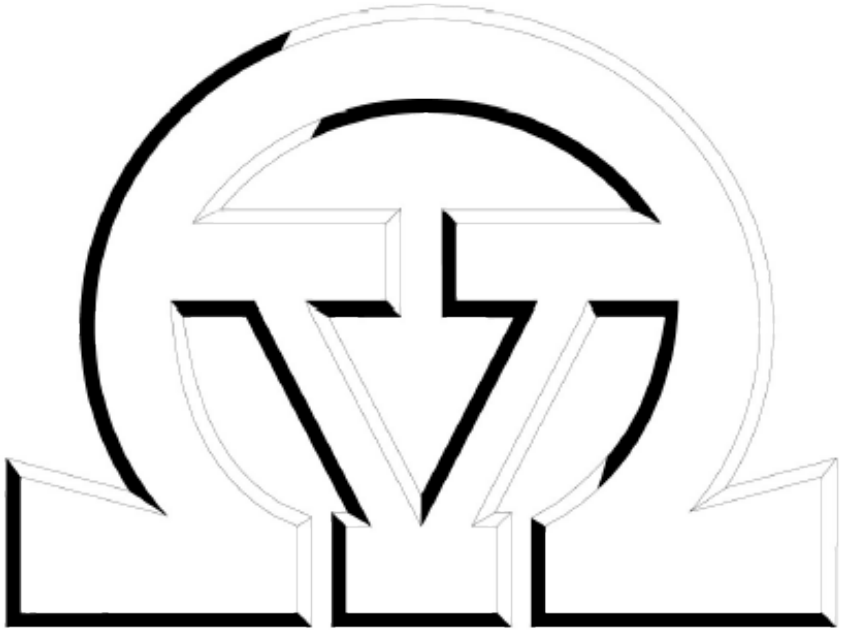
KURSHEFTE
AVR Kurs vår 2016



OMEGA VERKSTED

Grunnleggende om mikrokontrollere.

Printed in The Constitutional Anarchy of Norway.

Innhold

1 FORORD

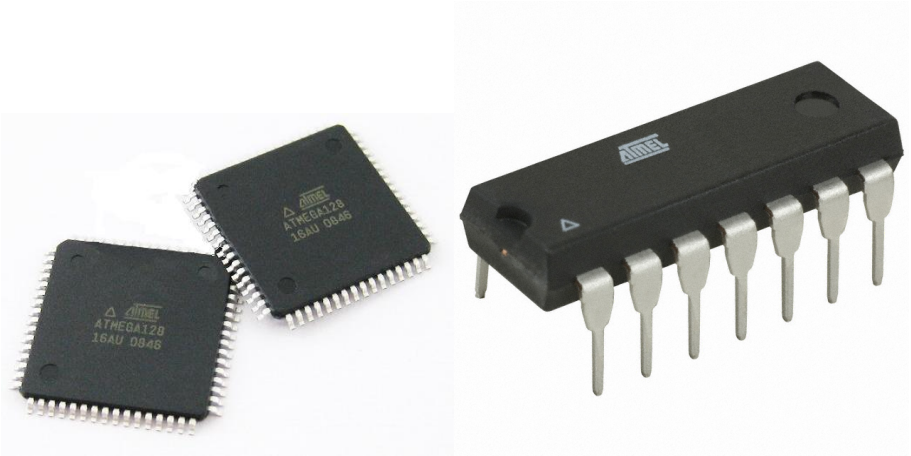
Dette kompendiet er et sammensurium av grunnleggende ting om mikrokontrollere og hva som må til for å kunne programmere de. AVR Kurset er et kræsjkurs og er innom mange aspekter, hvis man ønsker å skjønne alt som foregår i kurset og bli en god mikrokontrollerprogrammerer er det bare en ting som gjelder, øvelse!

Med de ♡ -ligste hilsener og lykkeønskninger

OMEGA VERKSTED

2 MIKROKONTROLLERE

En mikrokontroller (MCU, μC eller uC) er en liten datamaskin på en integrert krets, den inneholder en prosessor, minne og programmerbare Input/Output moduler (I/O pheripherals). Mikrokontrollere har et programminne (flash) som ikke slettes når mikrokontrolleren mister strøm og RAM (Random Acces Memory) som brukes til mellomlagring av data mens et program kjører. I/O enhetene kan ha mange funksjoner fra helt enkle digitale pinner som kan settes til 1 eller 0 til mer avanserte moduler som timere, kommunikasjonsmoduler som USB, USART, SPI, I^2C og CAN, analog til digital konverterere (ADC), digital til analog konverterere (DAC). Det som avgjør prisen på en mikrokontroller er i hovedsak antallet I/O-moduler og hvor avanserte oppgaver de skal gjøre, størrelse på programminne, antall pinner og størrelse på RAM. En typisk 8-bit mikrkontroller kan ha fra 512byte programminne og 32byte med RAM, opp til 256Kbyte programminne og noen Kbyte RAM og pakketyper med 6 til 144 pinner. Det fins også mikrokontrollere med noen megabyte med programminne, mye RAM og 400+ pinner.



