oss mycket väl på dessa 10 register.

Tips för att lösa problemet

1. Utgå ifrån föregående laboration där vi har konstruerat både en ringräknare och en Johnsonräknare. Paketera koden i var sin subrutin. Om du arbetar med registren som lokala variabler måste dessa vara olika för Johnson- respektive Ring-räknaren med avseende på registeret som håller räknarens värde.
2. Studera följande pseudokodslösning i C, försök sedan att implementera denna i assembler.

```
char vring;

char vjohn;

void init_func()
{
    DDRB=0xFF; //PB7-PB0 are programmed as outputs
    vjohn=0x00;
}

void loop_func()
{
    if ( (PINA & 0x01) == 0 )
        ring();
    else
        john();
}

void ring()
{
    vring=vring<<1

    if (vring==0x00)
        vring=0x01;

    PORTB = ~vring;
}

void john()
{
    if ((vjohn & 0x80)==0)
        vjohn=(vjohn<<1)+1;
    else
        vjohn=vjohn<<1;

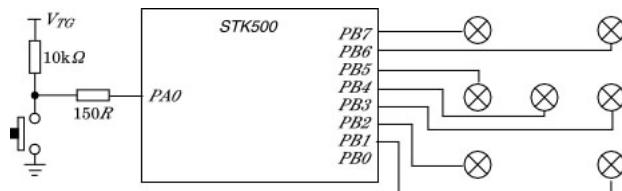
    PORTB = ~vjohn;
}
```

7.2. Elektronisk tärning

Problemformulering

I denna uppgift skall du göra en elektronisk tärning. Tyvärr så kan vi ej ordna lysdioderna som ligger på en rad i en matris enligt figuren. Detta skall ej hindra oss, vi kan låtsas att dioderna är i en matris och anslutna till PORTB enligt figuren.

Din uppgift är nu att förverkliga en elektronisk tärning med en kastknapp som är ansluten till PORTA och PA0.



Figur 7.2.1.: Elektronisk tärning

Tips för att lösa problemet

1. Lägg upp en mönstertabell för de 6 möjliga lysdiodsmönstren. Inför en räknarvariabel i form av en 1 till 6 räknare alternativt 0 till 5 räknare. Tärningen har 6 tillstånd hur vi representerar dem spelar ingen större roll. Studera följande pseudokodlösning i C, försök sedan att implementera denna i assembler. Korrekt mönstertabell i pseudokodlösningen skall vara 0x10,0x82,0x92,0xc6,0xd6,0xee (ändring 040203).

7. Laboration 2: Subrutiner och aritmetik

```
char pattern[6]={0x08,0x82,0x52,0xC6,0xD6,0xEE};

void init_func()
{
    DDRB=0xFF; //PB7-PB0 are programmed as outputs

    counter=0x00; //R20 is used as vjohm
}

void loop_func()
{
    if ((PIN_A & 0x01) == 0 )
        dice_update();
}

void dice_update()
{
    counter++;

    if (counter == 6 )
        counter=0;

    PORT_B = ~pattern[counter];
}
```

2. Problem med WinAVR-kompilatorn från och med 2009 Ett problem (FEL) har dykt upp som gör att initierade variabler/tabeller/vektorer i assemblerfilerna aldrig initieras (de läses som 0xFF). Ett sätt att ta sig förbi detta problem är att i main.c deklarera en dummy-variabel som skall initieras, på köpet får man då att initieringen av variabler/tabeller/vektorer i assemblerfiler också görs på rätt sätt. Gör så här för att det skall fungera:

```
int dummy=0x1234; //LÄGG TILL DENNA RAD I main.c

int main()
{
```

Detta gör att följande fungerar korrekt i Subrutiner.S

```
.data
Tabell: .byte 1,2,3,4,5,6
```

- a) Om du använder Atmel Studio 7, behöver du ej göra ovanstående.

7. Laboration 2: Subrutiner och aritmetik

3. Något om assemblermakrona lo8 och hi8 En adress i dataminnet i ATmega16/Atmega32 är alltid på 16 bitar, dvs alla symboliska adresser/labels som deklarerats i assembler-koden representerar ett konstant numeriskt värde på 16 bitar som antingen tolkas som en adress i dataminnet eller programminnet. Makrona lo8 och hi8 har som funktion att ur ett 16 bitars värde plocka ut minst signifikanta 8 bitarna respektive de 8 mest signifikanta bitarna. Antag att vi har ett 16 bitars värde på hexadecimall form 0x1234:

```
lo8(0x1234) <-> 0x34  
hi8(0x1234) <-> 0x12
```

- a) Makrona används ofta till att ladda adresspekarregistrena Z (R31:R30), Y (R29:R28) och X (R27:R26) med adressen till starten av en tabell/vektor i dataminnet:

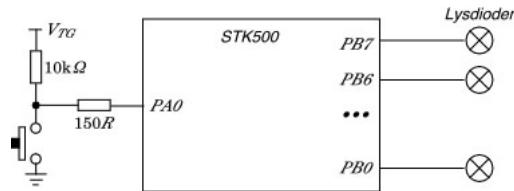
```
LDI R30,lo8(Tabell)  
LDI R31,hi8(Tabell)
```

Genom att addera till ett offset till t.ex Z-registret kan man adressera sig fram till starten för ett element i tabellen/vektorn.

7.3. Förändringsräknare

Problemformulering

Skriv ett program som klarar av att beräkna förändringarna på en strömställare. Som förändring räknar vi när strömställaren SW0 går från 0 till 1 respektive 1 till 0, vi räknar alltså positiva och negativa flanker. Vi räknar förändringarna i en bytevariabel och lägger ut dess värde på PORTB.



Figur 7.3.1.: Koppling av tryckknapp och lysdioder till portar

Tips för att lösa problemet

1. Studera följande pseudokodslösning i C, försök sedan att implementera denna i assembler.

```
char oldValue,newValue,counter;

void init_func()
{
    DDRB=0xFF;
    DDRA=0x00;

    newValue=PINA & 0x01;
    oldValue=newValue;

    counter = 0;
}

void loop_func()
{
    oldValue=newValue;
    newValue=PINA & 0x01;

    if ( oldValue != newValue )
        counter++;

    PORTB = counter;
}
```

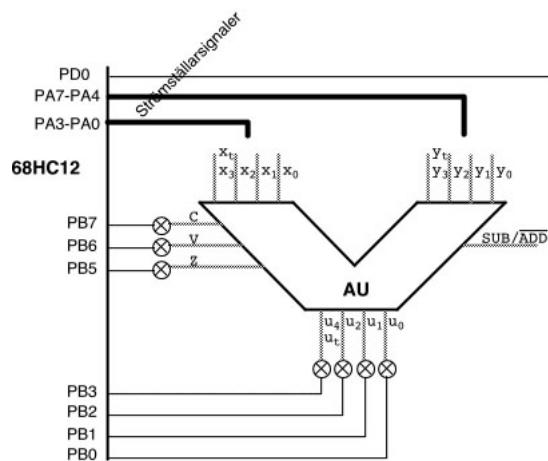
2. Utnyttja assemblerinstruktionen EOR för Exklusivt Eller i din lösning

7.4. Simulerad aritmetisk enhet (Arithmetical Unit - AU)

Följande uppgift är tänkt att köras i simulatorn på grund av att vi ej har tillräckligt med lysdioder och strömställare tillgängligt. Uppgiften är formulerad så att den även går att ladda ner och provköra på STK500-kortet om man gör uppkopplingen (se figuren nedan).

Problemformulering

Skriv ett program för en fyra bitars aritmetisk enhet. Följande koppling är tänkt att gälla för den simulerade aritmetiska enheten (AU).



Figur 7.4.1.: Aritmetisk enhet

Testvärden för att kontrollera funktionen hos den simulerade aritmetiska enheten

Här är några förslag på testvärden för att kontrollera funktionen hos den aritmetiska enheten, hitta gärna på egna testfall.

1. X=0001, Y=0001, SUB/ADD-N=0 (addition)
2. X=0001, Y=0001, SUB/ADD-N=1 (subtraktion)
3. X=1000, Y=1000, SUB/ADD-N=0 (addition)
4. m.m.

Beräkna vilket resultat du förväntar dig att få, d.v.s. U=??, Z=?, V=? och C=?.

7. Laboration 2: Subrutiner och aritmetik

7.5. Några loopar

- a) Skriv ett program som räknar upp en variabel varX i steg om 1. När variabeln nått värdet 7 skall räknaren börja om. Använd lysdioderna för att presentera räknarens värde och en lämplig tid på vänteloopsfördröjningen.
- b) Skriv ett program som räknar upp en variabel varX i enligt uppgift1, när detta har gjorts 3 ggr skall en räknare varY startas som räknar ned från 30 till 0 i steg om 2. Då denna räknare är klar börjar vi om från början. Använd 3 lysdioder för att presentera varX och 5 lysdioder för att presentera varY.

8. Intel _x86 assembler

Innehåll

8.1 Mål	59
8.2 Läsanvisningar	59

8.1. Mål

Några mål i denna modul:

- Att få en introduktion till datorarkitekturen och maskinspråket för en Intel x86-processor

8.2. Läsanvisningar

- Böcker
 - Assembly Language Succinctly
- <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>
- Slides
 - Intel x86 assembler
- Wikipedia: Intel 8086 föregångaren till dagens Intel-processorer
- Wikipedia: Micro Architecture

9. C# och CIL (Common Intermediate Language)

Att läsa

- Wikipedia: Common Intermediate Language
- Understanding Common Intermediate Language
- Verktyg
 - <http://ilspy.net/>
 - ◊ ILSpy is the open-source .NET assembly browser and decompiler.
 - ◊ Finns som en plug-in i Visual Studio också.

Part III.

Hårdvarunära programmering i C

Innehåll

10 Hårdvarunära programmering i C	64
10.1 Mål	64
10.2 Läsanvisningar	64
10.3 Videos	65
11 Laboration 3: Hårdvarunära programmering i C	66
11.1 C-program för att tända och släcka en lysdiod med en strömbrytare	69
11.2 Ringräknare i C	70
11.3 Spegling av en switchs värde på en LCD-display	71
11.4 Elektronisk tärning med presentation på en LCD-display	73
11.5 Bitfältstruktur för att visa två tärningars värden på lysdioder	73
11.6 Mätning av exekveringstider med simulatorn	74
11.7 Strukturen sedd på maskinnivån som värdeparameter respektive retur-typ	76
11.8 Falsk elektronisk tärning	76
11.9 Kodlås	76
11.10 Avstudsning av en tryckknapp	77
11.11 Trafikljusstyrning	77
12 Klassen time	80
12.1 Att läsa	80
12.2 Tidmätning	80
12.3 Mätning av exekveringstid	81
12.4 Styrning av en uppgifts periodiska exekvering	82

INNEHÅLL

13 Tillståndsmodellering	84
13.1 Att läsa	84
13.2 Beteende hos objekt	85
13.2.1 Enkelt beteende	85
13.2.2 Tillståndsbeteende	85
13.2.3 Kontinuerligt beteende	86
13.3 Syntax för tillståndsmaskiner i UML	87
13.3.1 Vilka variabler skall fungera som tillståndsvariabler?	89
13.3.2 Näslade och parallella tillstånd	90
13.4 Fördelar med tillståndsmodellering	92
14 FSM-implementering	94
14.1 Praktikfall: Trafikskolebilen	95
14.2 Lösning 1: Användning av en switch-sats	96
14.2.1 Huvudprogrammet "main.c"	96
14.2.2 Genomgång av inkluderingsfilen "fsm_driver.h"	97
14.2.3 Källkodsfilen "fsm_driver.c"	98

10. Hårdvarunära programmering i C

Innehåll

10.1 Mål	64
10.2 Läsanvisningar	64
10.3 Videos	65

10.1. Mål

Några mål i denna modul:

- Att få kännedom om de grundläggande ideerna i den objektorienterade programmeringsparadigmen och hur detta görs i C.
- Att inse att objektorientering till stor del är ett synsätt hos programmeraren.
- Att få en färdighet i olika sätt att programmera mot processorns I/O-register i språket C.

10.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- [Objektorienterad modellering](#)
 - Kapitel 1: Vad menas med inbyggda system?
 - ◊ Genomgång av olika begrepp relaterade till realtidsdatorsystem.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 2: Objektorienterad modellering
 - ◊ Grundläggande ideer i den objektorienterade programmeringsparadigmen tas upp.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 3: Gruppering av samhörande data i en struktur
 - ◊ Strukturen är grunden för att åstadkomma modellnärlhet och ett objektorienterat programmeringssätt i C.

10. Hårdvarunära programmering i C

- ◊ Gå igenom alla uppgifter.
- Kapitel 4: C och hårdvarunära programmering
 - ◊ Programmering mot en mikrostyrenhets I/O-register behandlas.
 - ◊ Gå igenom alla uppgifter.

10.3. Videos

- Microcontroller Tutorial - A Beginners Guide
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

11. Laboration 3: Hårdvarunära programmering i C

Innehåll

11.1	C-program för att tända och släcka en lysdiod med en strömbrytare	69
11.2	Ringräknare i C	70
11.3	Spegling av en switchs värde på en LCD-display	71
11.4	Elektronisk tärning med presentation på en LCD-display	73
11.5	Bitfältstruktur för att visa två tärningars värden på lysdioder	73
11.6	Mätning av exekveringstider med simulatorn	74
11.7	Strukturen sedd på maskinnivån som värdeparameter respektive returnertyp	76
11.8	Falsk elektronisk tärning	76
11.9	Kodlås	76
11.10	Avstudsning av en tryckknapp	77
11.11	Trafikljusstyrning	77

Att kontrollera : Om du använder en portpinne som Digital Utgång (DO) ska motsvarande bit i datariktningregistret (DDR) vara satt till 1

Mål

- Att kunna programmera mot hårdvarunära register i en processor och pereferikretsar anslutna till någon port.
- Förståelse för hur fysiska objekt i omvärlden såsom switchar, lampor, LCD-displayer, sensorer, aktuatorer, etc. kan modelleras som mjukvaruobjekt i datorsystemet.

Läsanvisningar, tips och uppgifter att göra

- Kompendiet: [Inbyggda system med AVR RISC-processorer](#)
 - Kapitel 8: C OCH ASSEMBLER
- [UML](#)

11. Laboration 3: Hårdvarunära programmering i C

- Kompendiet: [Objektorienterad modellering](#)
 - Kapitel 1: Vad menas med inbyggda system?
 - Kapitel 2: Objektorienterad modellering
 - Kapitel 3: Gruppering av samhörande data i en struktur
 - Kapitel 4: C och hårdvarunära programmering
- Tips rörande kodning av C-program
 - I de fall du håller på med stränghantering och vill använda sprintf-funktionen för att skapa strängar kan du ha nytta av följande tutorial som behandlar format strängen i printf-funktionen: [print.pdf](#). Prototypen för printf ser ut på följande sätt:

```
int printf(const char *format, arg1, arg2, arg3, ...);
```

Funktionen används ofta för formaterad utskrift till till ett konsolfönster (terminalfönster). Funktionen printf kommer vi att använda då vi kommunicerar mellan en PC-dator och vår AVR-processor, där vi då kan koppla printf till vår processors USART (seriell kommunikationsenhet). För formaterad utskrift till vår LCD-display kommer vi att använda oss av sprintf:

```
int sprintf(char *target_string, const char *format, arg1,
           arg2, arg3, ...);
```

Utdatat från sprintf, den formaterade strängen skickas ut via pekarargumentet (target_string) till en teckenarray.

- GNU C vs GNU C++ på objektkodsnivå (kompilerad kod)
 - [Wikipedia: Name mangling](#)
 - [Name mangling and extern C](#)

Inkluderingsfilen avr/io.h och io_bit_fields.h

Inkluderingsfilen "avr/io.h" innehåller i sin tur inkludering av den rätta inkluderingsfilen. Den rätta inkluderingsfilen beror av vald processor, i vårt fall är processorn en ATmega16/32 och detta gör att vi inkluderar "iom16.h" eller "iom32.h" i slutändan via "avr/io.h". Detta gör att vi på ett enkelt sätt kan flytta ett program till någon annan variant av AVR-processorn.

Då det är intressant att kunna arbeta med bifältsbeskrivningar av register så har för denna kurs tagits fram en inkluderingsfile som heter "io_bit_fields.h". En bitfältsbeskrivning av ett generellt 8 bitars register är bitfältstrukturen byteRegister

```
typedef struct
{
    unsigned char D0 : 1;
```

11. Laboration 3: Hårdvarunära programmering i C

```
unsigned char D1 : 1;
unsigned char D2 : 1;
unsigned char D3 : 1;
unsigned char D4 : 1;
unsigned char D5 : 1;
unsigned char D6 : 1;
unsigned char D7 : 1;
} byteRegister;
```

Med hjälp av denna bitfältsbeskrivning kan sedan portarna A-D betraktas som bitfältsvariabler. Detta ser till exempel ut på följande sätt för de tre register som är associerade med PORTB:

```
#define bPINB (*(volatile byteRegister *) &PINB)
#define bDDRB (*(volatile byteRegister *) &DDRB)
#define bPORTB (*(volatile byteRegister *) &PORTB)
```

Antag att en lysdiod är ansluten till pinne PB3 på processorn, den kan då tändas eller släckas med en enkel programsats enligt följande:

```
bPORTB.D3 = 1;
```

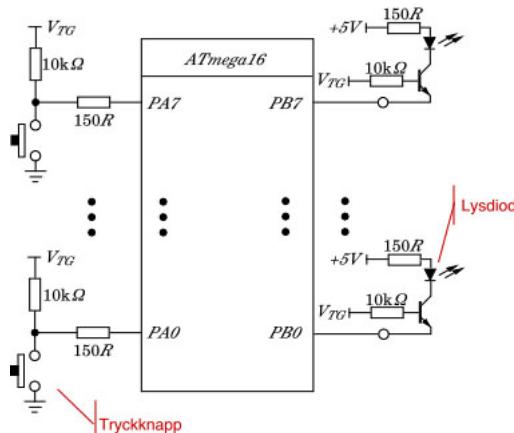
11.1. C-program för att tända och släcka en lysdiod med en strömbrytare

Uppgiften kan ses som en introduktionsuppgift för att komma igång med skriva C-program för processorn. Vi kommer även att använda simlatorn.

Problemformulering

Följande hårdvaruuppkoppling gäller: Åtta tryckknappar är anslutna till PORTA och åtta lysdioder till PORTB. Uppgiften är att låta var och en av tryckknapparna styra om korresponderande lysdiod skall vara tänd eller släckt. På STK500-kortet finns 8 tryckknappar märkta SW7,..,SW0 och 8 lysdioder märkta LED7,..,LED0. Tryckknapparna kopplas ihop med PORTA via en flatkabel, på liknande sätt kopplas lysdiodeerna ihop med PORTB.

Problemet i denna uppgift består i att skriva ett program som fångar upp en 1 till 0 övergång på switchen SW i ($i=0..7$) och togglar korresponderande lysdiod LED i ($i=0..7$).



Figur 11.1.1.: Uppkoppling ATmega16/32

Tips för att lösa problemet

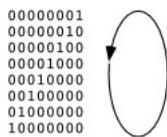
- Formulera en boolsk ekvation för problemet.
- Databehandlingen kan ske på alla 8 switchar med bitlogik-instruktioner.

11.2. Ringräknare i C

Denna uppgift skall lösas med hjälp av ett C-program.

Problemformulering

Skriv ett program som implementerar en 8 bitars ringräknare. Ringräknarens värde skall visas på 8 lysdioder anslutna till PORTB. Ringräknaren skall kunna startas och stoppas med SW3 anslutet till PA3. Du skall ej behöva stå och trycka på knappen för att den skall gå eller vara stoppad.



Figur 11.2.1.: Ringräknarmönstret

Några frågor

Svara på följande frågor som kan vara till en hjälp vid lösningen av detta och andra problem.

Med vilket heltalsvärde representeras det logiska värdet sant (true) i C?

- A) 1
- B) 0
- C) 0xFF
- D) -1

Följande C-kod gäller:

```
unsigned char sw6;  
//...  
sw6 = (PIN_A & 0x40) != 0;
```

Vilka värden kan sw6 anta?

- A) 0x00 eller 0x40 (0 eller 64)
- B) 0 eller 1 (0x00 eller 0x01)
- C) -128 till +127 (0x80 till 0x7F)
- D) 0 till 255 (0x00 till 0xFF)

11.3. Spegling av en switchs värde på en LCD-display

Problem 1

Skriv ett program som läser av switchen SW6 (PA6) och skriver ut dess värde på en LCD-display.

I denna uppgift har vi behov av att använda en LCD-display, beskrivningen av denna finns i först i kapitel 8 i kompendiet: "Inbyggda system och OBJEKTORIENTERAD MODELLERING" Vi ger därför ett programskeletten som hjälper till att lösa denna uppgift:

```
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include "lcd4.h"

//====Globala objekt====
lcd4 theDisplay;

//====main====
int main()
{
    char s[20];
    int value;

    DDRA = 0x00; //Hela PORTA är ingångar
    lcd4_init(&theDisplay, &PORTB, &DDRB, 4000, 50); //LCD-displayen
    //ansluten till PORTB

    while ( 1 )
    {
        value = PINA;

        sprintf( s, "PINA=%02X", value);
        lcd4_cup_row1(&theDisplay);
        lcd4_write_string(&theDisplay, s );

        _delay_ms(300);
    }
    return 0;
}
```

Kontrollera att fördjningsparametrarna vid funktionsanropet av lcd4_init har korrekta värden:

```
lcd4_init(&theDisplay, &PORTB, &DDRB, 4000, 100);
```

Tips för att lösa problemet

För att fixa till en sträng för utskrift på LCD-displayen kan du använda sprintf-funktionen:

```
char s[20];
int value_sw6;
//...
```

11. Laboration 3: Hårdvarunära programmering i C

```
value_sw6 = ((PINA & 0x40) != 0);  
sprintf( s , "SW6=%1d ", value_sw6);
```

I exemplet ovan skapas med hjälp av sprintf-funktionen en sträng s.

