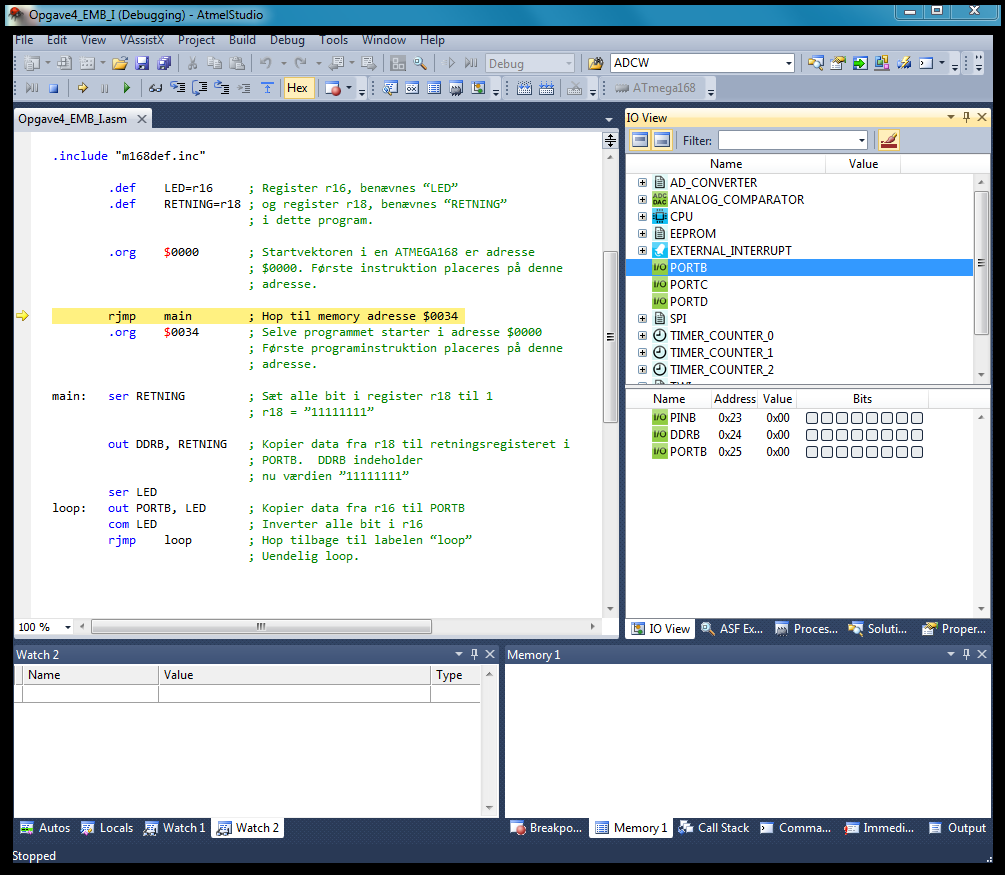
**Embedded programmering**

**med Atmega168**

****

**Praksisorienterede cases**

**Til brug i faget Embedded I controller**

Udarbejdet af Lars Thise Pedersen (December 2017):  
Redigeret af Lærke Brandhøj Kristensen (April 2020):

Indhold

[Embedded programmering 1](#_Toc70239253)

[Beskrivelse af læringsaktiviteten 5](#_Toc70239254)

[Formål 5](#_Toc70239255)

[Læringselementet 5](#_Toc70239256)

[Begreber i Assemblerprogrammering 5](#_Toc70239257)

[Udstyrsliste 6](#_Toc70239258)

[Formler 6](#_Toc70239259)

[Ordliste 6](#_Toc70239260)

[Facitliste 6](#_Toc70239261)

[Opgave 1: Bliv fortrolig med Atmel Studio & ATmega168 8](#_Toc70239262)

[Opgave 2: AVR-hardware 9](#_Toc70239263)

[Opgave 3: Det første program, tænd en eller flere LED’s 12](#_Toc70239264)

[Alt kode skal gemmes i samme solutionen med forskellige projekter for hver opgave. 12](#_Toc70239265)

[12](#_Toc70239266)

[a. Du skal nu til at anvende Atmel Studio til at skrive et assembler program, som skal indeholde følgene instruktioner: 12](#_Toc70239267)

[ #define – bruges til at give en værdi et alias. 12](#_Toc70239268)

[ Shift operation – til at sætte værdien af registret (PORTB ^= (1 << LED5)). 12](#_Toc70239269)

[ Kommenter hver linje af dit program. 12](#_Toc70239270)