Problem 2

Skriv ett program som läser av de 8 switcharna SW7 till SW0 och skriver ut deras värden hexadecimalt på rad 1 och binärt på rad 2 i LCD-displayen.

11.4. Elektronisk tärning med presentation på en LCD-display

Problemformulering

Skriv ett program för en elektronisk tärning som presenterar sitt värde på en LCD-display.

Tips för att lösa problemet

I inkluderingsfilen "stdlib.h" finns en del funktionsprototyper för slumptalsgenerering:

```
long random (void) ;
void srand (unsigned long __seed) ;
long random_r (unsigned long *ctx) ;
```

11.5. Bitfältstruktur för att visa två tärningars värden på lysdioder

Problemformulering

Gör ett program som med hjälp av en tryckknapp för kast av tärningarna slumar fram 2 tärningsvärden. Ett tärningsvärdet visas på 3 lysdioder (binärform). För tärning 1 används bitarna D7-D5 och för tärning 2 används bitarna D2-D0 i PORTB. För att på ett smidigt sätt kunna sätta värdena på lysdiodeerna skall en bitfältstruktur användas:

```
typedef struct
{
    unsigned char dice2 : 3; //Bit 0-2
    unsigned char unused1 : 1; //Bit3 används ej
    unsigned char dice1 : 3; //Bit 4-6
    unisigned char unused2 : 1; //Bit 7 används ej.
} diceRegister;
```

Använd sedan makrokonstanter för att definiera någon port till att vara av denna typ. Lämpliga namn på makrokonstanterna om PORTB används är bDicePORT, bDicePIN och bDiceDDR:

```
#define bDicePORT (*(volatile diceRegister *) &PORTB)
#define bDiceDDR (*(volatile diceRegister *) &DDRB)
#define bDicePIN (*(volatile diceRegister *) &PINB)
```

Följande är ett exempel på användning av bitfältssstrukturen:

```
bDicePORT.dice2 = 6;
```

Alla definitioner lägger du lämpligvis i en inkluderingsfil med namnet "dice.h".

11.6. Mätning av exekveringstider med simulatorn

I ett realtidssystem är tiden av högsta vikt. Hinner vi att utföra beräkningar på den tid som vi vill i vår applikation? För att kunna avgöra detta är det till stor hjälp att kunna mäta upp tiden för exekvering av en funktion eller en beräkning. Tidsmätning i ett datorsystem kan göras på olika sätt, vi skall använda oss av simulatorn för att kunna mäta tiden uttryckt i maskincykler och i antalet mikrosekunder.

Som hjälp till uppgifterna som kommer har du följande program:

```
#include <stdio.h>

volatile int gXi;
volatile int gYi;
volatile int gRi;

int main(void)
{
    gXi = 20;
    gYi = 30;

    gRi = gXi * gYi;

    return 0;
}
```

- a)** Beräkna den tid det tar för AVR-processorn att utföra multiplikation av två stycken heltal:

```
int = int * int;
```

- b)** Vad händer om vi tar bort volatile framför de globala variablerna, studera assembler-koden före respektive efter borttagandet av volatile.
c) Använd simulatorn för att beräkna följande exekveringstider då processorn klockas med 8MHz:

```
int = int + int;
long = long + long;
float= float + float;
int = int * int;
long = long * long;
float= float * float;
int = int / int;
long = long / long;
float= float / float;
```

Förslagsvis gör du detta genom att skriva ett litet program som använder globala variabler:

11. Laboration 3: Hårdvarunära programmering i C

```
volatile int gXi;
volatile int gYi;
volatile int gRi;
volatile long gXl;
volatile long gYl;
volatile long gRl;
volatile float gXf;
volatile float gYf;
volatile float gRf;
void main()
{
    gXi=0x0040;
    gYi=0x0020;
    gRi=0;
    gXl=0x0040L;
    gYl=0x0020L;
    gRl=0;
    gXf=1.5;
    gYf=3.5;
    gRf=0;
    gRi = gXi + gYi;
    gRl = gXl + gYl;
    gRf = gXf + gYf;
//...
```

Använd nu simulatorn och sätt brytpunkter i programmet där de olika beräkningarna utförs. Nollställ tidsräknaren i simulatorn för varje enskild mätning.

- d) Vi skall nu titta lite på beräkningstider för uttryck som både innehåller flyttal och heltal:

```
float= float * int;
float= float * long;
float= float * float;
```

Mät upp exekveringstiderna.

- e) Kommentera resultaten i uppgift c) respektive d). Förklara resultatet i uppgift d)

11.7. Strukturen sedd på maskinnivån som värdeparameter respektive return-typ

Gås igenom på ett lektionstillfälle

- a) Hur förs en "struct" över som en värdeparameter i ett funktionsanrop sett från maskinnivå?
- b) Hur returneras en struct av en funktion på maskinnivå?

Utgå ifrån följande kodfragment och komplettera detta med egna idéer:

```
typedef struct
{
    long long x,y,z,u,v; //Hur många bytes?
} data;

volatile data sData ={0x1011121314151617, 0x2021222324252627, ... };

__attribute__((noinline))
data dataF3( volatile data d)
{
    d.x=0;
    return d;
}
```

11.8. Falsk elektronisk tärning

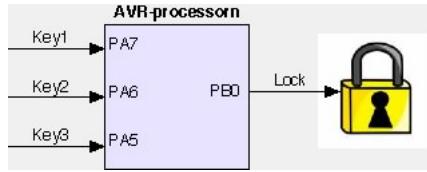
Gås igenom på ett lektionstillfälle

En tärning som skall kunna användas i falskspelssammanhang skall konstrueras. Förutom en tryckknapp – buttonThrow för att kasta tärningarna skall du ha en tryckknapp för falskspel – buttonFalse. Denna tryckknapp skall se till att de två nästföljande kasten ger tärningsvärdet 6. Falskspel får endast initieras då knappen nedtryckes (flankdetektering 1 till 0 eller 0 till 1 övergången skall detekteras).

11.9. Kodlås

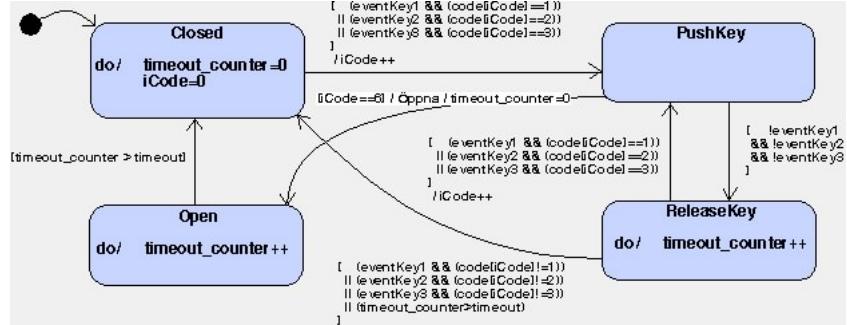
Ett kodlås ska konstrueras och implementeras. Kodlåset har tre trycknappar som skall nedtryckas i ordningen “123112” för att låset skall öppnas. Låset skall vara öppet i 20 sekunder då rätt kod har matats in innan låset stängs. Som lås använder du en lysdiod.

11. Laboration 3: Hårdvarunära programmering i C



Figur 11.9.1.: Uppkoppling för test av kodlåset

Ett förslag på tillståndsmaskin som löser problemet ges av följande tillståndsdiagram:



Figur 11.9.2.: Tillståndsdiagram för en möjlig lösning på problemet

Händelsen eventKey är typiskt att man detekterar antingen positiv eller negativ flank på en knapptryckning. På STK500-kortet är trycknappens normala värde en logisk 1:a om den ej är nedtryckt och en logisk 0:a om den är nedtryckt, detta leder till att en negativ flank bör detekteras på en sån tryckknapp för att signalera händelsen eventKey.

11.10. Avstudsning av en tryckknapp

Skriv ett program som läser av en tryckknapp på STK500-kortet och försöker detektera 1 till 0 övergången. Efter en sådan detektering skall programmet vänta med detektering av en sådan övergång ett visst antal läsningar (n stycken) för att slippa problemet med kontaktstudsar. Varje gång som 1 till 0 övergången detekteras skall värdet på en lysdiod toggas. Lös problemet med hjälp av en tillståndsmaskin.

Redovisa förutom programmet även ett tillståndsmaskin.

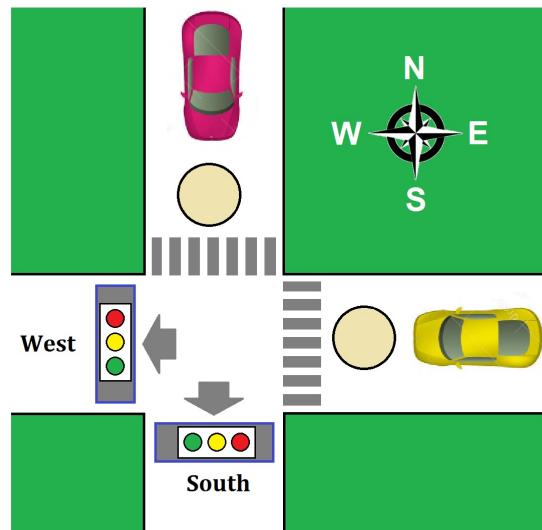
11.11. Trafikljusstyrning

Konstruera en trafikljustystyrning för två vägar som korsar varann, vägarna är enkelriktade.

Målet är att maximera trafikflödet, minimera väntetiden vid rött ljus och undvika trafikolyckor.

Trafikljustystyrningen använder sig av två sensorer och sex lampor. Sensorerna känner av trafik i de båda riktningarna. För varje trafikljus finns tre lampor: Rött, Gult och Grönt.

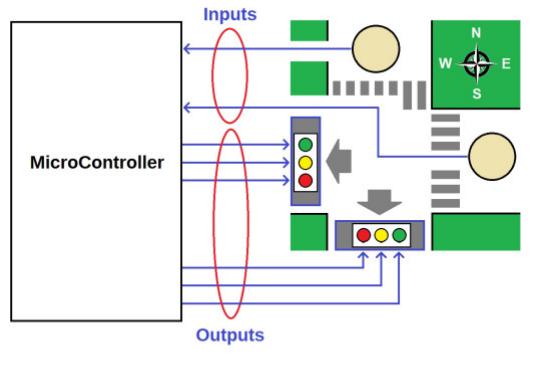
11. Laboration 3: Hårdvarunära programmering i C



Traffic Lights Intersection

Arduining.com

Figur 11.11.1.: Trafikljus



System Diagram

Arduining.com

Figur 11.11.2.: Trafikljus

Följande gäller för hur trafikljusen skall fungera:

- Om inga bilar kommer skall ett trafikljus lysa GRÖNT, vilket av de två spelar ingen roll, det andra lyser rött.
- Grönt ljus skall vara minst 3 sekunder.

11. Laboration 3: Hårdvarunära programmering i C

- Om bilar kommer endast i en riktning, skall motsvarande trafikljus gå till GRÖNT om det ej redan är det.
- Om bilar kommer i båda riktningarna skall tiden delas lika.

Följande fyra tillstånd är tänkbara i en lösning:

- StateGoSouth
 - Bilar körs i söderriktnings, GRÖN signal. Västerriktningen signalerar RÖTT.
- StateWaitSouth
 - Bilar väntar i söderriktnings, GUL signal. Vänsterriktning signalerar RÖTT.
- StateGoWest
 - Bilar körs i vänsterriktnings, GRÖN signal. Söderriktningen signalerar RÖTT.
- StateWaitWest
 - Bilar väntar i vänsterriktnings, GUL signal. Söderriktningen signalerar RÖTT.

Gör en objektorienterad lösning med tillståndsdiagram (UML). För sensorsignalerna används tryckknappar och lamporna med lysdioder.

12. Klassen time

Innehåll

12.1 Att läsa	80
12.2 Tidmätning	80
12.3 Mätning av exekveringstid	81
12.4 Styrning av en uppgifts periodiska exekvering	82

12.1. Att läsa

- Slides
 - [Klassen time - slides](#)
-

12.2. Tidmätning

I ett realtidsystem finns bland annat följande två skäl till varför tidsmätning är viktigt:

1. Styrning av uppgifter som skall utföras periodiskt.
2. Mätning av exekveringstid.
 - a) Viktigt att ha en uppfattning om vad som tar tid att exekvera i ett realtids-system. Tidmätningen kan ligga till grund för en WCET-analys, Worst Case Execution Time.

I ett inbyggnadsystem som också är ett realtidssystem har man nytta av en klocka för att kunna utföra olika uppgifter (tasks) periodiskt. Vanliga tidsstyrda uppgifter är:

- MMI - Människa Maskin Interface
 - Uppdatering av en LCD-display, typiskt 3-4 gånger per sekund
 - Avläsning av tryckknappar, typiskt 3-4 gånger per sekund
 - Avläsning av tryckknappar med öka/minska-facilitet som accelererar ju längre tid knappen hålls intryckt, periodisk exekvering med 100 ms tidsmellanrum

12. Klassen time

- Styrning/reglering
 - Temperaturreglering
- Seriell kommunikation med andra datorer
- .etc

12.3. Mätning av exekveringstid

I exemplet används klassen time för att mäta exekveringstiden för utskrift till en LCD-display.

```
time theTime(time::MilliSecondsTwo, time::TIMER2);
unsigned long long t, t1=0,t2=0;
ISR(TIMER2_COMP_vect)
{
    theTime.update();
}

void test_lcd()
{

    //Trimma fördröjningsparametrarna, ha sedan en 50% marginal för den mindre
    //och 25% för den störrre
    lcd4 theLCD(&PORTB, 30, 10);
    unsigned int x = 0;
    char s[20];

    asm(" SEI "); //time-klassen kräver aktivering av avbrottsfunktion
    theTime.get(&t2);

    while(1)
    {
        //Loop time measurement
        t1 = t2;
        theTime.get(&t2);

        x++;
        theLCD.cup_row1();
        sprintf(s, "LCD-TEST %7u", x++);
        theLCD.write_string(s);

        sprintf(s,"%12lu ms", time::difference_ms(&t1, &t2));

        theLCD.cup_row2();
        theLCD.write_string(s);
    }
}
```

Ett klocktick för klockan är 2 millisekunder. Mättider kortare än detta är ej lönt att mäta. Ett sätt att mäta på kortare tider är att bilda ett medelvärde över ett antal exekveringar av koden.

12. Klassen time

```
const int N=100;
int n=0;
while(1)
{
    n = (n+1) % N;
    if ( n == 0 ) // N varv mäter vi
    {
        t1 = t2;
        theTime.get (&t2);
    }

    x++;
    theLCD.cup_row1();
    sprintf(s, "LCD-TEST %7u", x++);
    theLCD.write_string(s);

    if ( n == 0 )
    {
        sprintf(s,"%12lu ms", time::difference_ms(&t1, &t2)/N);
        theLCD.cup_row2();
        theLCD.write_string(s);
    }
}
```

12.4. Styrning av en uppgifts periodiska exekvering

I exemplet är looptiden satt till 100 millisekunder, 1 varv i loopen tar alltså 100 millisekunder. Lysdioderna på port B, PB6 respektive PB7 blinkas med periodtiden 1 sekund respektive 2 sekunder.

```
void test_di_di()
{
    digital_input theDI0 (&PORTA,digital_input::D0);
    digital_input theDI1 (&PORTA,digital_input::D1);
    digital_input theDI2 (&PORTA,digital_input::D2);
    digital_input theDI3 (&PORTA,digital_input::D3);
    digital_input theDI4 (&PORTA,digital_input::D4);
    digital_input theDI5 (&PORTA,digital_input::D5);
    digital_input theDI6 (&PORTA,digital_input::D6);
    digital_input theDI7 (&PORTA,digital_input::D7);

    digital_output theDO0 (&PORTB,digital_output::D0);
    digital_output theDO1 (&PORTB,digital_output::D1);
    digital_output theDO2 (&PORTB,digital_output::D2);
    digital_output theDO3 (&PORTB,digital_output::D3);
    digital_output theDO4 (&PORTB,digital_output::D4);
    digital_output theDO5 (&PORTB,digital_output::D5);
    digital_output theDO6 (&PORTB,digital_output::D6);
    digital_output theDO7 (&PORTB,digital_output::D7);
```

12. Klassen time

```
//Trimma fördröjningsparametrarna, ha sedan en 50% marginal för den mindre
//och 25% för den störrre
lcd4 theLCD(&PORTD, 30, 10);
unsigned int n = 0;
char s[20];

asm(" SEI "); //time-klassen kräver aktivering av avbrottsfunktion
theTime.get(&t);
while(1)
{
    theDO0.write(theDI0.read());
    theDO1.write(theDI1.read());
    theDO2.write(theDI2.read());
    theDO3.write(theDI3.read());
    theDO4.write(!theDI0.read());
    theDO5.write(!theDI1.read());
    if ( n % 5 == 0)
        theDO6.write(!theDO6.read());
    if ( n % 10 == 0)
        theDO7.write(!theDO7.read());

    sprintf(s,"%12lu ms", time::difference_ms(&t1, &t2));
    theLCD.cup_row2();
    theLCD.write_string(s);

    n++;
    t = t + 100;
    theTime.wait_until(&t);
}
}
```

13. Tillståndsmodellering

Innehåll

13.1	Att läsa	84
13.2	Beteende hos objekt	85
13.2.1	Enkelt beteende	85
13.2.2	Tillståndsbeteende	85
13.2.3	Kontinuerligt beteende	86
13.3	Syntax för tillståndsmaskiner i UML	87
13.3.1	Vilka variabler skall fungera som tillståndsvariabler?	89
13.3.2	Nästlade och parallella tillstånd	90
13.4	Fördelar med tillståndsmodellering	92

Tillståndsmodellering och objektorienterad programmering tillsammans erbjuder oanade möjligheter för att konstruera programvara i form av objekt som är tillförlitliga, förutsägbara, testbara och hanterabara. Vi skall i detta avsnitt försöka att presentera och motivera till att använda tillståndsmaskiner vid objektkonstruktion. I UML används en tillstånds-diagram-metod som har skapats av David Harel i början på 1980-talet. Harel är en matematiker från Israel. På engelska går denna metod under namnet "Harel State Charts". Tillståndsdiagram enligt Harels metod supportar en mängd olika finesser som ej finns i tillståndsdiagram av typen Mealy eller Moore (se kursen i Digitalteknik).

13.1. Att läsa

- Slides
 - [Tillståndsmaskiner i UML](#)
- [Wikipedia: Finite State Machines](#)
- [www.embedded.com](#)
 - [A crash course in UML state machines: Part 1](#)
 - [A crash course in UML state machines: Part 2](#)
 - [A crash course in UML state machines: Part 3](#)

13.2. Beteende hos objekt

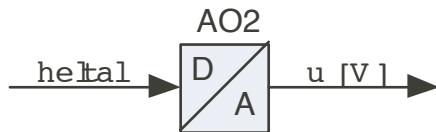
I boken "Doing Hard Time" av Bruce Powel Douglass grupperar han det dynamiska beteendet hos ett objekt i 3 olika grupper: Objekt med enkelt beteende, objekt med tillståndsbeteende och kontinuerligt beteende hos objekt. Ett objekt kan uppvisa en eller flera av dessa beteenden beroende på ur vilken synvinkel vi betraktar det. Objektens beteende är implementeringsmässigt sett metoder som manipulerar objektens interna dataareor. När vi gör modeller av verkligheten i form av objekt fokuserar vi mer på vilket beteende det verkliga objekten har och skall ha. Det jag vill ha sagt med detta är att vi skall ha en högre grad av abstraktion än den vi har då vi implementerar något i ett programspråk, dvs fokusera på helheten och ej på detaljerna.

13.2.1. Enkelt beteende

Ett objekt med ett enkelt beteende har ett beteende som ej beror av objektets historia. Ett enkelt objekt ger alltid samma svar på en given insignal oavsett dess tidigare historia. Ett enkelt beteende hos ett objekt kan till exempel finnas i allt som beskrivs med hjälp av matematiska funktioner såsom t. ex. $\sin(x)$ och $\log(x)$.

En digital till analog-omvandlare kan sägas ha ett enkelt beteende sett från en funktionell vy. Utsignalspänningen från D/A-omvandlaren beror direkt av den digitala insignalen, vi har ett funktionellt samband mellan ut- och insignal.

```
ao2.SetVoltage(3.5);
```



Figur 13.2.1.: En D/A-omvandlare har ett enkelt beteende

13.2.2. Tillståndsbeteende

Ett objekt med tillståndsbeteende har ett beteende som beror av dess historia samt att det har ett ändligt antal beteenden.

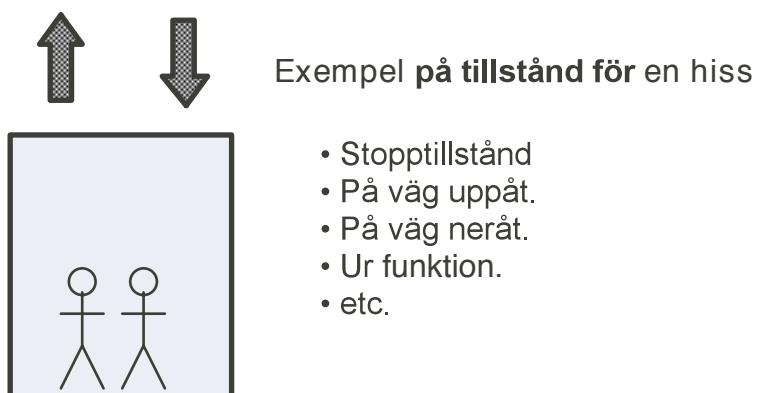
Ett objekts beteende kan i många fall realiseras med hjälp av en tillståndsmaskin. Ordet maskin har betydelsen av en apparat eller mekanisk anordning som utför ett visst arbete. I digitaltekniken har vi träffat på två varianter av tillståndsmaskiner eller sekvensnät som det brukar benämñas med, sekvensnät av typen Mealy respektive Moore. Allmänt kan vi säga att med hjälp av tillstånden kan vi minnas en sekvens av händelser som har inträffat. Med hjälp av tillståndsmaskiner inför vi underförstått ett tidberoende beteende hos ett objekt.