[Opgave 4: Tænd og sluk en eller flere LED’s 13](#_Toc70239271)

[Opgave 5: Forsinkelser / delay i software 14](#_Toc70239272)

[Opgave 6: Blinkende LED’s 16](#_Toc70239273)

[Opgave 7: Tænd én LED, v.h.a. en knap 17](#_Toc70239274)

[Opgave 8: Data ind & ud af AVR 18](#_Toc70239275)

[Opgave 9: Tilslut ekstra lysdiode 19](#_Toc70239276)

[Opgave 10: Udvikling af software 20](#_Toc70239277)

[Opgave 11: RS232 kommunikation 21](#_Toc70239278)

[Organisationsform 23](#_Toc70239279)

[Krav til produktet 23](#_Toc70239280)

[Dokumentation 23](#_Toc70239281)

[Elevforudsætninger 23](#_Toc70239282)

[Evaluering 23](#_Toc70239283)

[Vejledende tid 24](#_Toc70239284)

[Delmålpinde 24](#_Toc70239285)

[Relevante henvisninger til supplerende materiale 24](#_Toc70239286)

[Diagrammer 25](#_Toc70239287)

# Beskrivelse af læringsaktiviteten

I denne læringsaktivitet vil du komme til at arbejde med programmering i sproget C. Du kommer til at anvende instruktions sættet til en microcontroller fra firmaet Atmel, den hedder ”Atmega168”. Da hver MCU har sit eget instruktions sæt, er det nødvendigt at kende dette, for at kunne programmere i assembler. Det vil dog vise sig, at mange elementer ligger tæt op af hinanden, fra MCU til MCU, nogle elementer er også ens. Derfor vil du også kunne overføre viden fra arbejdet med dette læringselement, til arbejdet med andre typer processorer.

Der lægges meget stor vægt på at være praksisorienteret, d.v.s. at du skal kunne udføre hele processen, fra planlægning af programmet, til at kunne teste og debugge den færdige microcomputer-løsning. Alle eksempler kan afprøves, og fungerer på Freeduino v1.16

# Formål

Efter læringselementet er du i stand til:

* At planlægge et struktureret programmeringsforløb
* At programmere mindre programmer i C
* At simulere software, v.h.a. simulator på PC, samt at debugge v.h.a. PC
* At brænde/programmere en MCU
* At afprøve en embedded microcomputer løsning
* At udføre begyndende fejlfinding på software

# Læringselementet

Her kommer en vejledning til, hvordan du lettest arbejder med dette læringselement.

Læsestoffet er opdelt efter nedenstående oversigt:

Alle opgaver, øvelser og eksempler er inddelt efter sværhedsgrad og mængden af viden der skal arbejdes med. Derfor er det hensigten at rækkefølgen i læringselementet skal følges.

Det forudsættes, at du har lært dig en vis viden om hardware, i en MCU.

# Begreber i Embedded programmering

Struktureringsteknikker, til brug ved software-udvikling  
Opbygning af assembler  
Et såkaldt ”IDE” (Integrated development environment).  
Simulering af software, v.h.a. PC  
Debugging af software, v.h.a. PC

# Udstyrsliste

Et AVR-board, af typen Freeduino, som byggesæt  
AVRISP mkII programmeringsmodul eller en USBASP.  
En PC, med sidste nye version af Atmel Studio (kan downloades på deres hjemmesider)  
Et Breadbord

LED’er, modstande, en switch knap, en Lux sensor, en variable modstand og ledninger.

# Formler

De eneste formler du får brug for, er til beregning af delay-tider. Disse formler er beskrevet under de enkelte opgaver.

# Ordliste

Assembler: Et ”lav-niveau-sprog”, hvori der programmeres software til processorer (MPU, CPU & MCU). Er opbygget af forkortelser, såkaldte ”Mnemonics”.  
Lav-niveau-sprog: Et sprog der ligger langt fra vores daglige tale. Der programmeres meget ”tæt på hardware”.  
  
MPU: Micro-processing-unit (meget som en ”CPU”, men mindre).  
  
CPU: Central-processing-unit.  
  
MCU: Micro-controlling-unit (kaldes også for en ”Single-chip”).  
  
IDE: Integrated development environment. Et komplet miljø til at programmere, assemble/compile, simulere og debugge software i, på PC. (Atmel Studio er her vores ”IDE”).  
Atmel Studio: Se ”IDE”.  
  
Freeduino: Navnet på det print vi bruger til test og afprøvning af software, et ”evaluation board”. Kan også kaldes ”AVR-board”.

Mnemonics: Kommer fra memorya id, de er forkortelser som er nemmere at huske end de fulde navne. Et mnemonics består af forbogstaverne af alle ordende i navnet. Eksemple: data-directive register = DDR.

I dette dokument referer det til det korte navn for registre.

# Facitliste

Der er ikke nogen egentlig facitliste, til denne læringsaktivitet.   
Det der er vigtigt er, at du tilegner dig den viden, der skal til, for at kunne udføre de delmålpinde, der er beskrevet under punkt 8, i dette sæt.  
Desuden er der flere måder at løse en programmeringsopgave på. Der kan således være flere rigtige løsninger og svar, på et spørgsmål, eller en opgave.

**Øvelser og opgaver**

Softwaren i opgaverne er selvforklarende, der er mange kommentarer. Men hav alligevel dokumentation over Atmel Studio & instruktionssæt ved hånden.

Igennem læringsaktiviteten skal du nogle gange afprøve i praksis, og andre gange simulere i Atmel Studio.

Du skal selvfølgelig kunne programmere AVR i praksis, men simulatoren kan anvendes til at kunne se de elementer AVR (og en MCU) indeholder. Derfor er simulatoren god, til at understøtte teorien. F.eks. kan man hverken se stack-pointer (SP) eller program counter (PC), på selve AVR, men det kan man i simulatoren, og dermed få en bedre forståelse for, hvordan en MCU arbejder.

Opgave sættet skal afleveres individuelt, men i er velkommen til at hjælpe hinanden så længe i ikke skriver af og aflevere de samme vidoer.

# Opgave 1: Bliv fortrolig med Atmel Studio & ATmega168

Formålet med denne opgave er, at få en grundlæggende viden, om Atmel Studio og ATmega168. Dette er forudsætningen, for overhovedet at kunne anvende dem.

* Hvis Atmel Studio ikke allerede er installeret:  
  Download & installer Atmel Studio, i nyeste version. (**Version 7.xx**) Dette gøres, på nuværende tidspunkt, på adressen: http://www.atmel.com/microsite/atmel-studio/.
* I Moodle læringsrummet findes datablad over ATMega168 (ATMEGA Datasheet.pdf).

Man kan meget let blive ret overvældet over den mængde information, man bliver mødt med i det fulde datablad over ATMega168 processoren. Men man opnår hurtigt en erfaring I, hvordan man bedst læser et datablad på så mange sider. Man finder lige nøjagtig det, man er interesseret i at vide noget om og koncentrer sig om dette område. Og så kan man også sagtens finde application notes og lignende på nettet, der beskriver den del man arbejder med lige nu.

**Diskuter parvis de ”Features” MCU har.**

**Hvilke muligheder har MCU?**

**Læg mærke til, at her er de omtalte elementer beskrevet, såsom ALU, stack, program- & datahukommelse.**

* For at kunne analysere og programmere software til embedded er det nødvendigt at have et godt oveblik over de instruktioner, der er tilgængelige på den processor man arbejder med. I Moodle læringsrummet findes der 2 dokumenter, der beskriver Assembler instruktionerne for ATMega processorer. Disse er:

1. ATMega Datasheet (Section 31. Instruction Set Summary)
2. Assembler instruktioner til ATMega Processorer
3. AVR Assembler instruktioner (i listeform)

**Hvilke 5 hovedgrupper er instruktionssættet inddelt i?  
Hvad er den overordnede funktion, i de 5 grupper? (Hvad anvendes de til?)**

**Opgave 2: AVR-hardware**

Denne opgave består af en række mindre opgaver og spørgsmål. Formålet er, at danne et grundlag for, at arbejde med hardwaren i ATmega168.

**I/O map for ATmega168 (Pin Mapping)**

For at aktivere de 3 Input/Output porte, skal man kende deres fysiske adresser i I/O adresseområdet

Udfyld nedenstående skema for de 3 porte, ved hjælp af det udleverede materiale, eller ved søgning på internettet på [www.atmel.com](http://www.atmel.com)

Se evt. det fulde datablad, side 343 – 346.

Der benyttes 3 port adresser til hver port.

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Mnemonic** | **HEX Adresse** | **Funktion** |
| B | PORTB | 0x25 | Tager input til B |
| B | DDRB | 0x24 | Husker på det der ligger i B |
| B | PINB | 0x23 | Giver output fra B |
| C | PORTC | 0x28 | Tager input til C |
| C | DDRC | 0x27 | Husker på det der ligger i C |
| C | PORTC | 0x26 | Giver output fra C |
| D | PORTD | 0x2B | Tager input til D |
| D | DDRD | 0x2A | Husker på det der ligger i D |
| D | PORTD | 0x29 | Giver output fra D |

På tilsvarende måde, skal du finde adressen på ***Status Register***, og give en kort beskrivelse af funktionen på de enkelte bit.