Tillståndsmodellering kan i många fall vara ett bra sätt att modellera ett objekt beteende. Observera att denna metod ej går att applicera på alla typer av objekt. Andra

13. Tillståndsmodellering

typer av objekt kan ha beteende som ej tillåter sig att modelleras med hjälp av tillståndsmaskiner.

En tillståndsmaskin för ett objekt beskriver dess beteende internt i form av ett ändligt antal tillstånd. Ett objekts beteende beror av tillståndet som objektet befinner sig i. Objektet kan bara befina sig i ett tillstånd åt gången. Objektets beteende i ett tillstånd är oberoende av andra tillstånd och dess beteende i tillståndet definieras av de händelser som den reagerar på och de åtgärder som utförs på grund av dessa. Objekt som konstruerats med hjälp av tillståndsmaskiner reagerar på händelser på ett väldefinierat och förutsägbart sätt (förhoppningsvis), denna typ av objekt brukar också kallas reaktiva objekt (reactive objects).



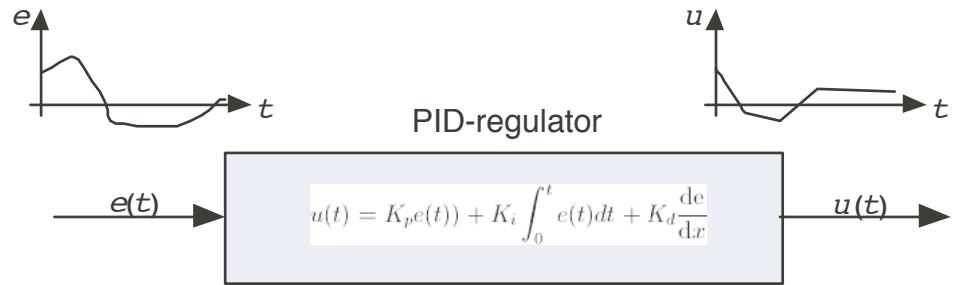
Figur 13.2.2.: En hiss uppvisar ett tillståndsbeteende

13.2.3. Kontinuerligt beteende

Ett objekt med kontinuerligt beteende har ett beteende som beror av dess historia samt att det har ett oändligt antal beteenden. Objektets beteende beror av dess historia på ett kontinuerligt sätt.

Ett kontinuerligt beteende hos ett objekt kan till exempel finnas i allt som beskrivs med hjälp av differentialekvationer eller differensekvationer. Ett exempel på detta är en PID-regulator som internt använder sig av flyttalsaritmetik, aktuell utsignal beror av regulators förhistoria och har ett kontinuerligt uppförande.

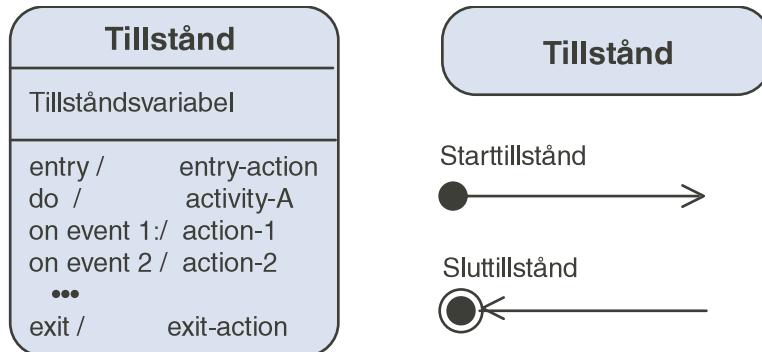
13. Tillståndsmodellering



Figur 13.2.3.: En regulator uppvisar ett kontinuerligt beteende

13.3. Syntax för tillståndsmaskiner i UML

Innan vi går in i detalj på modellering av objektbeteende med hjälp av tillståndsdiagram, skall vi presentera några grundläggande symboler i UML för detta. Tillståndsdiagrammet beskriver det beteendemönster som gäller för objekten i en klass. Detta i form av en mängd tillstånd som ett objekt kan anta och ett antal specifika övergångar mellan dem.



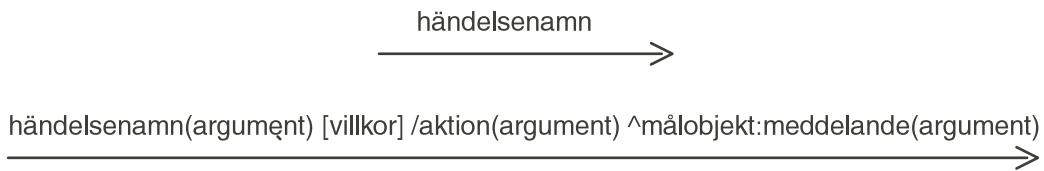
Figur 13.3.1.: Grafisk syntax för tillståndsdiagram

I punktform ger vi kommentarer till syntaxen i tillståndsdiagrammet:

- Det system som modelleras kan anta ett ändligt antal tillstånd. Där varje tillstånd karakteriseras av villkor för att vara i detta.
- Ett tillstånd kan klart urskiljas från övriga tillstånd. Ett tillstånd överlappar ej ett annat tillstånd, d.v.s. tillstånden är ortogonala.
- Ett system kan bara vara i ett tillstånd i en given tidpunkt.
- Systemets beteende i ett givet tillstånd definieras genom:
 - Meddelanden och händelser som accepteras.
 - Åtgärder som utförs då en händelse inträffar (**on event x /**).

13. Tillståndsmodellering

- De operationer (aktioner) som utförs då vi går in (**entry** /) i respektive lämnar (**exit** /) tillståndet.
- Den aktivitet som pågår i tillståndet (**do** /).
- Tidsmässigt så befinner sig ett system i ett visst tillstånd en signifikant tidsperiod. Ett objekt spenderar all sin tid i tillstånden.
- Tillståndsövergångar kan specificeras på olika sätt och med olika detaljeringsgrad.



Figur 13.3.2.: Grafisk syntax för tillståndsövergångar

I punktform kommenterar vi det viktigaste om tillståndsövergångar:

- Systemet kan byta tillstånd på ett ändligt antal väldefinierade sätt genom vad vi benämner som tillståndsövergång. En tillståndsövergång representerar ett svar på en händelse som inträffar.
- En tillståndsövergång representerar ett svar på en händelse. Händelser kan genereras externt i andra objekt eller internt i objektet självt.
- En tillståndsövergång sker ögonblickligen och tar noll i tid.
- En tillståndsövergång kan ha tillhörande aktioner – actions som skall utföras. Med aktioner menar vi handlingar som skall utföras.
- En aktion är en funktion som tar relativt lite tid att utföra. Om den skulle ta en signifikant tid att utföra bör ett tillstånd införas istället.
- En aktion kan implementeras som en operation i objektet eller externt som en vanlig funktion.
- En aktion kan inträffa när en tillståndsövergång göres, vid inträdet (**entry**) i ett tillstånd eller vid utgång (**exit**) ur ett tillstånd.

UML-syntaxen för en tillståndsövergång är:

```
händelsenamn (argument )
[villkor]
/aktion(argument)
^målObjekt.sändHändelse(argument)
```

Var och ett av dessa fält kan valfritt tas med:

13. Tillståndsmodellering

- Händelsenamn
 - Tillståndsövergångens namn, ofta är det endast detta som finns med i en tillståndsövergång. En argumentlista kan skickas med vid tillståndsövergången. Ett argument kan t.ex. vara en felkod, övervakat värde, etc.
- Villkor
 - Villkor som måste vara uppfyllt innan tillståndsövergången kan ske finns mellan haklammrar. I den engelska litteraturen användes ordet guard, vilket kan översättas till vakt eller skydd på svenska.
- SändHändelse
 - SändHändelse är en kommaseparerad lista av meddelanden som sänds till ett målobjekt. Händelser av denna typ kommer att få en verkan utanför objektet och det är på detta sätt som parallella tillståndsmaskiner kan kommunicera med varann. En tillståndsövergång i en tillståndsmaskin kan påverka tillståndet i en annan tillståndsmaskin.
- Aktion
 - Aktion (handling,) är en kommaseparerad lista av åtgärder som skall utföras vid tillståndsövergången. Varje åtgärd kan ha ett valfritt antal argument. En åtgärd är normalt ett anrop på någon funktion.

Om ett meddelande skall skickas i en tillståndsmaskin för ett objekt till en annat objekt åskådliggörs detta med en streckad pil och ett meddelandenamn på pilen. Meddelandenamnet är normalt ett anrop på en klassmetod hos det mottagande objektet.

meddelande(argument) →

Figur 13.3.3.: Grafisk syntax för att skicka meddelanden i en tillståndsgraf

13.3.1. Vilka variabler ska fungera som tillståndsvariabler?

Ett systems/objekts tillstånd kan definieras på olika sätt. En definition skulle vara att objektets tillstånd representeras av all den information som hålls i objektet.

13. Tillståndsmodellering

Synsätt 1: Värdet av alla attribut definierar tillståndet.

Både `_value` och `_state` definierar tillståndet.

Antalet tillstånd är oändligt, då `_value=0` och t. ex. `_value=0.5` representerar olika tillstånd.

```
sensor
enum states
{ OFF,
  CALIBRATING,
  MEASURING,
  VALID_MEASUREMENT,
  NO_VALID_MEASUREMENT
};
float _value;
states _state;

states GetValue(float& v);

***
```

Synsätt 2: Värdet av en delmängd av attributen definierar tillståndet.

Endast `_state` definierar tillståndet. Antalet tillstånd är ändligt och i vårt exempel finns 5 tillstånd: OFF, CALIBRATING, VALID_MEASUREMENT, NO_VALID_MEASUREMENT, MEASURING.

Figur 13.3.4.: Vad skall betraktas som tillståndsvariabler i sensorklassen?

Men i ansatsen med tillståndsmodellering kommer vi att vara tydligare i vad vi menar med ett tillstånd. Och utgå ifrån att vi har vissa attribut i objektet ett eller flera som fungerar som tillståndsvariabler och unikt definierar tillståndet.

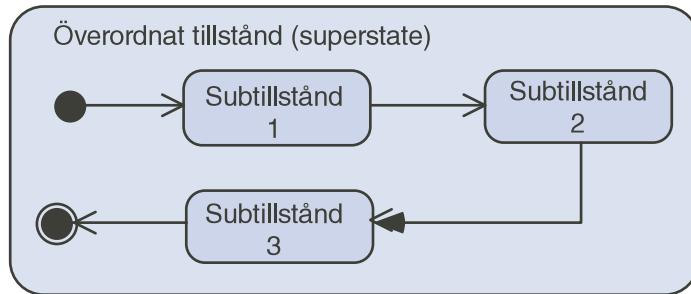
13.3.2. Nästlade och parallella tillstånd

Att kunna använda nästling av tillståndsmaskiner erbjuder en mängd olika fördelar:

- Ökar förståelsen av objektets beteende.
- Tillåter att problemet delas ner i delproblem (strategi divide and conquer).

13. Tillståndsmodellering

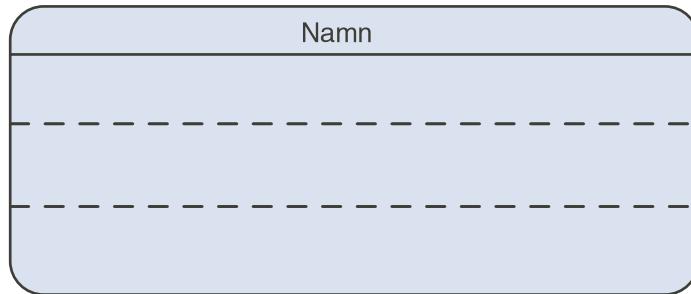
Nestling av tillstånd



Figur 13.3.5.: Syntax för nästlade tillstånd

Två eller flera tillståndsmaskiner kan återges i samma diagram och åtskiljs av streckade linjer. Varje tillståndsmaskin implementeras normalt som en egen klass. Detta kan vara speciellt värdefullt då flera parallella tillståndsmaskiner samverkar med varann på ett icke trivialt sätt. Det normala är dock att ha endast en tillståndsmaskin per diagram.

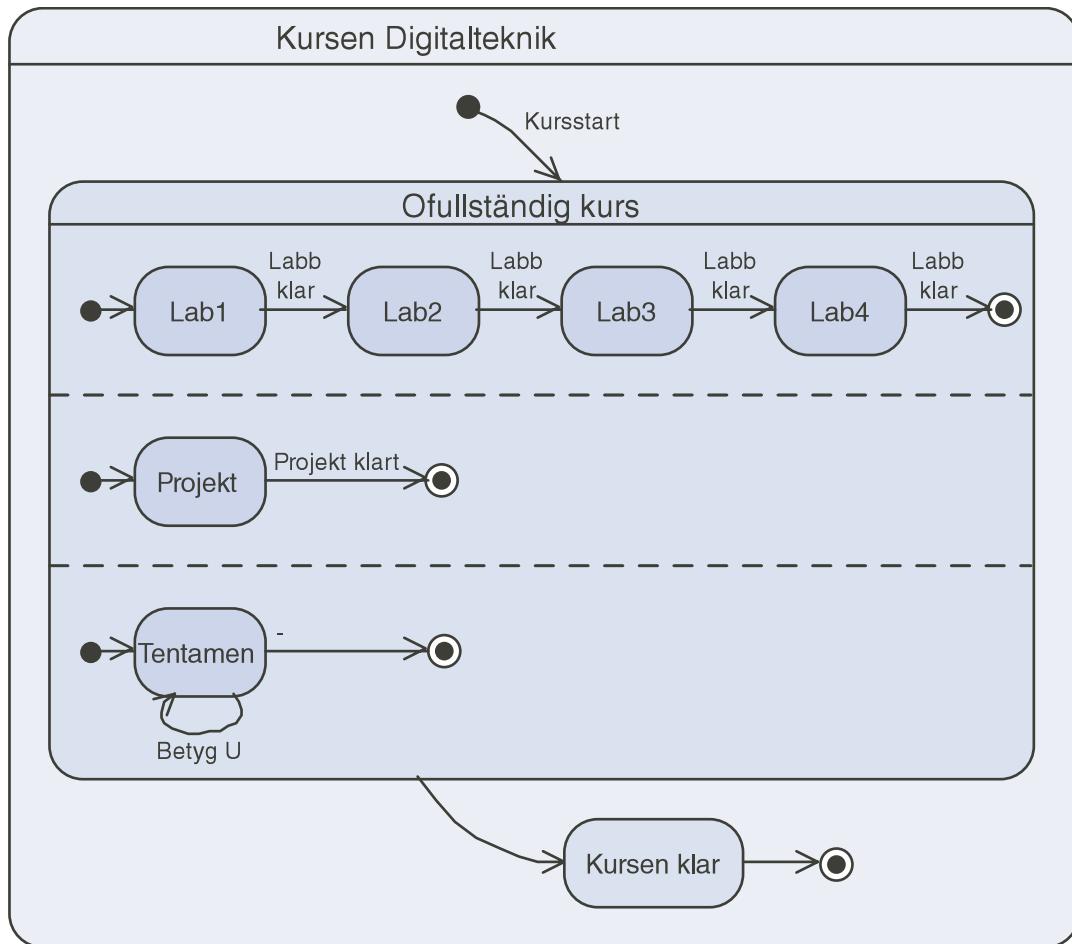
Tillstånd med parallella deltillstånd



Figur 13.3.6.: Syntax för parallella tillstånd

En typisk universitetskurs på den tekniska sidan kan betraktas som en tillståndsmaskin med både nästlade och parallella tillstånd.

13. Tillståndsmodellering



Figur 13.3.7.: Kursen digitalteknik betraktad ur ett tillståndsdiagram

13.4. Fördelar med tillståndsmodellering

Varför är tillståndsmodellering intressant? En del av svaret får vi genom att titta på hur de flesta datorsystem fungerar idag, de flesta är händelsedrivna system. Programmeringen av sådant system baserar sig då på att exekvera rätt kod som en reaktion på en händelse. Reaktionen på händelsen beror både på händelsens natur (egenskaper) och den aktuella kontexten för systemet är i. Kontexten för systemet beror av den sekvensen av tidigare händelser som systemet har varit involverad i. Den traditionella ansatsen med ”bottom-up-programmering” leder ofta till mängd av variabler och flaggor med en kod översällad av sammansatta if-else-satser och switch-satser. Lösningen på detta problem är att använda tillståndsmodellering och tillståndsmaskiner. En tillståndsmaskin ger respons på en händelse som uttryckligen beror av händelsens typ och sammanhanget/tillståndet (kontexten) systemet befinner sig i. Tillståndet karakteriseras av att det fångar de relevanta aspekterna av systemets historia på ett mycket bra sätt, samt att det kan bortse

13. Tillståndsmodellering

från alla irrelevanta händelser och bara fånga de relevanta.

Tillståndsmodellering förenklar problemlösningen genom att dela ner problemet i ett antal tillstånd samt genom att vi antar att systemet kan bara befina sig i ett tillstånd åt gången. Vi kan betrakta ett tillstånd som ett delproblem. Problemlösningfas uppmuntrar till att använda top-down-konstruktion av ett objekt, se först på helhetsproblemet och bryt ner detta till ett antal delproblem som representerar var sitt tillstånd.

I ett givet tillstånd har objektet en begränsad interaktion mot andra objekt. Ett ändligt antal tillståndsövergångar tillåts, andra händelser ignoreras, kan leda till inträde i ett fältillstånd eller eventuellt köas upp för senare användning.

Vi får också en förenklad felhantering då det är lätt att testa av giltiga och icke-giltiga indata i ett givet tillstånd.

Genom att använda tillståndsmodellering för ett system får vi något som är mer förståeligt och förutsägbart (predikterbart). Detta erhålls genom att varje tillstånd är enklare än helheten.

Utveckling och uttestning av programvaran förenklas genom att helheten delas ner i ett antal mindre bitar, d.v.s. tillstånd. Små programbitar är enklare att koda, enklare att testa ut, och enklare att förklara dess beteende. Varje tillstånd har ett mindre antal indata- och utdata-villkor än helheten, vilket underlättar uttestningen. Uttestningen av programvaran förenklas också genom att vi normalt har en låg koppling mellan tillstånd.

14. FSM-implementering

I detta kapitel shall en genomgång göras av några olika sätt att implementera en Finite State Machine på svenska översätter vi detta till tillståndsmaskin. Det absolut enklaste sättet är att använda en switch-sats för att implementera en tillståndsmaskin. Lösningsförslagen utgår också ifrån att lösningen skall implementeras på en 8 bitars AVR-processor.

Att använda tillståndsmaskiner i en lösning av ett programmeringsproblem innebär många fördelar: En fördel är att systemets beteende kan beskrivas med hjälp av formalisering UML-semantik, en annan fördel är att det uppmuntrar till att lösa programproblemet innan du börjar koda.

Två typer av konstruktionsmönster för tillståndsmaskiner är vanliga: den ena är den händelsebaserade (händelsedrivna) tillståndsmaskinen och den andra är den periodiska tillståndsmaskinen:

- *Händelsebaserade tillståndsmaskiner*

- Är tillståndsmaskiner i vilket alla interna tillståndsövergångar sker som svar på händelser. Händelserna kan typiskt i ett inbyggnadssystem genereras från I/O-enheter såsom räknare, seriella kommunikationsenheter, A/D- och D/A-omvandlare, sensorer, etc. Händelserna i dessa fall kan genereras med hjälp av avbrott eller pollning av I/O-enheten. Även mjukvarumoduler i programsystemet kan ge upphov till händelser.

- *Periodiska tillståndsmaskiner*

- Exekveras med reguljära intervall och gör alla tillståndsövergångar som svar på förändringar i indata/insignaler som pollas vid exekveringen. Den här typen av tillståndsmaskin kan fungera som källa för händelser till en händelsebaserad tillståndsmaskin.

Kodning av tillståndsmaskiner i C görs ofta som variationer på följande:

- Nästlade switch-satser

- Tillståndsbyte baseras på nuvarande tillstånd och senaste händelse från omgivningen.

- Blandning av switch- och if-satser

- Tillståndsvariabelns värde styr tvilken case-sats (tillstånd) som utväljs. If-satserna används för att testa om tillståndsbyte skall ske.

14. FSM-implementering

- Tabelldriven ansats

– Denna ansats kan utformas på en mängd olika sätt. Ett sätt är att koda övergångarna i tillståndsmaskinen med en tabell som kombineras med en drivrutin som behandlar händelsen och avgör nästa tillstånd.

14.1. Praktikfall: Trafikskolebilen

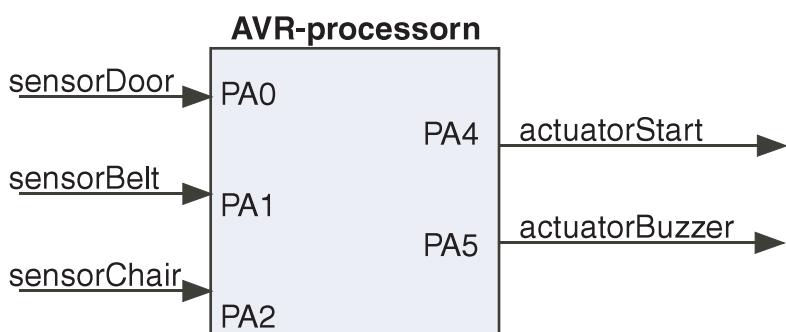
För att träna in den sekvens av moment som ingår i att starta en bil skall en körskolebil byggas om och ha följande startfunktion i ordningsföljd:

1. Öppna dörren och tag plats på förarsätet
2. Stäng dörren.
3. Sätt på säkerhetsbältet.