***Status Register*** er en del af AVR CPU Core funktonalliteter.

|  |  |
| --- | --- |
| **Status reg. adresse** | **Mnemonic** |
| 0x3F | SREG |

Beskriv funktionen af hver enkelt bit i statusregisteret i tabellen.

|  |  |  |
| --- | --- | --- |
| **Bit** | **Bogstavs -betegnelse** | **Funktion** |
| 7 | I | Global Interrupts Enable: Brugt til at håndtere interrupts, hvis interrupts er slået til |
| 6 | T | Bit Copy Storage: Bliver brugt til at kopiere bits fra **b**it **l**oa**d** eller **b**it **st**ore |
| 5 | H | Half Carry Flag: Kun halvdelen bliver brugt |
| 4 | S | Sign Bit: Imellem 2 & 3 |
| 3 | V | Two’s Complement Overflow flag: |
| 2 | N | Negative Flag: Minustal |
| 1 | Z | Zero Flag: Wannabe null |
| 0 | C | Carry Flag: Det hele bliver brugt |

**X-tal**

Til hvilke ben på processoren (PDIP) skal krystallet tilsluttes?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hvad er den højeste arbejdsfrekvens for ATmega168?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Processoren har en indbygget funktion til seriel kommunikation (USART).

Register beskrivelse for USART findes på side 190 – 198.

**Hvilke I/O adresser benyttes til seriel kommunikation.**

|  |  |  |
| --- | --- | --- |
| **Register (HEX)** | **Mnemonic** | **Funktion** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Timer/Counter**

Hvor mange timere/countere er der indbygget i MCU’en. Hvordan benævnes de?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Memory map**

Processoren har 3 forskellige typer hukommelse.

Programhukommelse som udgøres af Flash hukommelse. Det er i denne hukommelse, de programmer man udvikler placeres, når man downloader dem til ATMega168 processoren.

SRAM, som benyttes af processoren til opbevaring af flygtige data. SRAM indhold slettes når forsyningsspændingen fjernes.

EEPROM, benyttes til opbevaring af vigtige data, som ikke forsvinder når forsyningsspændingen fjernes. EEPROM data gemmes også i en Flash hukommelse (som program data). **Men det er ikke samme Flash hukommelse, der benyttes til program data og EEPROM data**.

**Find den laveste og højeste adresse for Program hukommelsen (Program Flash)**

Laveste adresse

Højeste adresse

Hvilket format har denne hukommelse (8,16,32 bit)

Hvor stor er programhukommelsen (KB/Byte)

**Find den laveste og højeste adresse for intern RAM Data hukommelsen (SRAM)**

Laveste adresse

Højeste adresse

Hvilket format har denne hukommelse (8,16,32 bit)

Hvor stor er data hukommelsen (KB/Byte)

**Find den laveste og højeste adresse for EEPROM Data hukommelsen (EEPROM)**

Laveste adresse

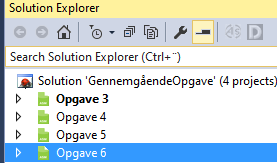
Højeste adresse

Hvilket format har denne hukommelse (8,16,32 bit)

Hvor stor er data hukommelsen (KB/Byte)

# Opgave 3: Det Assempler første program, tænd en eller flere LED’s

Alt kode skal gemmes i samme solutionen med forskellige projekter for hver opgave.



1. Du skal nu til at anvende Atmel Studio til at skrive et assembler program, som skal indeholde følgene instruktioner:

* .def – bruges til at give en værdi et alias.
* Assempler code der tænder for digital pin 13.
* Kommenter hver linje af dit program.
* Husk også og inkluder .include "m168def.inc" .

Programmér AVR, med AVRISP mkII eller USBASP. Efter flash programmering skal du se digital pin 13 lyse op.

1. Simuler programmet i Atmel Studio. Læg især mærke til, hvordan portene håndteres i simulatoren.

* Tag et screenshot der viser med simulatoren i Atmel studio at simulatoren viser at alle pins på PORTB er tændte, inkluder screenshot som billede i afleveringen.

# Opgave 4: Tænd og sluk en eller flere LED’s

I den følgende opgave, skal vi have alle LED’s koblet til Port B til at blinke. Da instruktionerne afvikles så hurtigt, at det menneskelige øje ikke kan følge med, skal du afprøve programmet i simulator debuggeren, for at se resultatet og derefter koble LED’er op Port B.

* Byg dit egent program i C der blinker Port B uden delay.
* Brug simulatoren til at debugge og tjekke din kode, før du kobler LED’er til.
* Inkluder screenshot af simulatoren.

De næste opgaver er i rækkefølge og skal løses trinvis.

# Opgave 5: Forsinkelser / delay i software

Når man laver software, har man nogle gange brug for forsinkelser, f.eks. når der skal gå et vis stykke tid, inden en port aflæses, eller hvis en enhed på en bus er langsommere, end MCU. Forsinkelser kan opnås på flere måder. Én af dem er, at programmere et delay, v.h.a. software.

1. **Byg videre på opgave 4 således at PORTB blinker i en hastighed der kan ses fysisk.**
   * **Husk at skrive kommentar til din egen kode.**
2. **Afprøv programmet nedenfor, og dokumenter alle kodelinjerne.**

* Brug datasheet til at slå om hvad registrene gør.
* Skriv ned hvilke fysiske pins bliver påvirket i kommentarene.
* Der må gerne sættes #define op så koden er mere læselig.

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 16 bit register and Bit shifts \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Opgave5b.c

\* Created: 13-10-2020 12:49:40

\* Author : lkri

\* Created for Atmega168 Datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-9365-Automotive-Microcontrollers-ATmega88-ATmega168\_Datasheet.pdf

\* Optimized for Atmel Studio 7.0

\* May be copied and used with source reference.

\*/

#define *F\_CPU* 16000000UL \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <avr/io.h>

#include <util/delay.h>

#define SPL \_SFR\_IO8 (0x3D) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define SPH \_SFR\_IO8 (0x3E) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define low(x) ((x) & 0xFF) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define high(x) (((x)>>8) & 0xFF) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void shift\_example () {

PORTB |= (1 << 5); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

PORTB &= ~(1 << 5); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

PORTB ^= (1 << 5); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

PORTB >>= 2; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

PORTB <<= 1; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

PORTB <<= 1; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*\_delay\_ms*(1000);

}

int main(void)

{

SPL = low(RAMEND); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SPH = high(RAMEND); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DDRB = 0xFF; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

while (1)

{

shift\_example();

}

}

# Opgave 6: Blinkende LED’s

Med udgangspunkt i opgave 4 og 5, skal du nu lave et nyt projekt, som med en delay funktion får de tilkoblede LED’s på Port B til at blinke så langsomt,

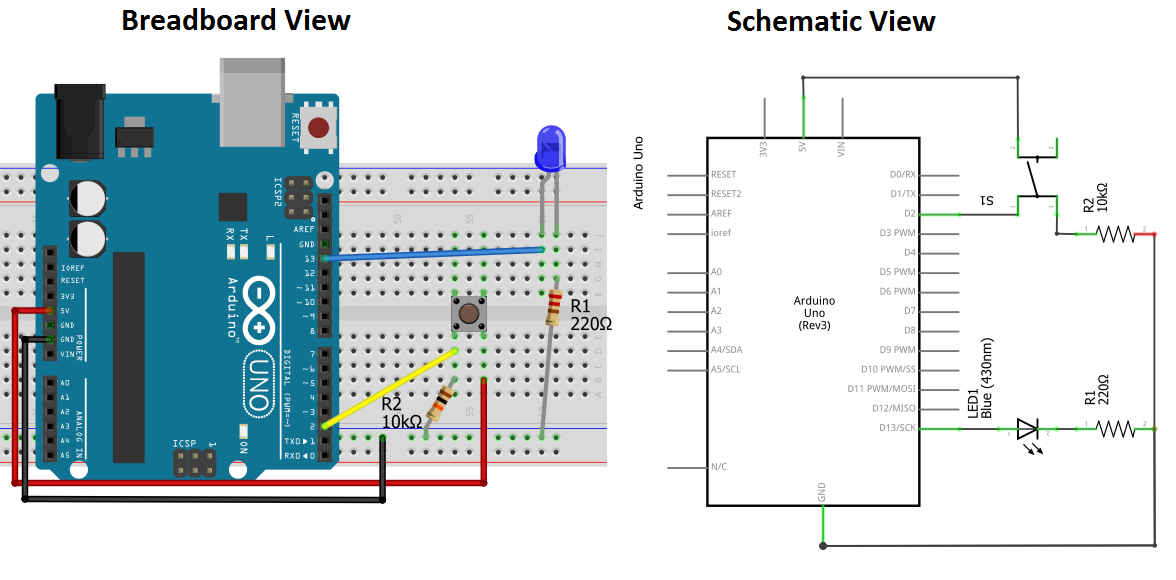
* Tiden, hvor dioderne lyser, skal være dobbelt så lang, som tiden hvor dioderne er slukket.
* Kommenter din kode.
* Brug #define til at navngive de LED’er du bruger.
* Brug Metoder til at dele din kode op.