Om sekvensen ovan genomlöps i korrekt ordning ges signal till ett relä (actuatorStart) som tillåter att startnyckeln kan vridas om och starta bilen. Om moment 2 och 3 kastas om så skall en summerton (actuatorBuzzer) ljudha. Enda sättet att stoppa denna är att öppna dörren och kliva ur bilen.

Följande digitala signalgivare (sensorer) finns:

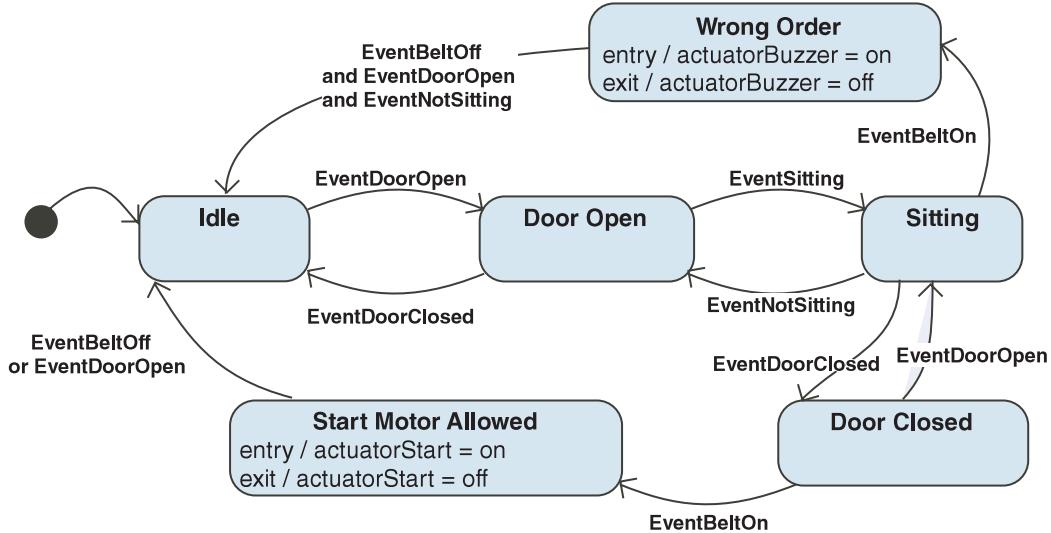
- Dörr öppen
 - sensorDoor, 1=öppen; 0=stängd
- Förarsätet upptaget
 - sensorChair, 1=upptaget; 0=ledigt
- Säkerhetsbältet påsatt
 - sensorBelt, 1=påsatt; 0=ej påsatt



Figur 14.1.1.: Sensorers och aktuatorers anslutning till AVR-processorn

14. FSM-implementering

Ett förslag på lösning av ovanstående problem är följande tillståndsgraf:



Figur 14.1.2.: Tillståndsdiagram

14.2. Lösning 1: Användning av en switch-sats

I detta avsnitt skall vi gå igenom hur hela lösningen kodas i C. Lösningen består av tre filer:

- main.c
- fsm_driver.h
- fsm_driver.c

14.2.1. Huvudprogrammet "main.c"

Huvud programmet blir väldigt kort och inleds med inkludering av ett antal hjälpfiler:

```

#include <stdio.h>
#include <avr/io.h>
#include "io_bit_fields.h"
#include "fsm_driver.h"
  
```

Ett globalt objekt med namnet putte av klassen fsm_driver finns deklarerat. Detta objekt är kapslar in hela hanteringen av tillståndsmaskinen:

```

fsm_driver putte;
  
```

I main-funktionen görs den sedvanliga initieringen av objektet putte och i while-loopen som aldrig termineras anropas metoden fsm_driver_run periodiskt:

14. FSM-implementering

```
int main()
{
    fsm_driver_init(&putte);

    while ( 1 )
    {
        fsm_driver_run(&putte);
    }

    return 0;
}
```

14.2.2. Genomgång av inkluderingsfilen "fsm_driver.h"

Inkluderingsfilen innehåller följande definitioner och deklarationer:

- bitfältstruktur (fsm_driver_register) som beskriver hur sensorer och aktuatorer är anslutna till processorn.
- en uppräknad datatyp (enum, fsm_driver_states) som namnger alla tillstånd för denna tillståndsmaskin
- en struktur som utgör grunden för klassen fsm_driver

Starten på inkluderingsfilen innehåller det sedvanliga skyddet för att slippa problemet med rekursiv inläsning av en och samma inkluderingsfil:

```
#ifndef _FSM_DRIVER
#define _FSM_DRIVER
#include <avr/io.h>
```

Alla sensor- och aktuator-signaler är digitala, i denna lösning kommer alla dessa att vara inkopplade till samma port (PORTA,PORTB,PORTC,PORTD) på AVR-processorn. För att kunna manipulera dessa på ett enkelt sätt definierar vi en bitfältstruktur:

```
typedef struct
{
    unsigned char sensorDoor : 1;      //D0
    unsigned char sensorBelt : 1;       //D1
    unsigned char sensorChair : 1;      //D2
    unsigned char unused1 : 1;          //D3
    unsigned char actuatorStart : 1;    //D4
    unsigned char actuatorBuzzer : 1;   //D5
    unsigned char unused2 : 2;          //D6-D7
} fsm_driver_register;
```

PORTA används för att koppla in sensorer och aktuatorer:

```
#define bPORT_DRIVER (*(volatile fsm_driver_register *) &PORTA)
#define bDDR_DRIVER  (*(volatile fsm_driver_register *) &DDRA)
#define bPIN_DRIVER  (*(volatile fsm_driver_register *) &PINA)
```

14. FSM-implementering

För att på ett enkelt sätt kunna hantera de olika tillstånden i tillståndsmaskinen använder vi symboliska konstanter i form av uppräknande heltalskonstanter genom en enum-sats:

```
typedef enum
{
    FMS_DRIVER_STATE_IDLE,
    FMS_DRIVER_STATE_DOOR_OPEN,
    FMS_DRIVER_STATE_SITTING,
    FMS_DRIVER_STATE_DOOR_CLOSED,
    FMS_DRIVER_STATE_START_MOTOR_ALLOWED,
    FMS_DRIVER_STATE_WRONG_ORDER
} fsm_driver_states;
```

Då vi arbetar io objektorienterad C skapar vi en struktur för att hålla allt data som rör klassen fsm_driver. I vårt fall så innehåller strukturen endast en medlemsvariabel:

```
typedef struct
{
    unsigned char state;
} fsm_driver;

void fsm_driver_init(fsm_driver *this);
void fsm_driver_run(fsm_driver *this);

#endif
```

14.2.3. Källkodsfilen "fsm_driver.c"

En tillståndsmaskin har en enkel uppbyggnad i form av en mängd händelser, en mängd av tillstånd samt en mängd av tillståndsovergångar.

```
#include "fsm_driver.h"
```

Init-metoden initierar hårdvaran samt tillståndsmaskinen.

```
void fsm_driver_init(fsm_driver *this)
{
    //---Hardware init---
    bDDR_DRIVER.actuatorStart=1;
    bDDR_DRIVER.actuatorBuzzer=1;
    this->state = FMS_DRIVER_STATE_IDLE;
}
```

Run-metoden skall anropas periodiskt och administrerar hela tillståndsmaskinen.

```
void fsm_driver_run(fsm_driver *this)
{
    char eventDoorOpen = bPIN_DRIVER.sensorDoor;
    char eventBeltOn = bPIN_DRIVER.sensorBelt;
    char eventChair = bPIN_DRIVER.sensorChair;

    switch ( this->state )
    {
        case FMS_DRIVER_STATE_IDLE:
```

14. FSM-implementering

```
if ( eventDoorOpen ) this->state = FMS_DRIVER_STATE_DOOR_OPEN;
break;
case FMS_DRIVER_STATE_DOOR_OPEN:
if ( !eventDoorOpen )
this->state = FMS_DRIVER_STATE_IDLE;
else if ( eventChair )
this->state = FMS_DRIVER_STATE_SITTING;
break;
case FMS_DRIVER_STATE_SITTING:
if ( !eventChair )
this->state = FMS_DRIVER_STATE_DOOR_OPEN;
else if ( eventBeltOn )
{
    this->state = FMS_DRIVER_STATE_WRONG_ORDER;
    bPORT_DRIVER.actuatorBuzzer=1;
}
else if ( ! eventDoorOpen )
this->state = FMS_DRIVER_STATE_DOOR_CLOSED;
break;
case FMS_DRIVER_STATE_DOOR_CLOSED:
if ( eventBeltOn )
{
    this->state = FMS_DRIVER_STATE_START_MOTOR_ALLOWED;
    bPORT_DRIVER.actuatorStart=1;
}
else if ( eventDoorOpen )
this->state = FMS_DRIVER_STATE_SITTING;
break;
case FMS_DRIVER_STATE_START_MOTOR_ALLOWED:
if ( ! eventBeltOn || eventDoorOpen )
{
    this->state = FMS_DRIVER_STATE_IDLE;
    bPORT_DRIVER.actuatorStart=0;
}
break;
case FMS_DRIVER_STATE_WRONG_ORDER:
if (eventDoorOpen && !eventBeltOn && !eventChair )
{
    this->state = FMS_DRIVER_STATE_IDLE;
    bPORT_DRIVER.actuatorBuzzer=0;
}
break;
}
```

Del IV.

Objektorienterad programmering i C

Innehåll

15 Modellering av inbyggda system med OOP-metodik i C	102
15.1 Mål	102
15.2 Läsanvisningar	103
15.3 Videos	104
16 Laboration 4: Objektorienterad programmering i C	105
16.1 Objektorienterad modellering av en mönstergenerator	107
16.2 Kast av tärningar	109
16.3 Avläsning av spänningen från en potentiometer	112
16.4 Styrning av ringräknarens blinkhastighet med hjälp av en potentiometer	112
16.5 Temperatursensor SMT160 med PWM-utgång	113
16.6 Öka/Minska-knappar med acceleration	115
16.7 Morse-kodlås	116
16.7.1 En tryckknapp för kort signal och en för lång signal	116
16.7.2 Morsekodlås	116
16.7.3 Äkta Morsekodlås	117

15. Modellerings av inbyggda system med OOP-metodik i C

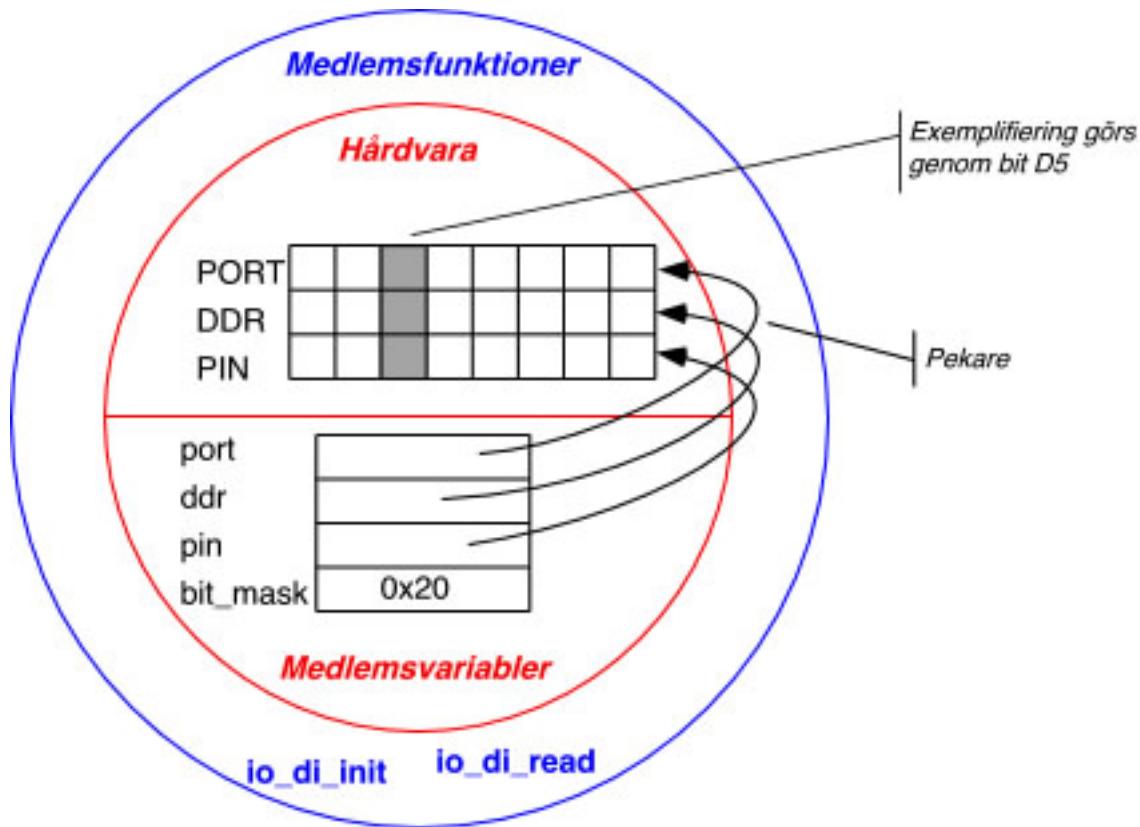
Innehåll

15.1 Mål	102
15.2 Läsanvisningar	103
15.3 Videos	104

15.1. Mål

Några mål i denna modul:

- Att få färdighet i att använda ej egenkonstruerade klasser för att hantera I/O-enheter. Speciellt behandlar vi I/O-enheter anslutna till digitala in-/utportar, analoga importar och seriella kommunikationsportar.
- Att kunna skapa egna klasser och objekt samt att modellera system med hjälp av klass-, objekt- och samarbetsdiagram i UML.



Figur 15.1.1.: Objektorienterad modellering av en digital ingång

15.2. Läsanvisningar

Kompendiet:

- **Objektorienterad modellering**
 - Kapitel 5: Objektorienterad programmering i C
 - ◊ Tar upp grunden för vår objektorienterade approach i C. Behandlar ett viktigt avsnitt om kodningskonventioner för C-program.
 - ◊ Gå igenom alla uppgifter.
 - Kapitel 6: Modellering av digitala och analoga portar
 - ◊ De mest grundläggande portarna i en mikrokontrollrhet är de digitala portarna. Till dessa kan vi ansluta sensorer (insignaler till mikrokontrollrheten) och aktuatorer (utsignaler från mikrokontrollrheten). Vi tar här upp hur vi kan modellera digitala portar med två klasser: Den första klassen hanterar digitala insignaler och heter `io_di`. Den andra klassen hanterar digitala utsignaler och heter `io_do`. Fördelen med att använda färdiga klasser är

15. Modellering av inbyggda system med OOP-metodik i C

att dessa kapslar in och döljer allt praktiskt hanterande av hårdvaran i ett mjukvaruksikt, vilket förhoppningsvis skall leda till en klarare och mer felfri kodning.

- ◊ Gå igenom alla uppgifter.

- Kapitel 7: Seriell kommunikation

- ◊ Seriell kommunikation är grunden för all kommunikation mellan datorer.
- ◊ Vi tar i detta kapitel upp principerna för att skicka parallellt data bit för bit (seriellt) över en digital kommunikationslänk och sedan i mottagarändan ta emot dessa bitar och omvandla dessa till parallellt data.
- ◊ Klassen `io_usart` är gjord för att hantera en USART-krets för seriell kommunikation i pollningsmod. Denna klass erbjuder grundläggande funktioner för att skicka och ta emot tecken.
- ◊ Gå igenom alla uppgifter.

- Kapitel 8: LCD-displayen

- ◊ En LCD-display som skall anslutas till en mikrokontroller kräver en grupp av digitala signaler, protokollet för att kommunicera med denna är relativt omfattande. Det passar då mycket bra att kapsla in beteendet hos en LCD-display i en egen klass och göra ett enkelt användargränssnitt mot denna.
- ◊ Gå igenom alla uppgifter.

- Kapitel 9: Modellering av system med UML

- ◊ En bild säger mer än 1000 ord. UML erbjuder via ett rikt grafiskt modelleringspråk möjlighet till att beskriva funktion och beteende hos programvara på ett bildmässigt sätt.
- ◊ Gå igenom alla uppgifter.

15.3. Videos

- Microcontroller Tutorial - A Beginners Guide

- Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

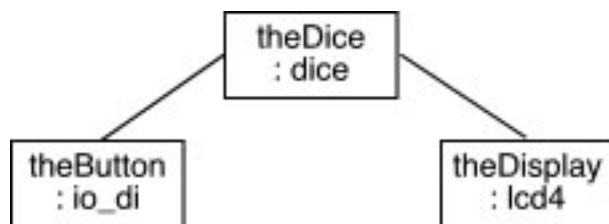
16. Laboration 4: Objektorienterad programmering i C

Innehåll

16.1	Objektorienterad modellering av en mönstergenerator	107
16.2	Kast av tärningar	109
16.3	Avläsning av spänningen från en potentiometer	112
16.4	Styrning av ringräknarens blinkhastighet med hjälp av en potentiometer	112
16.5	Temperatursensor SMT160 med PWM-utgång	113
16.6	Öka/Minska-knappar med acceleration	115
16.7	Morse-kodlås	116
16.7.1	En tryckknapp för kort signal och en för lång signal	116
16.7.2	Morsekodlås	116
16.7.3	Äkta Morsekodlås	117

Mål

- Att kunna lösa mindre problem på ett objektorienterat sätt och använda klass-, objekt- och samarbetsdiagram i UML (Unified Modelling Language) för att grafiskt beskriva en objektorienterad programmeringslösning på problemet.



Figur 16.0.1.: Samarbetsdiagram för de tre objekten theButton, theDice och theDisplay

Läsanvisningar, tips och uppgifter att göra

Kompendiet:

- Objektorienterad modellering

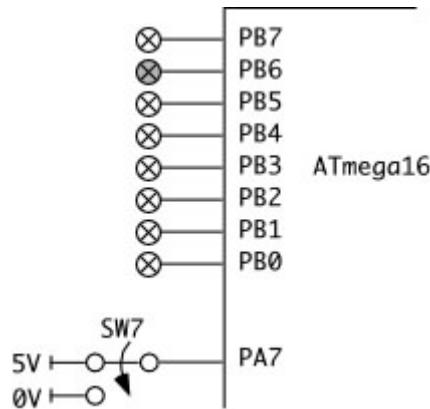
16. Laboration 4: Objektorienterad programmering i C

- Kapitel 6: Modellering av digitala och analoga portar
 - ◊ De mest grundläggande portarna i en mikrostyrenhet är de digitala portarna. Till dessa kan vi ansluta sensorer (insignaler till mikrostyrenheten) och aktuatorer (utsignaler från mikrostyrenheten). Vi tar här upp hur vi kan modellera digitala portar med två klasser: Den första klassen hanterar digitala insignaler och heter io_di. Den andra klassen hanterar digitala utsignaler och heter io_do. Fördelen med att använda färdiga klasser är att dessa kapslar in och döljer allt praktiskt hanterande av hårdvaran i ett mjukvaruksikt, vilket förhoppningsvis skall leda till en klarare och mer felfri kodning.
 - ◊ Gå igenom alla uppgifter.
- Kapitel 8: LCD-displayen
 - ◊ En LCD-display som skall anslutas till en mikrostyrenhet kräver en grupp av digitala signaler, protokollet för att kommunicera med denna är relativt omfattande. Det passar då mycket bra att kapsla in beteendet hos en LCD-display i en egen klass och göra ett enkelt användargränsnitt mot denna.
 - ◊ Gå igenom alla uppgifter.
- Kapitel 9: Modellering av system med UML
 - ◊ En bild säger mer än 1000 ord. UML erbjuder via ett rikt grafiskt modelleringspråk möjlighet till att beskriva funktion och beteende hos programvara på ett bildmässigt sätt.
 - ◊ Gå igenom alla uppgifter.

16.1. Objektorienterad modellering av en mönstergenerator

Problemformulering

Skriv ett program för att kunna blinka åtta lysdioder i olika mönster. Tag med minst 3 olika mönster, förslagsvis bör ring- och johnsonräknarmönster ingå. För att byta mönster skall switch SW7 ansluten till PA7 användas, med denna skall stegning till nästa mönster göras. Arbeta i objektorienterad programmeringsstil.



Figur 16.1.1.: Uppkoppling

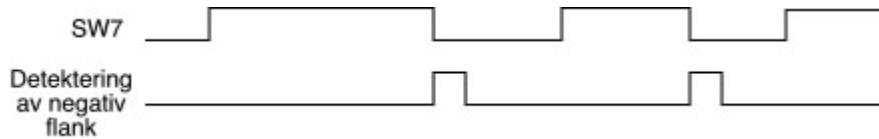
Tips för att lösa problemet

- Klass pattern8.
 - Skapa två filer pattern8.c respektive "pattern8.h", Addera dessa till projekt i AVRstudio.
 - Följande metoder (funktioner) att tillämpa mot ett objekt av denna typ är ett förslag
 - ◊ pattern8_init - Initiering av ett objekt
 - ◊ pattern8_run - Funktion som anropas periodiskt och ger nästa värde i mönstret som skall genereras.
 - ◊ pattern8_select - Val av mönster som skall genereras. Inför en funktion för varje mönster som skall genereras t.ex.
 - ◊ void pattern8_next(pattern8 *this); - Stegar fram och väljer nästa mönster och initierar om objektet.
 - Skapa en metod per mönster som skall kunna genereras
 - ◊ char pattern8_ring(pattern8 *this);'
 - ◊ etc.