# Opgave 7: Tænd én LED, v.h.a. en knap

Lav et program i der tænder en LED på digital pin 12 ved at læse inputtet fra en knap.

Læg mærke til, at vi til forskel fra de tidligere opgaver nu kun opererer på én LED.

* HUSK pull-up modstand, som ses på billedet.



# Opgave 8: Data ind & ud af AVR

1. I en MCU er der indbyggede porte, disse er selvfølgelig meget vigtige, da det er gennem dem, at vi kan få data ind og ud af MCU. Det er gennem portene, at MCU kommunikerer.

Uden disse porte kunne vi ikke få meget fornøjelse ud af MCU (vi kunne selvfølgelig montere perifere porte, men så er definitionen af en MCU jo ødelagt; MCU=MPU, porte program- og datahukommelse).

Følg youtube vejledning for mere info: [LINK](https://www.youtube.com/watch?v=w37FOUrXO6s&list=PLtQdQmNK_0DRhBWYZ32BEILOykXLpJ8tP&index=5)

* Lav et program der bruger en knap til at lave et PIN change interrupt.
* Programmet skal lyse indtil der kommer et PIN change interrupt, hvorefter den skal stoppe.
* Dokumenter programmet, ved hjælp af instruktionssættet (Datasheet section 11).

1. Med udgangspunkt i de foregående eksempler, skal du fremstille et program, som

ved hjælp af en trykknap funktion, kan styre 2 lysdioder, således at når knappen aktiveres tænder diode 2 og diode 1 slukker. Når knappen slippes, tænder diode 1 og diode 2 slukkes.

* Test programmet i simulatoren.
* Tilslut 1 ekstra lysdiode til dit print, og afprøv programmet igen.
* Husk at skrive kommentar til din kode.h

# Opgave 9: ADC Analog værdier

Her skal i bruge ADC kanalen til at læse på. I den embedded verden er det svært at arbejde med analog værdier, i ATMega168 er der 10 bit analog værdier, som man kan bruge. Der skal laves lidt setup af 2 registre (ADMUX, ADCSRA) som begge bestemmer nogle forskellige settings, som er relavente for at læse en analog værdi.

* Lav et program der læser en analog værdi fra en port og tænder en LED når den er over en hvis spænding.
* Brug til denne opgave en variable modstand og husk en pull-up modstand ligesom ved knappen.
* Skriv kommentar til hvad registrene gør og skriv gerne sidetal på hvor informationen er fundet i datasheet’et.

Følg youtube vejledning for mere info: [LINK](https://www.youtube.com/watch?v=9w3NpPQMG0I&list=PLtQdQmNK_0DRhBWYZ32BEILOykXLpJ8tP&index=24)

**Opgave 10:** **Udvikling af software**

I skal nu selv udvikle software, hvor du udnytter de strukturer og programmeringsteknikker, du har arbejdet med.

Opgaven skal udarbejdes i grupper, på 2-3 personer.

Når opgaven er udført, fremlægger alle grupper for klassen. Du skal regne med at kunne bruge 15 minutter på fremlæggelsen. På denne måde ses der forskellige måder at arbejde på. Samtidigt giver det øvelse i at strukturere, og fremlægge. Hvis et program skal kunne læses og forstås af andre, kræver det typisk mere struktur, end hvis man selv skal læse det.

Bemærk at jeres program skal virke i praksis.

Herunder er der vist, hvilke krav, der er til softwaren:

* Skal være programmeret i assembler.
* Programmet skal simulere et trafiklys. (Rød/Gul/Grøn)
* Der skal monteres 3 lysdioder.
* Der skal anvendes funktioner og makroer.
* Programmet skal struktureres, således at det med ændring af én define vil virke på andet HW end ATMega168.