16. Laboration 4: Objektorienterad programmering i C

- Klassen pulse

- För att byta mönster skall man känna av när SW7 går från 1 till 0 (nedtryckning av switchen) och då anropa pattern8_next för att byta mönster. Skapa en klass pulse för detta ändamål som ger en utsignal 1 då en 1 till 0 övergång detekteras på. För att kunna detektera en flank måste du minnas föregående avlästa värde på SW7 och jämföra med det aktuella värdet på SW7:



- Strukturen pulse

```
typedef struct
{
    char in;
    char in_old;
    char out;
} pulse;
```

- Metoden pulse_run som skall anropas periodiskt

```
char pulse_run(pulse *this, char in )
{
    this->in_old = this->in;
    this->in = in;

    // Logisk ekvation för att bilda out
    this->out = ????;

    return this->out;
}
```

- Användning av klassen pulse

```
int main()
{
    pulse p7;
    char sw7;

    while (1)
    {
        sw7 = (PORTA & 0x80) != 0; // Avläsning av SW7 på PORTA

        if (pulse_run(&p7, sw7) )
        {
            // ??? Här gör vi något då vi uppfångar en puls
        }
    }
}
```

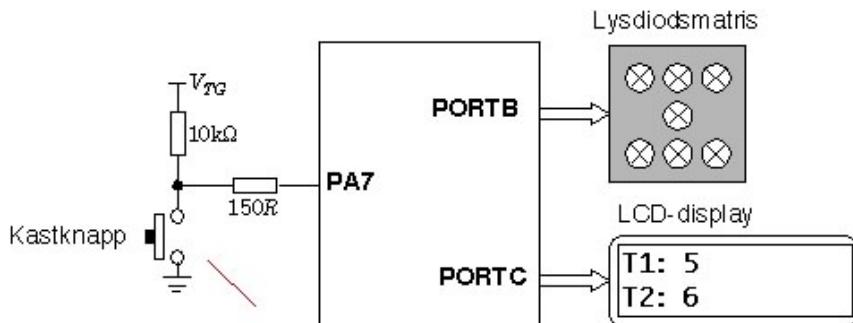
16.2. Kast av tärningar



Figur 16.2.1.: Tärningar

Problemformulering

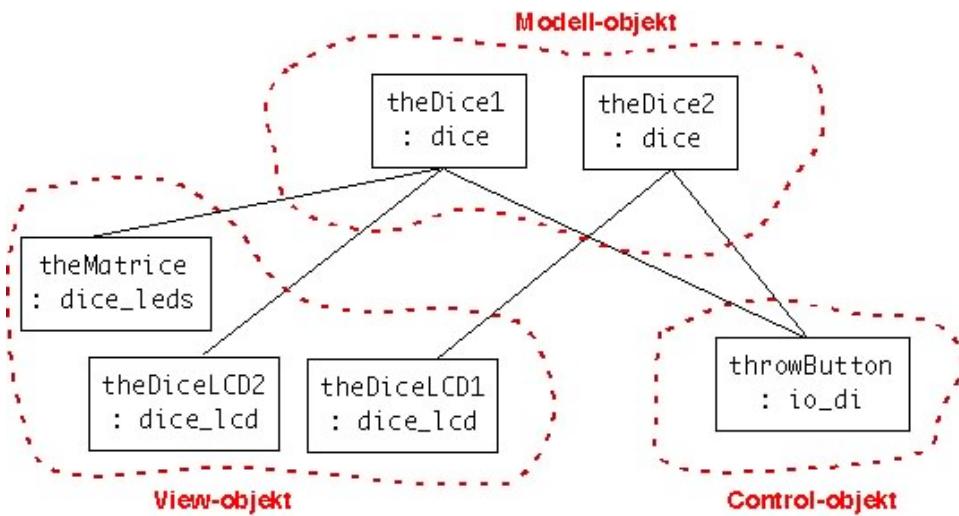
I denna uppgift skall du använda AVR-processorn, lysdioder, LCD-displayen och en tryckknapp för att förverkliga ett system utgörande två tärningar. Den första tärningens värde skall kunna presenteras på två olika sätt: dels på en lysdiodsmatris och dels på rad 1 på LCD-displayen. Den andra tärningens värde skall enbart presenteras på LCD-displayen.



Figur 16.2.2.: Uppkoppling

I den objektorienterade lösningen av detta problem skall vi använda oss av konstruktionsmönstret MVC (Model View Control).

16. Laboration 4: Objektorienterad programmering i C



Figur 16.2.3.: Konstruktionsmönstret MVC

Följande klasser skall finnas med i lösningen:

- Klassen dice (skall skrivas av dig)
 - Våra två tärningar är objekt (variabler) av klassen (datatypen) dice. Klassen dice är en modell av en virtuell tärning som ej vet något om hur den fysiskt är kopplat till en kasstangent eller hur tärningens värde skall presenteras visuellt, denna klass skall du skriva själv.
- Klassen io_di (färdig klass, se kompendiet för hur du använder dess metoder/funktioner)
 - Kastknappen är ett styroobjekt (control) av den färdiga klassen io_di, dess uppgift är att styra om kast av tärningen skall ske eller ej.
- Klassen dice_lcd (skall skrivas av dig)
 - För att呈现出值得 av den virtuella tärningen dice på LCD-displayen finns vy-objekt (view) av klassen dice_lcd. Ett objekt av klassen dice_lcd vet om på vilken rad tärningens värde skall presenteras LCD-displayen, den behöver också ha en referens till ett objekt av klassen lcd4 som används för att göra själva utskriften till LCD-displayen.
- Klassen dice_leds (skall skrivas av dig)
 - Är en klass som vet om hur presentationen av en tärnings värde skall ske på en lysdiodsmatris. Klassen bör också kapsla in information om vilken fysisk port som matrisen är ansluten till.

16. Laboration 4: Objektorienterad programmering i C

Tips för att lösa problemet

För att generera slumptal finns funktionen `random` att tillgå, om du vill använda denna skall du inkludera filen "stdlib.h". Funktionsprototypen ser ut på följande sätt:

```
long random (void);
```

För att generera ett slumptalsvärde i intervallet 1 till 6 kan du använda modulusoperatorn %:

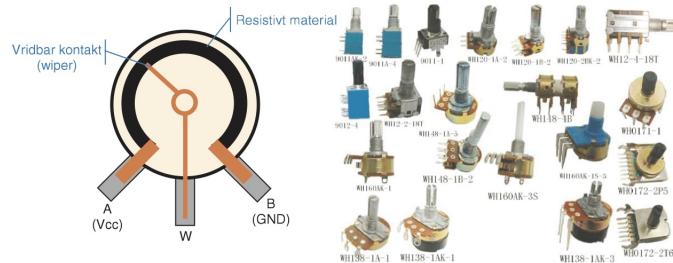
```
value = (random() % 6)+1;
```

16. Laboration 4: Objektorienterad programmering i C

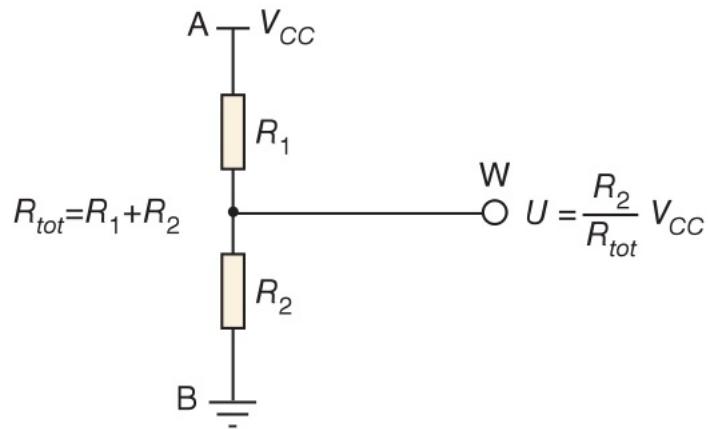
16.3. Avläsning av spänningen från en potentiometer

Problemformulering

Skriv ett program som avläser spänningen från en potentiometer och presenterar spänningsvärdet i Volt med 1 decimal på LCD-displayen.



Figur 16.3.1.: Principen för vridpotentiometern - Variabel spänningsdelning



Figur 16.3.2.: Spänningsdelning

16.4. Styrning av ringräknarens blinkhastighet med hjälp av en potentiometer

Problemformulering

Skriv ett program som med hjälp av en potentiometer kan styra väntloopstidsfördröjningen för hur ofta ringräknaren skall förändra sitt värde. Låt 0V motsvara en tidsfördröjning på 100 millisekunder och 5V en tidsfördröjning på 1 sekund.

16.5. Temperatursensor SMT160 med PWM-utgång

Teori om sensorn SMT160

Innan du går på uppgiften bör du läsa igenom följande sidorn om sensorn SMT160:

- SMart Temperature sensor 160

Observera att pseudo-koden i skriven i programmeringsspråket C i detta dokument skall du ej använda dig av då denna ger ett offsetfel. Problemet består i att avläsning av en 1:a respektive 0:a ej kan garanteras att ta exakt lika lång tid exekveringsmässigt om du använder t.ex. en if-sats..

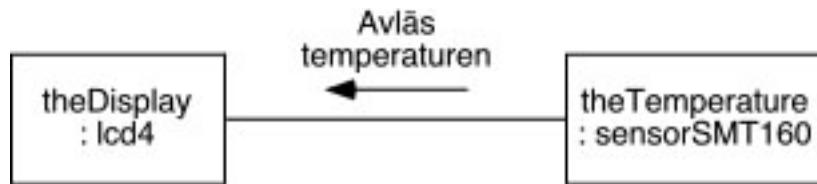
Problemformulering

Skriv ett program som använder temperatursensorn SMT160 för att mäta temperaturen och presentera denna på en LCD-display.

Sensorn SMT160 modellerar du mjukvarumässigt med en klass sensor_smt160.

Applikationen kommer att innehålla minst 2 stycken objekt: Ett objekt av klassen lcd4 med namnet theDisplay och ett objekt av klassen sensor_smt160 med namnet theTemperature.

Samarbetsdiagrammet för interaktionen mellan dessa två blir enligt det som följer i figuren:



Figur 16.5.1.: Samarbetsdiagram i UML

Klassen sensorSMT160

För klassen skall du skapa en källkodsfil med namnet "sensorSMT160.c" respektive en inkluderingsfil med namnet "sensorSMT160.h", dessa filer skall läggas till i ditt projekt.

När du skriver klassen skall du ha i baktanke att i en större tillämpning av typen temperaturreglering kan du ha ett 10-tal sensorer som skall läsas av. Hur skriver du klassen så att du på ett enkelt sätt kan hantera sensorer anslutna till olika portar och bitar i dessa?

- Klassen sensorSMT160
 - Metoder
 - ◊ sensorSMT160_init
 - ▷ Metod för att initiera objekten.

16. Laboration 4: Objektorienterad programmering i C

- ◊ float sensorSMT160_read(sensorSMT160 *this)
 - ▷ Metod för att avläsa PWM-signalen från sensorn och beräkna tempe-raturen.
- ◊ int sensorSMT160_sample (sensorSMT160 *this)
 - ▷ Metod för att sampla PWM-signalen n ggr, returnerar antalet gånger som 1 avlästs.
 - ▷ Denna metod anropas i sensor_smt160_read.
- ◊ sensorSMT160_testSignalChange (*kan kodas i samband med labora-tion 6, temperaturreglering av femrumsvillan*)
 - ▷ Metod för att testa att vi har en sensorsignal på den digitala ingången. Om signalen ligger på konstant nivå under längre tid än t.ex. 10 ms är det någon form av fel på sensorn.
- ◊ sensorSMT160_testSignalFrequency (*kan kodas i samband med labora-tion 6, temperaturreglering av femrumsvillan*)
 - ▷ Frekvensen på signalen från sensor skall vara i intervallet 1-4 kHz, om den ej är det har vi troligtvis ett sensorfel. Metoden testar om frekvensen på sensorsignalen ligger i givet intervall.

Samplingsrutinen för att bestämma pulsperiodtidsförhållandet kan göras på många olika sätt. Ett sätt är att läsa av sensorsignalen under en längre tid och beräkna antalet gånger som man läser av 1:or och sedan beräkna pulsperiodtidsförhållandet som antalet 1:or / totala antalet avläsningar. Detta kan göras både i C respektive assembler. Om man gör det i C bör räkningen av antalet 1:or ej göras med syntax som genererar hopp, vilket troligtvis kommer att ge ett fel då avläsning av en 1:a ej tar lika lång tid som avläsning av en 0:a.

En annan lösning på problemet är göra en positiv flankdetektering och räkna antalet gånger man avläser 1 i en loop och direkt därefter i en annan loop räkna antalet gånger man avläser 0. Denna lösning är snabbare då man endast använder en period och tar för en fungerande sensor mellan 0.25 ms till 1 ms.

Följande samplingsrutin skriven med hjälp av inline-assembler har Örjan Eskilsson bidragit med, en kompakt och elegant lösning.

För att undvika att funktionen blir en inline-funktion kan attributet `noinline` klistras på funktionen:

```
__attribute__((noinline))
unsigned int sensor_smt160_sample(
    volatile unsigned char *pin, //Register R25:R24
    unsigned char bit_mask, //Register R22
    unsigned int n //Register R21:R20
)
{
    asm volatile (
        " movw r30, r24 \"\n\t"
```

16. Laboration 4: Objektorienterad programmering i C

```
" movw r26, r20 " "\n\t"
" ldi r24, 0x00 " "\n\t"
" ldi r25, 0x00 " "\n\t"
" ldi r19, 0xff " "\n\t"
" ldi r20, 0x00 " "\n\t"
"loop: " "\n\t"
" ld r18, z " "\n\t"
" and r18, r22 " "\n\t"
" add r18, r19 " "\n\t"
" adc r24, r20 " "\n\t"
" adc r25, r20 " "\n\t"
" sbiw r26, 0x01 " "\n\t"
" brne loop " "\n\t"
" ret " "\n\t"
);
}
```

Antag att sensorn SMT160 är ansluten till pinne PB7 på PORTB. För att peka ut pinnen PB7 används bitmasken 0x80, läsning av insignalen görs via port in-registret PINB. Ett funktionsanrop för att beräkna puls-period-tidsförhållandet görs då på följande sätt:

```
float dc; //Duty cycle
unsigned int n = 40000; //Number of readings
dc = (float) sensor_smt160_sample( &PINB, 0x80, n) / (float) n;
```

Tips för att lösa problemet

- Det som karakteriseras en SMT160-signalen sett från dator är att detta är en digital insignal. Kika på klassen io_di (io_di.c, io_di.h) för att se hur en allmän digital ingång hanteras.
- Gradertecknet på LCD-displayen har koden 0xDF. Använd metoden lcd4_write_char för att skriva ut tecknet.
- Vanliga fel
 - Den digital insignalen från temperatursensorn skall läsas från PINB-registret (Port IN C) och ej utdataregistret PORTB.
 - Heltalsdivision används för att beräkna duty cycle (ex: 25000/40000 är lika med 0), se till att du använder flyttalsdivision.

16.6. Öka/Minska-knappar med acceleration

Ofta har man behov av att kunna ställa in en parameter med hjälp av öka/minska knappar. Ett mindre problem kan vara att det tar väldigt lång tid att ställa in rätt värde om ökning av parametervärdet är konstant. I denna uppgift är det tänkt att implementera öka/minska-funktionen med acceleration. Antag att inställningen av parametern görs i promilleskal dvs från 0 till 1000. De första 12 sekunderna som öka-knappen hålls nedtryckt

16. Laboration 4: Objektorienterad programmering i C

skall uppräkningen ske med +1 varje sekund, därefter med +1 varje 0.1 sekund tills knappen släpps. På analogt sätt skall minska-knappen fungera. Presentera värdet på en LCD-display eller i ett terminalfönster.

I denna uppgift kan du känna dig fri att realisera en ännu bättre accelerationsfunktion än ovan. Lös uppgiften i en objektorienterad programmeringsstil och med hjälp av en tillståndsmaskin.

16.7. Morse-kodlås

16.7.1. En tryckknapp för kort signal och en för lång signal

I denna uppgift skall du konstruera en tillståndsmaskin för att detektera ett Morse-tecken. Tillståndsmaskinen bör innehålla någon slags timeout-funktion som återställer (reset) den om ingen Morse-signal kommer inom tidsintervallet 10 sekunder max mellan två signaleringar.

1. Tillståndsmaskin för att detektera ett Morse-tecken

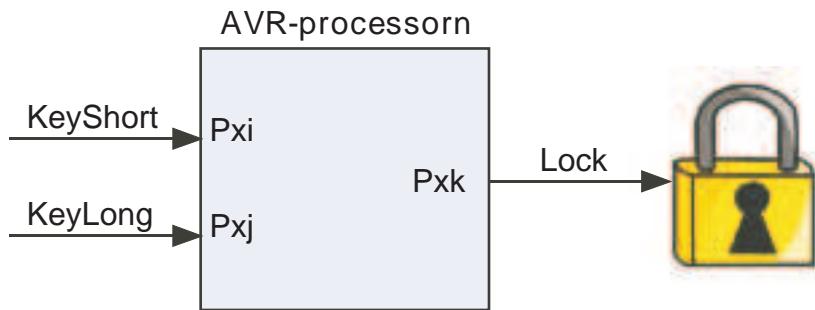
Tecken	Kod
0	- - - -
1	. - - -
2	. . - -
3	. . . -
4
5
6	- . . .
7	- - . .
8	- - - .
9	- - - -

Figur 16.7.1.: Morsetabell

16.7.2. Morsekedlås

1. Tillståndsmaskin för att detektera ett Morse-tecken
2. Tillståndsmaskin för att hantera kodlåset.

16. Laboration 4: Objektorienterad programmering i C

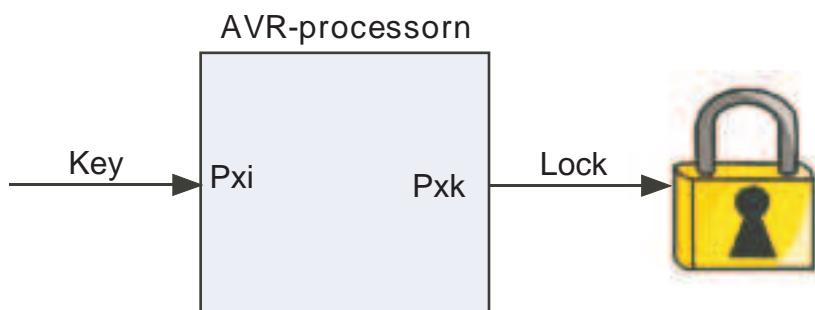


Figur 16.7.2.: Kodlås

16.7.3. Äkta Morsekodlås

I denna uppgift skall du realisera ett kodlås enligt Morsekod-principen med enbart en tryckknapp för att generera korta respektive långa signaler. I denna uppgift är det klokt att arbeta med flera tillståndsmaskiner:

1. Tillståndsmaskin för att detektera en kort respektive lång signal.
 - ButtonMorse.7z
 - Exempel gjort i C++ som visar på en lösning av detta problem.
2. Tillståndsmaskin för att detektera ett Morse-tecken
3. Tillståndsmaskin för att hantera kodlåset.



Figur 16.7.3.: Äkta Morse-kodlås

Del V.

Introduktion till realtidsprogrammering

Innehåll

17 Avbrottshantering	121
17.1 Mål	121
17.2 Läsanvisningar	122
17.3 Videos	123
18 Laboration 5: Objektorienterad modellering och avbrottshantering	124
18.1 Räkning av pulser med en avbrottssrutin	126
18.2 Elektronisk klocka	128
18.3 Cykeldator	130
19 Realtidsprogrammering och konstruktion av små inbyggnadssystem	131
19.1 Mål	131
19.2 Läsanvisningar	131
19.3 Videos	132
20 Projekt: Styrning och övervakning i en villa	133
20.1 Temperaturreglering och dataloggnings - betyg 4	135
20.2 Temperatursensorn DS1620	137
20.3 Temperaturreglering av ett rum	138
20.4 Temperaturreglering av tre rum och statistiskt exekveringschema	140
20.5 Realtidsklockan DS1302	141
20.6 Seriell kommunikation med en PC-dator	141
20.7 Grafiskt gränssnitt	141

INNEHÅLL

20.8 Sensorer för övervakning - betyg 5	142
20.9 Temperaturreglering görs med termostatfunktionen i DS1620	142
20.10 Dataloggning	142
20.10.1 Hantering av EE2PROM-minnet på 1kBytes	142
20.10.2 Organisering av minnet	143
20.10.3 Task för loggning	144
20.11 Ultraljudssensor	145
20.12 Kapacitiv närhetsdetekteringssensor	146

17. Avbrottshantering

Innehåll

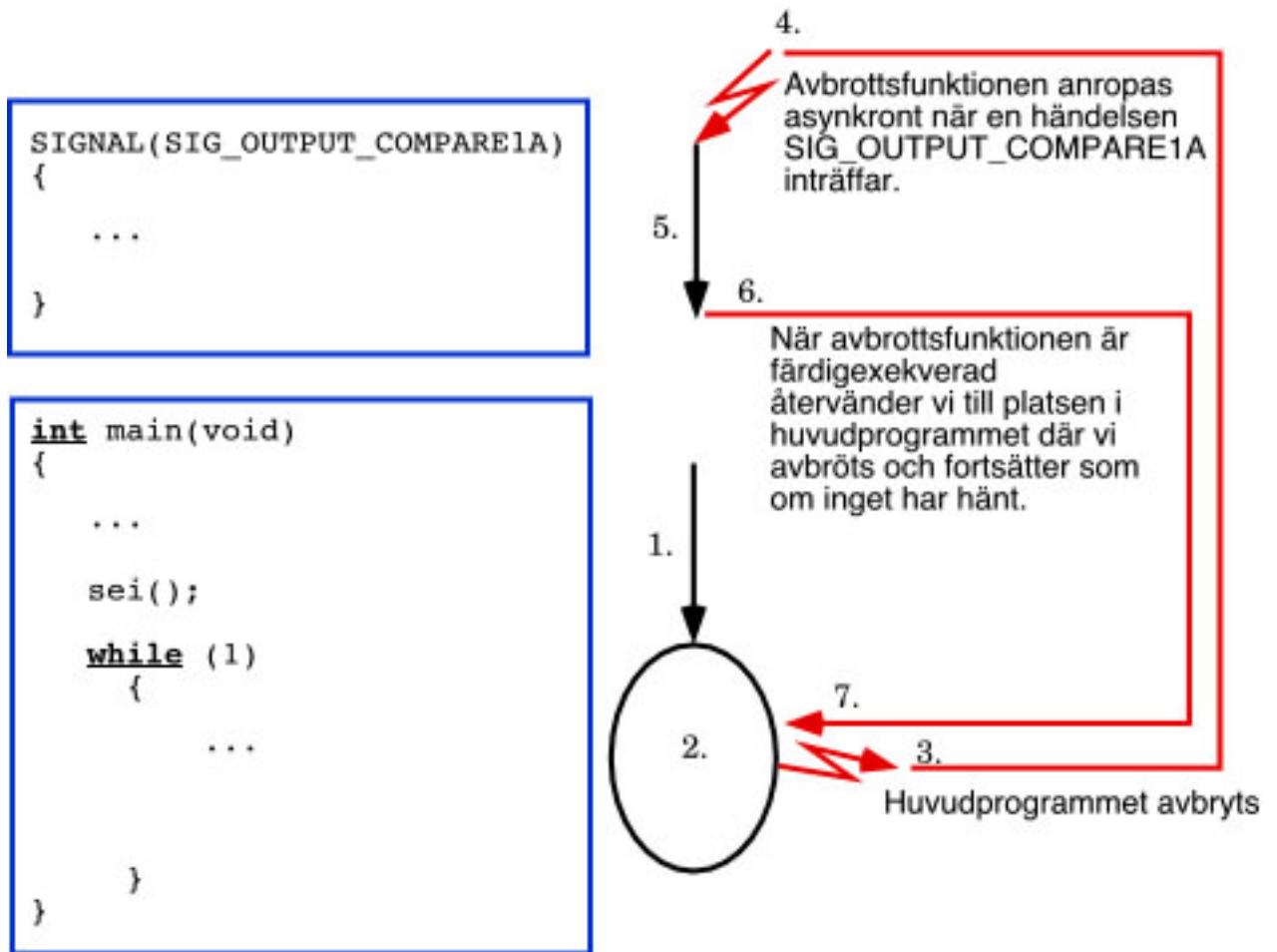
17.1 Mål	121
17.2 Läsanvisningar	122
17.3 Videos	123

17.1. Mål

Några mål i denna modul:

- Att få färdighet i att använda avbrott och avbrottshantering som ett alternativ i problemlösningsfasen.
- Att få en insikt om problem som kan uppstå då (pseudo) parallellexekverande aktiviteter delar en gemensam resurs och hur detta principiellt lösas.

17. Avbrottshantering



Figur 17.1.1.: Pseudoparallell exekvering av flera "program" med hjälp av avbrottfsfunktioner

17.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- [Objektorienterad modellering](#)
 - Kapitel 10: Avbrott och avbrottshantering
 - ◊ I/O-enheter kan betjänas på två olika sätt: avbrott eller pollning. Avbrott och avbrottfsfunktioner (avbrottstask) erbjuder möjligheter till att lösa mjukvaruproblem på ett enklare sätt.
 - ◊ Gå igenom alla uppgifter.

17.3. Videos

- Microcontroller Tutorial - A Beginners Guide
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

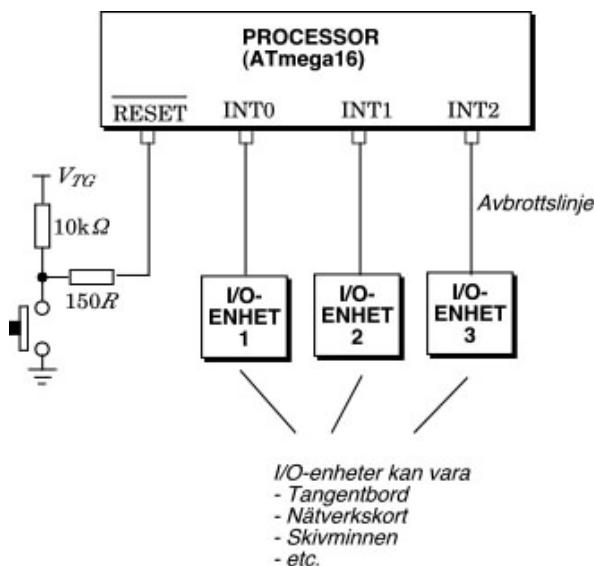
18. Laboration 5: Objektorienterad modellering och avbrottshantering

Innehåll

18.1 Räkning av pulser med en avbrotsrutin	126
18.2 Elektronisk klocka	128
18.3 Cykeldator	130

Mål

- Att förstå fördelar och nackdelar med att använda pollning respektive avbrott för att ta hand om och svara på händelser som inträffar i olika I/O-enheter i datorsystemet. Färdighet i att implementera avbrottsfunktioner i C.



Figur 18.0.1.: Ingångar för extern avbrottssignalering på en ATmega16/32-processor

Läsanvisningar, tips och uppgifter att göra

- Kompendiet: [Objektorienterad modellering](#)

18. Laboration 5: Objektorienterad modellering och avbrottshantering

- Kapitel 5: Objektorienterad programmering i C
- Kapitel 6: Modellering av digitala och analoga portar
- Kapitel 7: Seriell kommunikation
- Kapitel 8: LCD-displayen
- Kapitel 9: Modellering av system med UML
- Kapitel 10: Avbrott och avbrottshantering

Lite mer att läsa om avr-gcc-kompilatorn och hur denna gör med avbrott

OBServera att från 2007 rekommenderas att använda nya namngivningsregler för avbrottsfunktioner. I kompendiet och i många exempel använder jag de gamla reglerna som fortfarande gäller men ej rekommenderas: http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Exempel på det NYA sättet att definera en avbrottsfunktion:

```
#include <avr/interrupt.h>

ISR( INT0_vect )
{
    // user code here
}
```

GAMLA sättet:

```
#include <avr/interrupt.h>
SIGNAL( SIG_INTERRUPT0 )
{
    // user code here
}
```

18.1. Räkning av pulser med en avbrottssrutin

Teorin för avbrottssrutiner finns i kapitel 10 i kompendiet "Inbyggda system och OBJEKTORIENTERAD MODELLERING".

Problemformulering

Vi skall i denna uppgift använda oss av INT0 (INTerrupt0, pinne PD2) och räkna antalet pulser som kommer in på denna ingång. Själva pulsräkningen skall ske i en avbrottssrutin som anropas varje gång en positiv flank kommer på INT0-pinnen (PD2). Presentationen av antalet räknade pulser sker på en LCD-display. Vi kommer i denna uppgift att ha två parallelexekverande program: Ett i form av avbrottsfunktionen och ett i form av main-funktionen.

För att hantera räkningen av antalet pulser skall en generell counter-klass skapas. Klassen bör ha följande uppbyggnad:

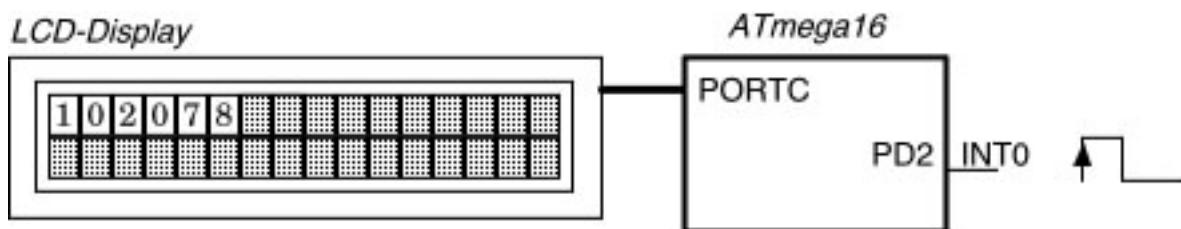
```
typedef struct
{
    int max; //Maximalt värde innan vi börjar om från min.
    int min; //Minimalt värde.
    int value; //Aktuellt värde
} counter;
```

och metoder

```
void counter_init(counter *this, int max, int min); //Initiering av
//objektet
int counter_inc(counter *this); //Öka på räknarens värde med 1, om vi
//kommer över
//max så sätter vi value till min och returnerar
//att minnessiffra (carry) är sant.
int counter_dec(counter *this); //Minska räknarens värde med 1.
int counter_read(counter *this); //Läs av räknarens värde.
```

Testuppkoppling

För att kunna generera pulser på pinne PD2 kopplar du om strömställarna som normalt är anslutna till PORTA till PORTD. Genom att trycka och släppa SW2 kommer du att generera en positiv flank (0 till 1 övergång) på pinne PD2, vilket gör att avbrottss rutinen kommer att anropas.



Figur 18.1.1.: Uppkoppling

18. Laboration 5: Objektorienterad modellering och avbrottshantering

Programskellet

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "lcd4.h"
#include "counter.h"
//== Global objects =====

//???

//==Interrupt Service Routine for INT0 =====
SIGNAL(SIG_INTERRUPT0)
{
    // ???
}

/*== main =====
Pulse counter example
INT0 = pin PD2
INT1 = pin PD3
INT2 = pin PB2
=====*/
int main(void)
{
    // ???

    //--- Programming of registers MCUCR and GICR
    // see Technical Reference Manual

    // ???

    //--- Enable interrupts ---
    sei();
    while (1)
    { //Presentation på LCD-displayen
        //???
        _delay_ms(300);
    }
    return 0;
}
```

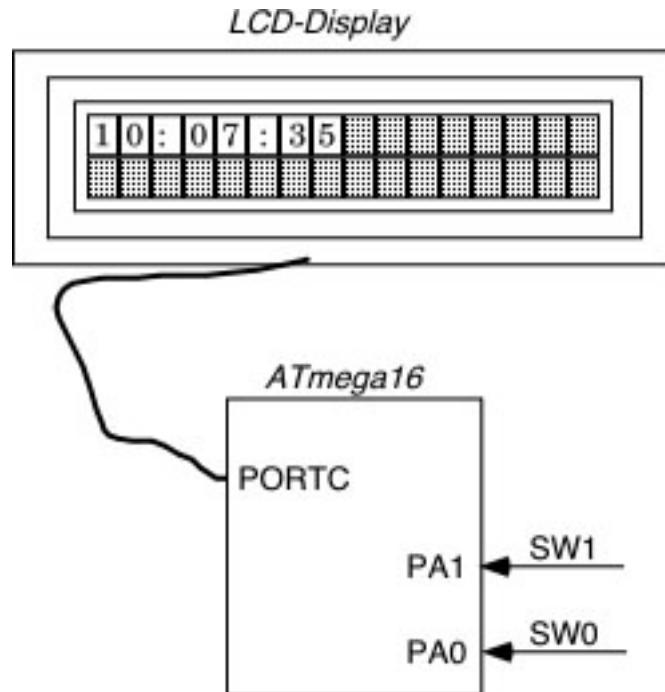
Några frågor

1. Hur skall MCUCR-registret programmeras för att aktivera möjligheten till INT0-avbrott samt att avbrott skall genereras på positiv flank?
2. Producenten av data i vår applikation är avbrottssrutinen för INT0-avbrott som räknar pulser och konsumenten är main-funktionen som presenterar antalet pulser. Datautbytet mellan dessa måste ske med globala variabler. Hur säkerställer vi att läsningen av data från main-funktionen ej kan bli avbrutet?

18.2. Elektronisk klocka

Problemformulering

En elektronisk klocka skall konstrueras. För detta ändamål skall TIMER0 användas för att ge klocktick till klockan. För att kunna ställa klockan används 2 switchar SW1 och SW0. Med SW1 ställer vi timmar och med SW0 ställer vi minuter. Observera att du bör ansluta LCD-displayen på PORTB om



Figur 18.2.1.: Uppkoppling

OBSERVERA Använd ej klassen time som finns beskriven i kompendiet. Det är möjligt att lösa klockproblemet med denna men det krånglar bara till lösningen. Utgå ifrån det som finns i tips-delen som följer.

Tips för att lösa problemet

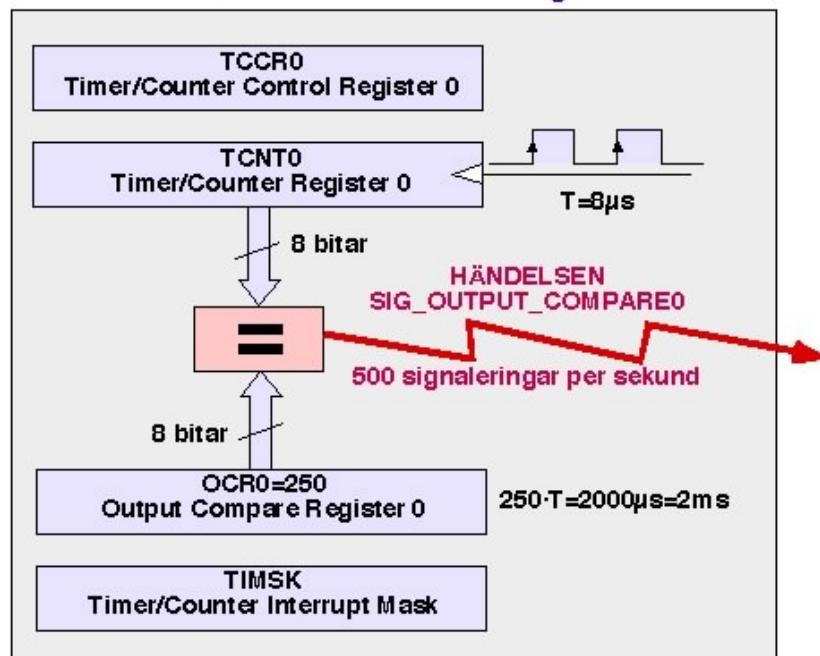
För att komma igång med TIMER0-avbrott kan du kika på följande programskelet. Kika också i "Teknisk manual till Atmel Mega 16 processorn" och avsnittet om TIMER0. För att tolka programskeletet har du nytta av följande figur:

MJUKVARA FÖR ATT INITIERA TIMERO

```
int main()
{
    ...
    bTCCR0.cs0 = 3;      //Clock Select för TIMER0
    //CK=8MHz/64
    bTCCR0.wg01=1;      //Arbetssätt för TIMER0
    //Output Compare Match-mode
    bTIMSK.ocie0 =1;    //Tillåt avbrott från TIMER0

    OCR0 = 250;          //Avbrott var 250:e klockpuls
}
```

HÄRДVARA I PROCESSORN ATmega16



MJUKVARA-AVBROTTSFUNKTION

```
SIGNAL(SIG_OUTPUT_COMPARE0)
{
    ...
}
```

Händelsesignaleringen
triggar exekveringen av
denna funktion 500 gånger
per sekund.

Figur 18.2.2.: Uppkoppling

För att kunna definiera avbrottsrutiner måste du inkludera följande fil:

```
#include <avr/interrupt.h>
```

Med den interna RC-oscillatoren blir klockfrekvensen för CPU:n ej exakt 8MHz, denna kan dock trimmas. Studera i "Teknisk manual till Atmel Mega 16/32 processorn" och

18. Laboration 5: Objektorienterad modellering och avbrottshantering

avsnittet som handlar om OSCCAL-registret.

18.3. Cykeldator

- Slides
 - [Cykeldator](#)

Problemformulering

En cykeldator shall konstrueras.

19. Realtidsprogrammering och konstruktion av små inbyggnadssystem

Innehåll

19.1 Mål	131
19.2 Läsanvisningar	131
19.3 Videos	132

19.1. Mål

Några mål i denna modul:

- Att presentera ideen med task-abstraktionen som ett sätt att förenkla problemlösningen i ett komplext inbyggnadssystem.
- Att visa på ideen med statiska exekveringsscheman för att exekvera periodiska tasks.

19.2. Läsanvisningar

Kompendiet "Inbyggda system med AVR RISC PROCESSORER"

- [Objektorienterad modellering](#)
 - Kapitel 11: Nedbrytning av ett problem i ett antal tasks
 - ◊ Taskabstraktionen kan förenkla problemlösning där en dator skall utföra flera uppgifter, där uppgifterna kan sägas vara oberoende av varann. Lösningen blir då att låta varje uppgift vara ett eget sekventiellt program (funktion).
 - ◊ [Taskabstraktionen](#)
 - Kapitel 12: Statiska exekveringsscheman
 - ◊ Vi behandlar här några olika sätt att exekvera periodiska tasks (uppgifter) med hjälp av en vänteloop och någon forma av klockticksgenerering.
 - ◊ [Statiska exekveringsscheman](#)

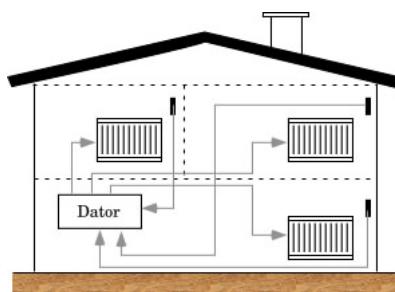
19.3. Videos

- Microcontroller Tutorial - A Beginners Guide
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

20. Projekt: Styrning och övervakning i en villa

Innehåll

20.1 Temperaturreglering och dataloggning - betyg 4	135
20.2 Temperatursensorn DS1620	137
20.3 Temperaturreglering av ett rum	138
20.4 Temperaturreglering av tre rum och statiskt exekveringschema	140
20.5 Realtidsklockan DS1302	141
20.6 Seriell kommunikation med en PC-dator	141
20.7 Grafiskt gränssnitt	141
20.8 Sensorer för övervakning - betyg 5	142
20.9 Temperaturreglering görs med termostatfunktionen i DS1620	142
20.10 Dataloggning	142
20.10.1 Hantering av EE2PROM-minnet på 1kBytes	142
20.10.2 Organisering av minnet	143
20.10.3 Task för loggning	144
20.11 Ultraljudssensor	145
20.12 Kapacitiv närhetsdetekteringssensor	146



Figur 20.0.1.: Uppkoppling för temperaturreglering av ett rum

Alla uppgifter skall lösas med det objektorienterade synsättet. Börja med att läsa igenom kapitel 10-12 i kompendiet innan du kör igång med uppgifterna.

20. Projekt: Styrning och övervakning i en villa

Läsanvisningar, tips och uppgifter att göra

- Kompendiet "Inbyggda system och OBJEKTORIENTERAD MODELLERING"
 - Kapitel 10: Avbrott och avbrottshantering
 - Kapitel 11: Nedbrytning av ett problem i ett antal tasks
 - ◊ [Taskabstraktionen](#)
 - Kapitel 12: Statiska exekveringsschema
 - ◊ [Statiska exekveringsscheman](#)

Dokumentation av projektet

Varje klass skall modulariseras i en källkodsfil och en inkluderingsfil om du arbetar med objektorienterad C och med en inline-klass om du arbetar med C++. Klasserna skall dokumenteras med JavaDoc-kommentarer. HTML-dokumentation av hela systemet skall göras med DoxyGen.

20.1. Temperaturreglering och dataloggning - betyg 4

Laborationen handlar om temperaturreglering av ett hus (tre rum skall regleras) och med där tillhörande loggning av temperaturdata. Följande deluppgifter finns i laborationen:

1. Temperatursensorn DS1620
2. Temperaturreglering av ett rum
3. Temperaturreglering av tre rum och statiskt exekveringsschema
4. Realtidsklockan DS1302
5. Seriell kommunikation med en PC-dator
6. Grafiskt gränssnitt
7. Dataloggning
 - a) Hantering av EEPROM-minnet
 - b) Organisering av minnet
 - c) Task för dataloggning

Innan du kör igång med en uppgift, bör du läsa igenom alla uppgifter för att få ett helhetsintryck av det som skall göras. Detta kommer också att påverka dina dellösningar.

Följande användning av portarna föreslås:

- PORTA PA0 - Används av 3Wire-bussen för den dubbelriktade datasignalen DQ, den används även av den mjukvaruimplementerade SPI-bussen för datasignalen SI (Slave Input).
 - PA1 - Används av 3wire-bussen för klocksignalen CLK, den används även av SPI-bussen för klocksignalen CLK.
 - PA2 - DS1620
 - PA3 - DS1620
 - PA4 - DS1620
 - PA5 - DS1302
 - PA6 - CS-N för EEPROM på 1kBytes
 - PA7 - SO (Slave Output) signalen från EEPROM på 1kBytes.
- PORTB - Används av LCD-displayen
- PORTC P
 - PC0-PC1 fria
 - PC2-PC5 används av JTAG-interfacet och därmed av AVR Dragon

20. Projekt: Styrning och övervakning i en villa

- PC6-PC7 fria
- PORTD
 - PD0-PD1 används för seriell kommunikation till PC-datorn
 - PD2-PD4 används av de tre värmelementen (transistorkopplingen)
 - PD5-PD7 används för tre tryckknappar

Observera att du tjänar mycket på att jobba med flera olika projekt i denna uppgift.

Förutom huvudprojektet bör du ha flera mindre testprojekt som du använder för att testa av hårdvaran.

I fall något ej fungerar bör du kunna verifiera att din uppkoppling av hårdvaran är korrekt.

20.2. Temperatursensorn DS1620

Som temperatursensor kommer vi att använda en krets med namnet DS1620 som vi kommunicerar med genom ett seriellt bussgränssnitt som heter 3wire. Bekanta dig med denna krets genom att läsa och göra första uppgiften i kapitlet om 3wire-bussen:

- [Treträdsbussen - 3wire](#)

här hittar du allt material för att komma igång med denna buss.

20.3. Temperaturreglering av ett rum

Observera att uppgifterna beskrivs i objektorienterad C, du kan valfritt välja att arbeta med OOP i C eller i C++.

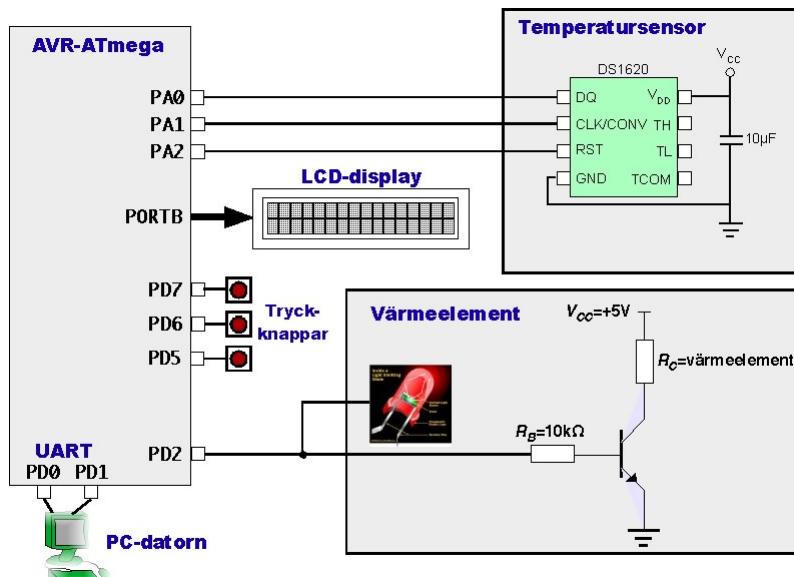
Uppgiften är att ta fram ett program för att reglera temperaturen i ett rum. Använd heltalsaritmetik för regleringen och ej flyttalsaritmetik, vi reglerar temperaturen på hela grader. Skapa en egen klass med namnet task_tc (tc=temperature control) vars uppgift är att klara av temperaturreglering av ett rum. Klassen bör innehålla en initieringsmetod