Ekstraopgave 1

* Lyset opbygges med 2 retninger. (6 dioder)

Ekstra opgave 2

* Lyset forsynes med højresvings pil. (1 grøn diode)

# Opgave 11: RS232 kommunikation

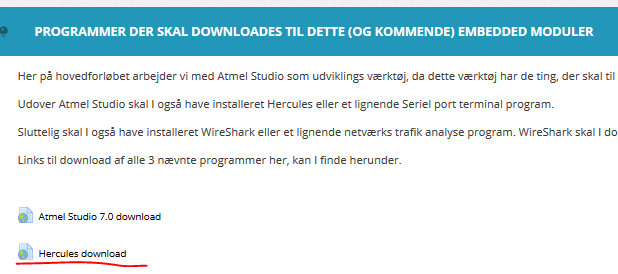
I denne opgave, skal du opsætte Com-porten til 115200 Baud 8 databit, og transmit enable for at kunne lave et program som skal skrive det tilbage som den for ind fra seriel kommunikation med din computer.

h

Følg youtube vejledning for mere info: [LINK](https://www.youtube.com/watch?v=1su5uSRHkpk&list=PLtQdQmNK_0DRhBWYZ32BEILOykXLpJ8tP&index=16)

Registeropsætningen er beskrevet i afsnit 19.10.xx, på siderne 190 – 198

I skal bruge programmet ’Hercules’ som i kan finde på moodle.



* Du skal bruge koden nedenunder til at skrive Seriel værdier imellem din computer og controlleren.

unsigned char cmd USART\_receive() {

/\* replace with your code \*/

return cmd;

}

void USART\_transmit(unsigned char cmd) {

/\* replace with your code \*/

}

void USART\_Init () {

/\* replace with your code \*/

}

int main(void)

{

USART\_Init();

while (1)

{

USART\_transmit(USART\_receive());

}

}

Opgaverne afleveres med software og dokumentation til underviseren, der bedømmes materialet. Der lægges vægt på læsbarhed, struktur og kommentare der reflektere forståelse for koden.

# Organisationsform

Igennem læringsaktiviteten skal du arbejde meget selvstændigt, i grupper på 2-3 personer.

Prøv først at løse en eventuel problemstilling i gruppen, inden underviseren tilkaldes.

Det er vigtigt, at du giver dig og dine kollegaer tid til at arbejde med stoffet, da det er den eneste måde at lære sig det på.

# Krav til produktet

Du skal kunne læse og forstå alle eksempler, i denne læringsaktivitet, samt kunne strukturere et programmeringsforløb selv. Der stilles især krav til dette i opgave 10.

Du skal arbejde selvstændigt og grundigt, hvis du ”jabber” sættet igennem, er der risiko for, at du ikke lærer dig nok og at du senere glemmer det lærte.

# Dokumentation

Det er vigtigt, at du dokumenterer alle dine erfaringer og oplevelser med notater, da du kan bruge dem senere i din uddannelse og i dit arbejde.

At kunne dokumentere er af afgørende betydning for, om et stykke software kan læses og dokumentationen er altafgørende, hvis du senere (evt. flere uger, måneder eller år efter) skal fejlfinde eller opgradere softwaren.

# Elevforudsætninger

Det forudsættes, at du har et grundlæggende kendskab til microcomputerteknik, som beskrevet i det første kapitel af ”Microcomputerteknik”, fra industriens forlag.

For at kunne afprøve programmerne skal du desuden have et ATMega168 bestykket udviklings board.

# Evaluering

Konstant i gruppearbejdet skal I evaluere for at finde ud af, hvad I får ud af jeres arbejde. Og hvad I evt. skal arbejde mere dybdegående med.

Undervejs i forløbet vil underviseren også evaluere arbejdet i klassen.

# Vejledende tid

Der er afsat 32 lektioner til denne læringsaktivitet.

# Delmålpinde

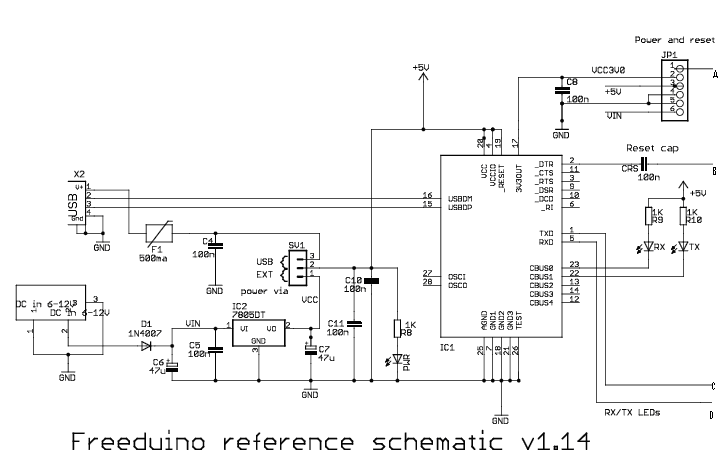
* Du har kendskab til en microcontrollers opbygning, herunder ATMega168
* Du har kendskab til arkitekturer, herunder ”Von Neumann” og ”Harvard”
* Du kan programmere en MCU, og har kendskab til hele denne proces
* Du har kendskab til et instruktionssæt, dets opbygning og anvendelse