```
void task_tc_init(task_tc *this, ... );
```

och en main-metod som anropas periodiskt

```
void task_tc_main(task_tc *this);
```

Skriv klassen på sånt sätt den kan lätt kan användas för att reglera fler rum än ett. Detta innebär bland annat att initieringsmetoden i argumentlistan skall ta information om vilka portar som används sensor och värmeelement.



Figur 20.3.1.: Uppkoppling för temperaturreglering av ett rum

Ett objekt av typen task_tc kan t.ex. innehålla följande karakteristika data (medlemsvariabler)

```
typedef struct
{
    io_do heater; //Digital utsignal kopplad till värmeelementet (resistor
                   //Rc i vårt fall)
    ds1620 sensor; //Digital insignal kopplad till sensorn
    int rTemp; //Önskad temperatur
```

20. Projekt: Styrning och övervakning i en villa

```
int yTemp; //Mätt temperatur  
//...  
} task_tc;
```

20.4. Temperaturreglering av tre rum och statiskt exekveringsschema

Skriv ett program som reglerar temperaturen i tre rum i villan. Vi har köket, vardagsrum och ett sovrum. Visning av aktuell temperatur och önskad temperatur skall göras för ett rum i taget på en LCD-display, vilket rum som visas på displayen bestäms med en switch. För att ställa önskad temperatur i rummet finns två switchar, en för att öka önskad temperatur och en för att minska önskad temperatur. Tre stycken temperatursensorer av typen DS1620 finns placerade i rummen, tre styrsignaler för att slå PÅ eller AV värmeelementen finns.

Följande periodiska uppgifter (tasks) har vi:

1. Människa-maskin-gränssnitt, utförs med periodtiden 300 millisekunder.
 - a) Ett objekt av typen task_hmi kan t. ex. innehålla följande karakteristiska data (medlemsvariabler):

```
typedef struct
{
    task_tc *rooms; //Pekare till en vektor av rum
    int i; //Rumsindex, rum utvalt för manipulering
    int n; //Antal rumsobjekt i vektorn
    ...
} task_hmi;
```
 - b) LCD-displayen kan ta lång tid att uppdatera om födröjningsparametrarna som skickas med har för stora värden. Detta kan spräcka ditt exekveringschema. Mät upp tiden för att skriva ut två rader med 16 tecken, minimera värdena på de två födröjningsparametrarna för att tiden för exekvering skall bli minimal.
2. Temperaturregleringsuppgifterna (en för varje rum, tre tasks totalt) utförs med periodtiden 12 sekunder.

Läs i kompendiet om statiska exekveringsscheman och implementera ett sådant för det fyra tasken/uppgifterna i detta problem. Det statiska exekveringschemat kommer sedan att byggas på med andra uppgifter/tasks.

Studera exempelprojektet med ett statiskt exekveringsschema som kan vara till hjälp för att komma igång:

- Startprojekt och exempelprogram - Project static_execution_scheme

20.5. Realtidsklockan DS1302

Som realtidsklocka kommer vi att använda en krets med namnet DS1302 som vi kommunicerar med genom ett seriellt bussgränssnitt som heter 3wire. Bekanta dig med denna krets och skapa en klass ds1302 för att hantera en sådan krets. Metoder för att läsa och ställa klockan är ett måste.

20.6. Seriell kommunikation med en PC-dator

Skapa ett kommandogränssnitt för att kunna kommunicera med en annan dator. Börja med att implementera kommandon för att läsa och ställa realtidsklockan, avläsning av temperatursensorer, etc. Skapa ett task task_sio (serial input output) för detta ändamål. Exekveringen av detta task gör förslagsvis i det statiska exekveringschemat, välj en lämplig periodtid för detta task med beaktande bl. a. av hur ofta ett kommando kan inkomma till processorn.

Studera exempelprojektet seriell kommunikation med hjälp av avbrotsrutiner som kan vara en hjälp för att komma igång:

- [Startprojekt och exempelprogram](#) - Projekt avr_sio_isr

Provör programmet som innehåller 4 kommandon för att manipulera PORTA respektive PORTB. I persondatorändan använder du antingen terminalprogrammet TeraTerm Pro eller Terminal.

FEL: Kolla upp att i inkluderingsfilen qrb.h att makrokonstanten QRB_SIZE är en potens av 2, dvs storleken skall uppfylla kravet 2^N där N är valbart. Om N=7 leder detta till storleken $2^7=128$ som kan vara ett lämpligt värde.

20.7. Grafiskt gränssnitt

Gör ett grafiskt gränssnitt till ditt system för att övervaka och styra din villa. Hur gränssnittet skall se bestämmer du själv. Det grafiska gränssnittet gör vi med C# och seriell kommunikation till vår processor. Det grafiska gränsnittet kommer att interagera mot task_sio i processorn.

Studera projektet för seriell kommunikation via en COM-port som är programmerad i C#. Programmet använder sig av klassen SerialPort för att kunna koppla sig till någon av datorns COM-portar: [Startprojekt och exempelprogram](#).

20.8. Sensorer för övervakning - betyg 5

Vårt program från föregående uppgift skall byggas på med ny funktionalitet i form av några olika typer av sensorer, dataloggning och ett bättre utnyttjande av DS1620-kretsen i form av att denna själv sköter om temperaturregleringen.

20.9. Temperaturreglering görs med termostatfunktionen i DS1620

Utnyttja möjligheten att göra temperaturregleringen i DS1620, koppla om och programmera kretsen så att all reglering av temperaturen görs med DS1620. Läs mer i databladet om skrivning till TL- och TH-registren, läs mer om skrivning till det ickeflyktiga minnet och biten NVB i status- och kontrollregistret. Hur lång tid kan en skrivning ta till minnet?

20.10. Dataloggning

20.10.1. Hantering av EE2PROM-minnet på 1kBytes

För dataloggning har vi tillgång till en krets med minnesutrymmet 1kBytes, kommunikation sker via SPI -bussen. Minnet är minimalt och är valt enbart för att det är billigt. Det går att köpa minnen som är bra mycket större och mer användbara i dataloggningssammanhang, för vårat ändamål duger det mycket bra då det är det totala systemet och dess programmering som intresserar oss.

För att hantera SPI-bussen behöver vi programvara som är analog med Wire3-bussens grundläggande programvara. För att spara pinnar kommer vi att använda 3Wire-bussens pinnar för DQ och CLK för SPI-bussens signaler SI och CLK. Börja med en bitfältsbeskrivning som anger hur minneskretsen skall kopplas till

```
typedef struct
{
    unsigned char SI : 1;      //DO, Digital Utgång
    unsigned char CLK : 1;     //D1, Digital utgång
    unsigned char unused : 4; //D2-D5
    unsigned char CS_N : 1;   //D6, aktiv låg, Digital utgång
    unsigned char SO : 1;     //D7, Digital ingång
} spi_pins;
```

Läs mer om EEPROM:et i databladet. Som hjälp att komma igång med detta minne finns ett AVR-projekt. Observera att i exemplet skrivas endast en byte åt gången, i din dataloggning bör du skriva alla byten i en post i ett sammanhang. Läs mer om hur skrivningen fungerar till minnet i databladet.

- Datablad för EEPROM
- Projekt

20. Projekt: Styrning och övervakning i en villa

Följande uppdateringar har gjorts:

- 2011-02-23

Exempelprojektet uppdaterat med en metod eeprom_write_vect för att skriva godtyckligt antal bytes till EEPROM:et. Metoden löser en del praktiska problem när det gäller skrivning till minnet som rör dess sidorienterade (page) funktionalitet.

- 2011-03-10

Problem kan uppstå då WIRE3- och SPI-bussen delar biten D0 i PORTA. För SPI-bussen gäller att signalen SI alltid är en digital utgång, för WIRE3-bussen är DQ dubbelriktad. Se till att sätta datariktningregistret SPI_DDR.si=1 innan någon operation på SPI-bussen görs. Observera att om detta ej görs och en läsning av har gjorts med 3-wire bussen (DS1302/DS1620) kommer denna operation ha gjort PA0 till en ingång vilket gör att SPI-bussen ej fungerar som tänkt. Modifiera därför lämplig funktion i "spi.c" så SPI-bussen fungerar i alla väder.

20.10.2. Organisering av minnet

Organisera minnet på sätt att du kan lagra maximalt med temperaturvärden (heltalsdata) och även en tidsstämpel. Tänk igenom något smart sätt att administrera minnet. Här kommer några punkter som kan vara värda att tänka på:

- Två typer av data att hantera: temperaturdata och tidsdata
- Antag att det intressanta temperaturintervallet är mellan 10 och 30 grader Celsius.
 - Använd 5 bitar för att koda temperaturen.
 - Koda även in att temperaturen är lägre än 10 grader och större än 30 grader.
 - Totalt 15 bitar skulle åtgå för detta.
 - Den 16:e biten skulle kunna användas bl.a. för att indikera att inget temperaturdata finns lagrat och/eller indikera tidsdata.
- Tidsstämpeln behöver inte sparas med varje post av temperaturvärdet.
 - Om samplingstidsintervallet ändras bör klockan avläsas och sparas undan tillsammans med det nya samplingstidsintervallet.
- Använd bitfältstrukturen för att undelätta hanteringen.
- Reservera ett antal bytes i början av minnet för administration av resten av minnet.
 - Skrivpekaren för en ringkö
 - Aktuellt samplingstidsintervall (kan väljas mellan 1 till 30 minuter)
 - Tidsstämpel för senast avlästa temperaturpost. Denna information kan användas för att avgöra om ny tidstämpel bör skrivas in i ringkönen.
- etc.

20. Projekt: Styrning och övervakning i en villa

Beroende på hur du organiserar minnet bör du kunna spara upp emot 500 temperaturposter.

20.10.3. Task för loggning

Implementera ett task_log som sköter om hela datalogningen. Antag att loggning av temperaturer skall kunna vara valbart i intervallet 1-60 minuter.

20.11. Ultraljudssensor

För övervakningsändamål finns i husets garage en ultraljudssensor tänkt att användas för att registrera otillbörligt intrång i lokalen. Fundera kring hur du vill montera sensorn i garaget, hur ofta du behöver läsa av den, etc.?



Figur 20.11.1.: Ultraljudssensorn SRF05

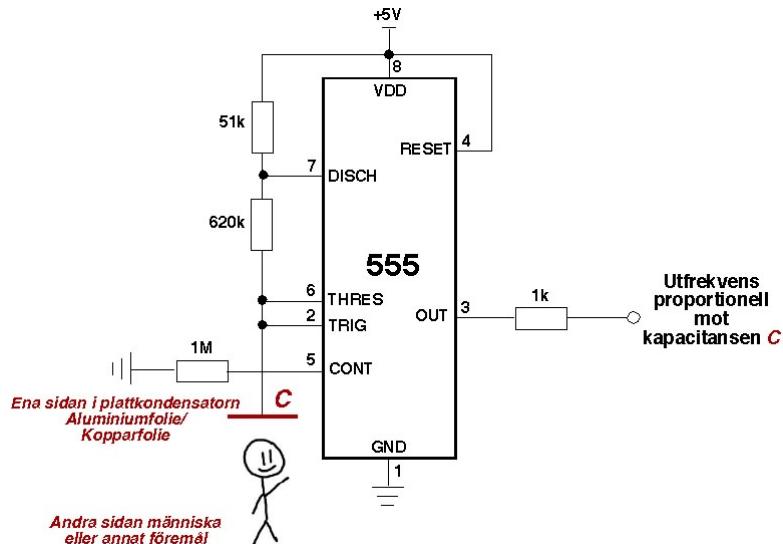
Komplettera ditt C#-program så att det presenterar sensorsignalen på något bra sätt.

Att läsa

- Wikipedia Ultraljud: <http://en.wikipedia.org/wiki/Ultrasound>
- Sensorn SRF05, webbsida: <http://www.robot-electronics.co.uk/htm/srf05tech.htm>

20.12. Kapacitiv närväxtdetekteringssensor

Konstruera en kapacitiv närväxtdetekteringssensor som skall användas i en familj med en spelberoende tonårsunge för att avgöra om han sover på nätterna eller sitter vid sin dator. Ideen bakom sensorn är enkel, vi skall tillverka ena sidan av en plattkondensator med aluminiumfolie eller kopparfolie. Den andra sidan av plattkondensatoren utgörs av en människa i närväxten av folien.



Figur 20.12.1.: Kopplingschema för en kapacitiv närväxtdetekteringssensor

Använd gärna en 555-kalkylator för att förstå hur kretsen fungerar:

- [555 Calculator](#)
- [555 Astable Circuit Calculator](#)

Komplettera ditt C#-program så att det presenterar sensorsignalen på något bra sätt.

Del VI.
Seriella bussar

Innehåll

21 Treträdsbussen - 3wire	149
21.1 Att läsa	149
21.2 Introduktion	149
21.3 Treträdsbussens programvara	150
21.3.1 Skikt 0 - Inkluderingsfil wire3.h	152
21.3.2 Skikt 1 - Källkodsfil wire3.c	152
21.3.3 Skikt 2 - Källkodsfil wire3.c	154
21.3.4 Skikt 3 - Applikationsskiktet	156
21.4 Uppgifter	156
21.4.1 Avläsning av temperatursensorn i DS1620	156
21.4.2 Realtidsklockan DS1302	157
22 RS232	159
22.1 Att läsa	159

21. Treträdsbussen - 3wire

Innehåll

21.1	Att läsa	149
21.2	Introduktion	149
21.3	Treträdsbussens programvara	150
21.3.1	Skikt 0 - Inkluderingsfil wire3.h	152
21.3.2	Skikt 1 - Källkodsfil wire3.c	152
21.3.3	Skikt 2 - Källkodsfil wire3.c	154
21.3.4	Skikt 3 - Applikationsskiktet	156
21.4	Uppgifter	156
21.4.1	Avläsning av temperatursensorn i DS1620	156
21.4.2	Realtidsklockan DS1302	157

21.1. Att läsa

- DS1620 - Temperatursensor och termostatreglering
 - [Datablad](#)
 - [Applying and Using the DS1620 in Temperature Control Applications](#)
 - [Temperature Sensor Eludes Analog Interfacing](#)
- DS1302 - Realtidsklocka
 - [Datablad](#)
- Slides
 - [3wire-bussen](#)

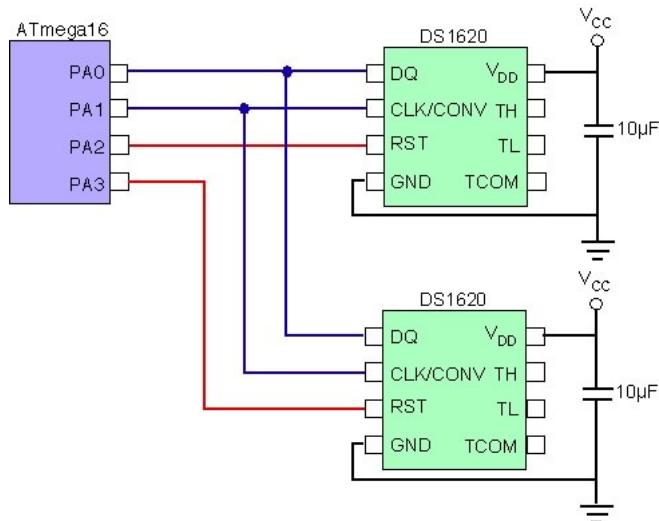
21.2. Introduktion

Treträdsbussen är en extremt enkel seriell buss för kommunikation mellan kretsar på ett kretskort. Den här typen av seriell kommunikationsbuss brukar också benämnas chip-to-chip communication. En krets med detta gränssnitt har tre signalledningar:

21. Tretrådsbussen - 3wire

- DQ
 - Data In-/Output, dubbelriktad dataledning för överföring av kommandon och data, vilket möjliggör både läsning respektive skrivning.
- CLK
 - Klocksignal genererad från mastern
- RST-N
 - Reset Input. Om denna signal är LÅG (0:a) till kretsens kan ingen kommunikation ske med denna. När signalen går från LÅG till HÖG så är kretsen utpekad för en kommunikationssession. Varje krets som är anslutet till bussen har en egen unik RST-N signal.

Figuren som följer visar på hur två temperatursensorkretsar av typen DS1620 har anpassats till processorn ATmega16. Observera att bussignalerna DQ och CLK är gemensamma för alla som är anslutna till bussen. Däremot har varje krets en egen unik RST-N signal.



Figur 21.2.1.: Uppkoppling av två temperatursensorer av typen DS1620 mot en ATmega-processor

Bussen är av typ master-slave, där ATmega-processorn är mastern som styr över allt på bussen. Mastern bestämmer med vilken slav den vill prata med och hur dialogen ska ske.

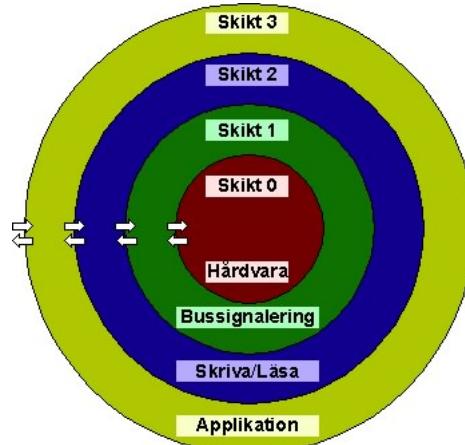
21.3. Tretrådsbussens programvara

Ett vanligt sätt att bygga upp programvara i kommunikationssammanhang är att använda skiktprincipen (lökmodellen). Principen bygger på att inre skikt är oberoende och vet

21. Tretrådsbussen - 3wire

inget om hur de yttre skikten är konstruerade. I vår ansats för att hantera 3-trådsbussen är den inre kärnan hårdvaran i form av DI och DO utgångar i ATmega16/32-processorn och hur dessa används i form av de tre signalerna DQ, CLK och RST-N.

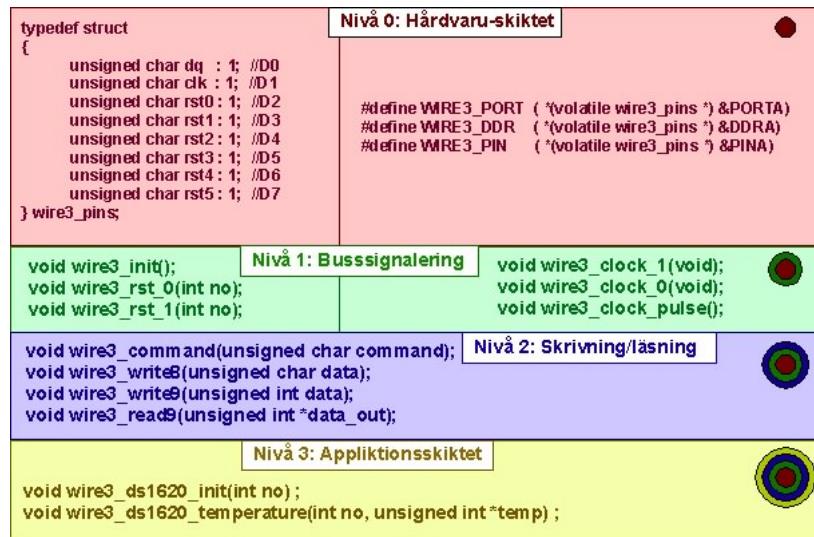
För att förenkla livet för sig då kommunikationsprotokoll skall programmeras brukar "lökskiktsmetaforen" användas. Från den innersta kärnans enkla funktionalitet, byggs programvaran upp med ett skikt i taget och där varje skikt innebär en större funktionalitet och komplexitet. Det fina med denna modell för att bygga programvara är att programmeringen av varje skikt är förhållandevis enkel.



Figur 21.3.1.: Lökskiktsmetaforen

- Skikt 0 - Hårdvaran
 - Beskrivning av hårdvaran med bitfältstruktur och makrokonstanter
- Skikt 1 - Bussignalering
 - Elementär bussignalering implementeras i form av några funktioner,
- Skikt 2 - Elementär skrivning och läsning
 - Bygger vidare så att skrivning och läsning av 8- respektive 9-bitars data kan göras.
- Skikt 3 - Applikationsskiktet
 - Är helt beroende av vilka typer av kretsar som vi ansluter till bussen, i vårt fall så kommer vi att använda kretsen DS1620 som innehåller en temperatursensor och en termostatfunktion. För att hantera DS1620 och dess funktionalitet behöver vi skriva några funktioner. Fördelen med att skriva programvara på detta sätt är att det blir någorlunda lätt att implementera även funktionalitet som är komplex och risken att implementera felaktigheter minskar.

21. Tretrådsbussen - 3wire



Figur 21.3.2.: 3-trådsbussen i vår kontext

21.3.1. Skikt 0 - Inkluderingsfil wire3.h