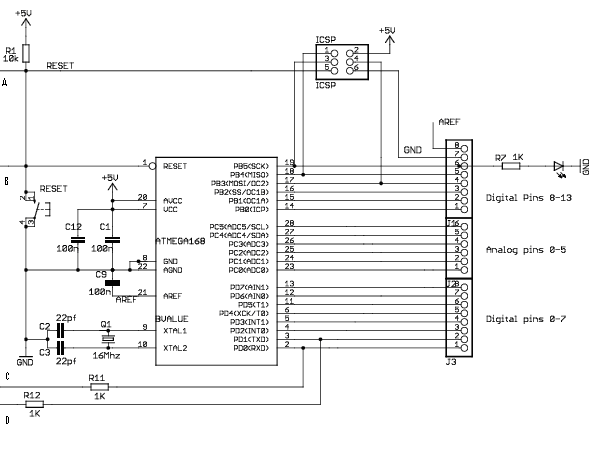
Niveau: Rutine.

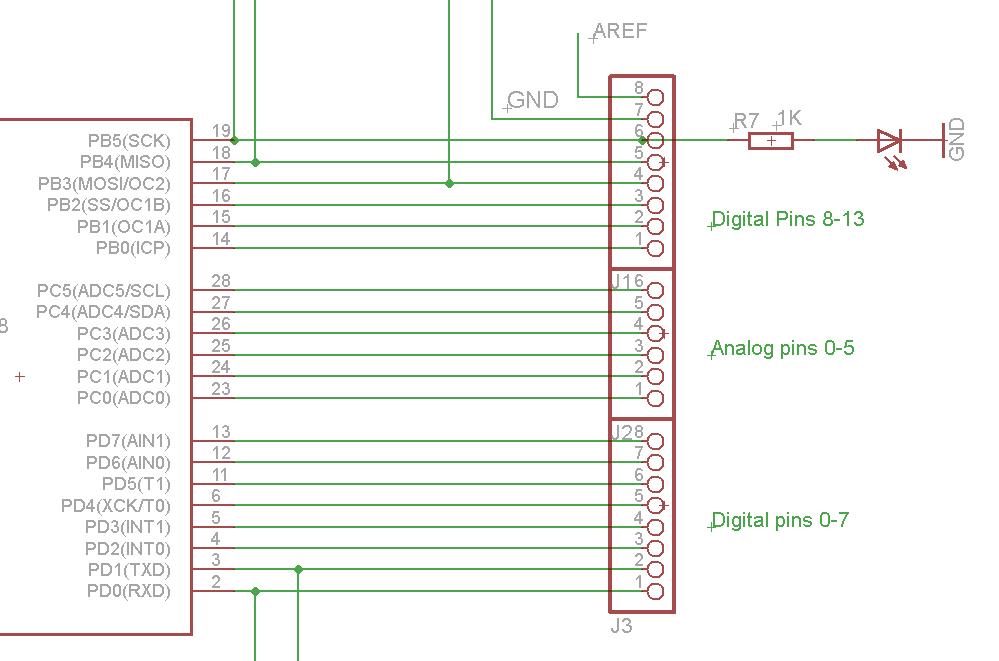
# Relevante henvisninger til supplerende materiale

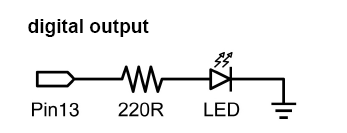
De to første kapitler i ”Microcomputerteknik 2”, fra industriens forlag.  
[www.atmel.com](http://www.atmel.com) → Products → AVR 8bit RISC.  
“Embedded C programming and the Atmel AVR”.  
[www.avrfreaks.net](http://www.avrfreaks.net)  
[www.epanorama.net](http://www.epanorama.net)  
www.avr-asm-tutorial.net

# Diagrammer

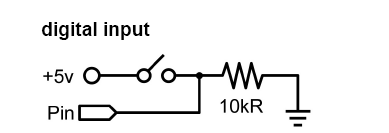
****

****

****

****

**Eller en anden passende modstandsværdi (1k vil også være en fin modstandsværdi at anvende),**

****