```
typedef struct
{
    unsigned char dq : 1; //D0
    unsigned char clk : 1; //D1
    unsigned char rst0 : 1; //D2
    unsigned char rst1 : 1; //D3
    unsigned char rst2 : 1; //D4
    unsigned char rst3 : 1; //D5
    unsigned char rst4 : 1; //D6
    unsigned char rst5 : 1; //D7

} wire3_pins;

//ÄNDRA PORT FÖR BUSSEN HÄR
#define WIRE3_PORT ( *(volatile wire3_pins *) &PORTF)
#define WIRE3_DDR ( *(volatile wire3_pins *) &DDRF)
#define WIRE3_PIN ( *(volatile wire3_pins *) &PINF)
```

21.3.2. Skikt 1 - Källkodsfil wire3.c

```
void wire3_init(int n_RST)
{
    WIRE3_DDR.dq = 1;
    WIRE3_DDR.clk = 1;

    if (n_RST >= 1) WIRE3_DDR.rst0 = 1;
```

21. Tretrådsbussen - 3wire

```
if ( n_RST >= 2 ) WIRE3_DDR.rst1=1;
if ( n_RST >= 3 ) WIRE3_DDR.rst2=1;
if ( n_RST >= 4 ) WIRE3_DDR.rst3=1;
if ( n_RST >= 5 ) WIRE3_DDR.rst4=1;
if ( n_RST >= 6 ) WIRE3_DDR.rst5=1;

WIRE3_PORT.dq =0;
WIRE3_PORT.clk =0;

if ( n_RST >= 1 ) WIRE3_PORT.rst0=0;
if ( n_RST >= 2 ) WIRE3_PORT.rst1=0;
if ( n_RST >= 3 ) WIRE3_PORT.rst2=0;
if ( n_RST >= 4 ) WIRE3_PORT.rst3=0;
if ( n_RST >= 5 ) WIRE3_PORT.rst4=0;
if ( n_RST >= 6 ) WIRE3_PORT.rst5=0;

}

/***
 * RST signal is set to 0.
 */
void wire3_rst_0(int no)
{
    switch ( no )
    {
        case 0: WIRE3_PORT.rst0=0; break;
        case 1: WIRE3_PORT.rst1=0; break;
        case 2: WIRE3_PORT.rst2=0; break;
        case 3: WIRE3_PORT.rst3=0; break;
        case 4: WIRE3_PORT.rst4=0; break;
        case 5: WIRE3_PORT.rst5=0; break;
    }
}

/***
 * RST signal is set to 1
 */
void wire3_rst_1(int no)
{
    wire3_clock_0(); //DS1302 needs this
    _delay_us(5);
    switch ( no )
    {
        case 0: WIRE3_PORT.rst0=1; break;
        case 1: WIRE3_PORT.rst1=1; break;
        case 2: WIRE3_PORT.rst2=1; break;
        case 3: WIRE3_PORT.rst3=1; break;
        case 4: WIRE3_PORT.rst4=1; break;
        case 5: WIRE3_PORT.rst5=1; break;
    }
}
```

21. Treträdsbussen - 3wire

```
/**  
 * Clock pulse generation.  
 */  
void wire3_clock_pulse()  
{  
    WIRE3_PORT.clk = 0;  
    _delay_us(1);  
    WIRE3_PORT.clk = 1;  
    _delay_us(1);  
}  
  
/**  
 * Clock set to 1  
 */  
void wire3_clock_1(void)  
{  
    WIRE3_PORT.clk = 1;  
    _delay_us(1);  
}  
  
/**  
 * Clock set to 0  
 */  
void wire3_clock_0(void)  
{  
    WIRE3_PORT.clk = 0;  
    _delay_us(1);  
}
```

21.3.3. Skikt 2 - Källkodsfil wire3.c

```
/**  
 * Write command  
 */  
void wire3_command(unsigned char command)  
{  
    int i;  
  
    WIRE3_DDR.dq=1;  
  
    for ( i=0; i<8; i++)  
    {  
        WIRE3_PORT.dq = ((command & 1) != 0);  
        command = command >> 1;  
        wire3_clock_pulse();  
    }  
}  
  
/**  
 * Write 8 bits data  
 */
```

21. Tretrådsbussen - 3wire

```
void wire3_write8(unsigned char data)
{
    int i;

    WIRE3_DDR.dq=1;

    for ( i=0; i<8; i++)
    {
        WIRE3_PORT.dq = ((data & 1) != 0);
        data = data >> 1;
        wire3_clock_pulse();
    }
}

/***
 * Write 9 bits data
 */
void wire3_write9(unsigned int data)
{
    int i;

    WIRE3_DDR.dq=1;

    for ( i=0; i<9; i++)
    {
        WIRE3_PORT.dq = ((data & 1) != 0);
        data = data >> 1;
        wire3_clock_pulse();
    }
}

/***
 * Read 9 bits data
 */
void wire3_read9(unsigned int *data_out)
{
    unsigned int data=0;
    int i;

    WIRE3_DDR.dq=0;
    //WIRE3_PORT.dq=0; //Enable pull up

    for ( i=0; i<9; i++)
    {
        wire3_clock_0();
        data = data >> 1;
        if ( WIRE3_PIN.dq )
            data = data | 0b100000000;
        wire3_clock_1();
    }
}
```

21. Tretrådsbussen - 3wire

```
*data_out = data;
}

/***
 * Read 8 data bits
 */
unsigned char wire3_read8(void)
{
    unsigned int data=0;
    int i;

    WIRE3_DDR.dq=0;
    //WIRE3_PORT.dq=0; //Enable pull up

    for ( i=0; i<8; i++)
    {
        wire3_clock_0();
        data = data >> 1;
        if ( WIRE3_PIN.dq )
            data = data | 0b10000000;
        wire3_clock_1();
    }

    return data;
}
```

21.3.4. Skikt 3 - Applikationsskiktet

Detta skikt är olika beroende av vilken typ av krets du kopplar in. Vi har två typer av kretsar temperatursensorn DS1620 och realtidsklockan DS1302.

21.4. Uppgifter

21.4.1. Avläsning av temperatursensorn i DS1620

Problemformulering

Skriv ett program som kan läsa av temperaturen från en temperatursensor av typen DS1620 och presentera denna på något lämpligt sätt.

Skriv två funktioner för att hantera kretsen DS1620 som har det seriella gränssnittet 3-wire. Den första funktionen skall initiera kretsen får du given i det som följer:

```
void wire3_ds1620_init(int no)
{
    wire3_RST_1(no);      //FRÅGA 1 VILKEN PINNE PÅ ATmega16 resp. DS1620
    //påverkas?
    //          Enable communication,
    wire3_write8(0x0C); //FRÅGA 2: VILKET KOMMANDO SKICKAS???
    wire3_write8(0x0A); //FRÅGA 3: VILKET REGISTER TAR EMOT VÄRDE??? VAD
    //BETYDER DET???
    wire3_RST_0(no);    //          Disable communication
    wire3_RST_1(no);    //          Enable communication
```

21. Tretrådbussen - 3wire

```
wire3_write8(0xEE); //FRÅGA 4: VILKET KOMMANDO SKICKAS???
wire3_RST_0(no); // Disable communication
}
```

Gå igenom databladet för kretsens DS1620 samt koden för wire3-funktionerna och besvara ovanstående frågor.

Den andra funktionen som du själv skall skriva skall läsa av temperaturen som sensorn ger. Funktionsprototypen är:

```
void wire_DS1620_read_temperature( int number, int *temp );
```

Börja med att koppla upp en DS1620-krets, när du fått denna att fungera kopplar du en till krets av denna typ. Visa temperaturen från den ena sensorn på rad 1 och den andra sensorn på rad 2 på LCD-displayen.

Startprojekt

Följande zippade katalog innehåller en startmapp för att snabbt komma igång med uppgiften:

- avr_ds1620.zip

21.4.2. Realtidsklockan DS1302

Börja med att koppla in kretsen DS1302 på kopplingsdäcket tillsammans med de två temperatursensorerna, se vidare databladet Datablad DS1302.

Koppla Vcc1 till +5V och jorda Vcc2

Några funktioner skall skrivas för att hantera en realtidsklocka. En funktion med namnet ds1302_read skall kunna avläsa år, månad, dag, timmar och minuter, den andra funktionen ds1302_write skall skriva till kretsen för att sätta tiden i denna. Funktionsprototyperna bör vara på formen:

```
void ds1302_init( ds1302 *this, int no); //no=RST-N signal, 0-5, spara info om no i strukturen
```

Vilket värde skall WP-biten i CONTROL-registret ha?

```
void ds1302_write( ds1302 *this);
```

Vilket värde skall CH-biten i SEC-registret ha?

```
void ds1302_read(ds1302 *this);
```

Observera att funktionerna bör använda sig av skikt 1-2 funktionerna i "wire3.h" respektive "wire3.c".

Modularisera programmet med hjälp av källkodsfiler, samla realtidsklockefunktionerna i en källkodsfil med namnet "ds1302.c" respektive en inkluderingsfil med namnet "ds1302.h". För att hantera datat till och från kretsens bör du definiera en struktur enligt följande:

```
typedef struct
{
    unsigned char year;
    unsigned char month;
```

21. Tretrådsbussen - 3wire

```
unsigned char date;
unsigned char day;
unsigned char hour;
unsigned char min;
unsigned char sec;
unsigned char no; //RST-N nummer
} ds1302;
```

Komplettera programmet i föregående uppgift så att du presenterar aktuell tid (HH:MM:SS) med avläst temperatur på LCD-displayen. Fundera ut en lösning på hur du skall kunna ställa klockan.

22. RS232

Innehåll

22.1 Att läsa

159

22.1. Att läsa

- How To Work With C# Serial Port Communication
- Slides
 - RS232 slides
- Prolific-driver för USB-till Seriell-omvandlare
 - Konverteraren som vi använder är kina-tillverkad och kräver en äldre variant av Profilic-drivrutinerna för att fungera, ladda ner och packa upp bifogade fil:
 - ◊  Drivrutiner Profilic från 2008
 - Anslutning till STK500
 - ◊ Svart - GND
 - ◊ Röd - VDD
 - ◊ Vit - PD1 (TXD)
 - ◊ Grön - PDO (RXD)
 - Enhetshanteraren startas
 1. Öppna ”Portar (COM och LPT)”
 2. Öppna kontextmenyn för drivrutinen ”Prolific USB-to-Serial Portability”
 3. Välj menyalternativet ”Uppdatera drivrutin ...”
 4. I fönstret väljer du ”Välj drivrutsprogramvara som redan finns på dator”
 5. I det nya fönstret väljer du
 - a) Bläddra fram till katalogen där du har packat upp drivrutinskatalogen.
 - b) ”Låt mig välja från en lista över drivrutiner som finns på datorn”
 6. Välj här Profilic från 2008 och ej 2015

Part VII.

Programvaror, datafiler och dokument

Innehåll

23 Programvaror som används i kurserna	162
23.1 Adobe Reader	162
23.2 MSDNAA	162
24 Utvecklingsmiljön och program tillhörande denna	164
24.1 Atmel Studio7	164
24.2 Terminalprogram	164
24.3 Startprojekt och exempelprogram	165
25 Böcker, kompendier och artiklar	167
25.1 Kompendier	167
25.2 AVR manualer, m.m.	168
25.3 Slides	168
25.4 Videos	169
26 Sensorer och andra typer av kretsar	170
26.1 Sensorer	170
26.2 3wire-kretsar	170
26.3 PS 2 tangentbordsprotokoll	170
27 Intel assembler	171

23. Programvaror som används i kurserna

Innehåll

23.1 Adobe Reader	162
23.2 MSDNAA	162

23.1. Adobe Reader

Dokumentet du löser nu skall helst öppnas med hjälp av Adobe Reader för att fullt ut utnyttja alla möjligheter som ett pdf-dokument erbjuder. Några inställningar av Adobe Reader som kan vara bra att göra:

- Menyn Adobe Reader
 - Preferences -> Documents
 - ◊ Se till att alternativet “Open cross documents in same Window” ej är förbockat. Originalfönstret dvs detta dokument finns då alltid öppet i ett eget fönster.
- Videofilmer i pdf-dokumentet
 - Ladda Adobe Flash Player och installera denna
 - ◊ <http://www.adobe.com/support/flashplayer/downloads.html>
- zip-, jar-, exe-filer
 - På grund av säkerhets restriktioner i Adobe Reader så slutar alla zippade filer med ändelserna .zip.txt, där txt har lagts till för att vi skall kunna spara ner filen. När du sparar ner filen döper du om den genom att ta bort .txt i filnamnet. Förutom zip-filer gäller detta analogt för alla typer av exekverbara filer.

23.2. MSDNAA

MSDNAA är ett speciellt avtal/erbjudande som låter studenter och anställda vid Örebro universitet prova nästan alla Microsoftmjukvaror gratis. Förutsättningen är att de

23. Programvaror som används i kurserna

använts för icke kommersiellt bruk eller i testsyfte. Microsoft Office och spel är inte inkluderat i detta avtal, men nästan all annan mjukvara inklusive operativsystem ingår. Studenter har bara möjlighet att ladda ned mjukvara via webbsidan under den tid de aktivt studerar, men mjukvaran behöver inte avinstalleras efter att studierna avslutats. För att få tillgång till nedladdningssidan måste du först skicka ett e-post meddelande till MSDNAA@oru.se. Meddelandet måste innehålla ditt användarid om du skickar meddelandet från en emailadress som inte slutar med student.oru.se eller studentmail.oru.se. Du kommer då att få ett e-post meddelande med kontouppgifter direkt från Microsoft. Mer information finns att läsa på Microsofts hemsida: [Microsoft DreamSpark](#).

24. Utvecklingsmiljön och program tillhörande denna

Innehåll

24.1 Atmel Studio7	164
24.2 Terminalprogram	164
24.3 Startprojekt och exempelprogram	165

24.1. Atmel Studio7

Programvarans som används primärt i denna kurs är AtmelStudio7 som är en integrerad utvecklingsmiljö och innehåller både kompilator, assembler och en simulator. Programvaran kan hämtas hem från Atmel. Utvecklingsmiljön baserar sig på samma IDE (Integrated Development Environment) som finns i Visual Studio.

Nerladdning av AtmelStudio7 görs från www.atmel.com Ovanstående program finns installerade på skolans datorer.

24.2. Terminalprogram

För att kunna kommunicera med ditt program via utskrifter med printf-satser och inläsning av data från tangentbordet via scanf-satser i ditt program, behöver du använda ett terminalprogram.

- TeraTerm Pro
 - Mycket bra terminalprogram som emulerar olika terminalstandarder , t. ex. VT100
 - Nerladdning från: <http://logmett.com/index.php?/download/tera-term-485-freeware.html>
- Terminal
 - Terminal är ett minimalt terminalprogram med många bra egenskaper, speciellt lämpad för felsökningsändamål i små inbyggnadssystem med seriell kommunikation.
 - Nerladdning från: <https://sites.google.com/site/terminalbpp/>

24.3. Startprojekt och exempelprogram

- Startmappar för projekt
 -  Hjälpfiler i form av C++-klasser - 160110
 - ◊ Klassen ai - Analoga Insignaler
 - ◊ Klassen di - Digitala Insignaler
 - ◊ Klassen do - Digitala Utsignaler
 - ◊ Klassen lcd4 - LCD-display med 4 bitars datagränssnitt (plus 2 styrsignaler)
 - ◊ Klassen usart - Seriell kommunikation med pollning.
 - ◊ Klassen usart_isr - Seriell kommunikation med pollning.
 - ◊ Klassen time - Timer
 - ◊ void init_stdio_stdout (void) - Funktions som anropas i CStartup, dvs innan vi kommer till main-funktionen, för att initiera att printf och scanf fungerar mot en USART.
 -  Hjälpfiler i C-kod - 150204
 - ◊ Hjälpfiler som går att använda i C++-program.
 - ▷ Hjälpfilerna är till stor del skrivna i ObjektOrienterad C-stil, första argumentet är en pekare till en struktur. Denna pekare hade namnet this tidigare, nu har den fått namnet obj för att inte få kompileringsfel då this är ett nyckelord i C++ (ej i C).
 - ▷ Om du låter en .c ingå i ett c++-projekt skall du byta namn på denna fil till .cpp (lcd4.c blir lcd4.cpp).
 -  Startkataloger för olika projekttyper - 150104
 - ◊ Atmel Studio 6 Version 6.2.9200
- Exempelprogram
 -  Exempelprogram - Datorteknikkursen - 131220
 1. Projekt StartStopTime - Tidtagarur programmerat i assembler
 2. Projekt MinMax - Parameteröverföring, hur görs detta på maskinnivån.
 3. Projekt Recursive - Rekursiva funktioner, hur görs detta på maskinnivån
 4. Projekt WhatHappens - Vad händer på maskinvån vid exekvering av ett C-program
 5. Projekt CPP_ISR_RING_COUNTER - C++-klassen ring_counter och en avbrottsfunktion

24. Utvecklingsmiljön och program tillhörande denna

-  Exempelprogram - Projekt husautomation - 131220
 1. Project static_execution_scheme - Statiskt exekveringsschema
 2. Projekt avr_sio_isr - Seriell kommunikation med avbrottssfunktioner för att undvika vänteloopar vid sändning och mottagning av tecken
 - a) EEPROM-kommandon för att skriva och läsa i ett EEPROM-minne med SPI-bussinterface.
 3. Projekt avr_ds1620 - Temperatursensorkretsen DS1620 med 3wire-bussgränssnitt
-  Exempelprogram - Seriell kommunikation via en COM-port med Csharp 131220

25. Böcker, kompendier och artiklar

Innehåll

25.1 Kompendier	167
25.2 AVR manualer, m.m.	168
25.3 Slides	168
25.4 Videos	169

25.1. Kompendier

- [AVR RISC-processorer](#)
 - Olika modeller av en dator.
 - Talsystem, binära koder och binär aritmetik.
 - Repetition av grundläggande digitalteknik.
 - Programmerarens modell av processorn.
 - GNU assembler/länkare.
 - Adresseringstekniker.
 - Logiska och aritmetiska beräkningar på maskinnivå.
 - Villkorliga och repetitiva strukturer på maskinnivå
 - Modulär programmering med subrutiner.
 - I/O-programmering på assemblernivå.
- [Objektorienterad modellering](#)
 - Hårdvarunära programmering i C
 - Modellering av periferiheter
 - Avbrott och avbrottsfunktioner
 - Objektorienterad modellering av ett datorsystem
 - Problemlösning med hjälp av taskabstraktionen
 - Exekvering av periodiska tasks i en vänteloop.

25.2. AVR manualer, m.m.

- Tekniska referensmanualer
 - Referensmanual ATmega16
 - Referensmanual ATmega32
 - Manual för datorkortet STK500
- Assemblerprogrammering
 - Detaljerad Instruktionsmanual
 - ◊ Detaljerad beskrivning av instruktionerna till AVR-processorn Under assemblerprogrammeringsdelen av kursen har du stor nytta av denna manual som beskriver alla processorns instruktioner på ett detaljerat sätt.
 - AVR Inline assembler cookbook
 - ◊ Beskriver på ett bra sätt hur inline assembler fungerar i GNU C komplatorn.
- C-programmering
 - Effektiv C-kodning för AVR-processorer

25.3. Slides

- Datorteknik
 - OH-serie för campuskurserna Datorteknik
- Mekatroniska system
 - Ger en överblick över tillämpning av mikroprocessorer i inbyggda system såsom: bilen, hemautomation, gräsklippare, etc.
- Effektiv C-kodning
- Realtidsprogrammering
 - Taskabstraktionen
 - Statiska exekveringsscheman
- Kommunikationsbussar
 - 3wire-bussen

25.4. Videos

- Microcontroller Tutorial - A Beginners Guide
 - Spellista med 48 videofilmer av Patrick Hood-Daniel rörande ATmega32-processor och utvecklingsmiljön kring denna.

26. Sensorer och andra typer av kretsar

Innehåll

26.1 Sensorer	170
26.2 3wire-kretsar	170
26.3 PS 2 tangentbordsprotokoll	170

26.1. Sensorer

- Temperatursensorn SMT160
 - SMart Temperature sensor 160

26.2. 3wire-kretsar

- DS1620 - Temperatursensor och termostatreglering
 - Datablad
 - Applying and Using the DS1620 in Temperature Control Applications
 - Temperature Sensor Eludes Analog Interfacing
- DS1302 - Realtidsklocka
 - Datablad

26.3. PS 2 tangentbordsprotokoll

- PS 2 Mouse / Keyboard Protocol

27. Intel assembler

- Assembly Language Succinctly