(a) TQFP64 (b) DIP14

Figur 1: Noen pakketyper

Alle prosessorere trenger en klokke for å gå, på en typisk brukerdatamaskin i dag er klokkefrekvensen oppe på 3GHz+, som vil si at all logikken i prosessoren tikker og går over 3 miliarder ganger i sekundet. En typisk 8-bit mikrokontroller har en klokkefrekvens på 2-16MHz, her skjer det noe noen millioner ganger i sekundet.

2.1 AVR

AVR®er en serie 8-bits og 32-bits mikrokontrollere fra Atmel®, disse mikrokontrollerne er delt inn i 4 hovedfamilier, TINY, MEGA, XMEGA og UC3.

TINY er den familien med mikrokontrollere med få moduler, få pinner og lite programminne. De har mellom 6-32 pinner, 0.5-8K flash og er de billigste AVR mikroprosessorene. De brukes ofte hvis man må lage et fysisk lite design eller har en enkel oppgave som kan klare seg med en enkel mikrokontroller. **MEGA** er en familie med mikrkontrollere med mange moduler, mye programminne og mange pinner. De har mellom 28-100 pinner og 4-256K flash. De er allsidige

mikrokontrollere som kan brukes i større applikasjoner som trenger større programmer og flere dupeditter. Det meste av mikrokontroller-ting som lages på Omega Verksted bruker en MEGA AVR, nærmere bestemt ATmega128 som er en kontroller med 64 pinner og 128K flash, denne kan brukes til det meste. **XMEGA** er den nyeste 8-bit familien til Atmel. Den har flere moduler enn en typisk MEGA AVR, bedre analogmoduler og en del mer avansert funksjonalitet. **UC3** er Atmels 32-bits mikrokonttrollere,

2.2 MODULER

Det er mye snakk om moduler når man holder på med mikrokontrollere. En modul er kort fortalt en spesiell funksjon mikrokontrolleren tilbyr, alle moduler er implementert i hardware. Dette betyr at når en modul benyttes jobber den alltid i bakgrunnen, uavhengig av hva prosessoren jobber med!, Disse modulene er i utgangspunktet slått av, de skrus på og konfigureres etter behov til applikasjonen man jobber med.

2.2.1 I/O-PORTER

Den vanligste modulen er I/O-porter, i Atmels 8-bit kontrollere er en I/O-port en samling av 8 fysiske pinner på mikrokontrolleren. Disse pinnene er kan individuelt settes som en inngang eller utgang. Hvis en pinne som er satt som utgang blir satt "1/på/høy"vil mikrokontrolleren lage en spenning på denne pinnen tilsvarende driftsspenningen til mikrokontrolleren, blir pinnen satt til "0/av/lav"vil pinnen holde 0 volt. På samme måte kan en pinne som er satt til inngang lese om et signal er høyt eller lavt på utsiden av mikrokontrolleren. Hva kan dette brukes til? Den vanligste tingen man gjør som første mikrokontrollerprogram er å få en lydiode til å blinke, og eventuelt bruke en knapp til å skrud en lydiode av/på. Her er I/O-pinner perfekt!

Dette er en meget grunnleggende modul, alle andre moduler gjør mer avanserte ting og hvis de trenger en pinne for å lese noe / generere en spenning tar de over en standard I/O-pinne.

2.2.2 TIMERE

Timere brukes i hovedsak til å holde rede på tid. Hva trenger man det til? Siden mikrokontrolleren kan gjøre noen millioner operasjoner i sekundet sier det seg selv at hvis noe skal brukes til interaksjon med mennesker rekker ikke menneskene å reagere på noen miliontedels sekunder. Timere kan brukes til å generere signaler på pinner og er en god kilde til å lage noe som skal skje periodisk.

2.2.3 KOMMUNIKASJON

De fleste atmel kontrollere har USART-, TWI- (I^2C) og SPI-moduler, dette er vanlige kommunikasjonsmetoder. TWI og SPI brukes ofte for kommunikasjon mellom integrerte kretser på et kretskort. USART er seriell måte å sende data på som brukes veldig mye. Disse signalene kan konverteres til et hav av elektriske standarder som feks RS232, RS485, USB med mer. Noen Atmel kontrollere har også USB, CAN og LIN kontrollere. USB for å kommunisere med pcer, CAN og LIN er standarder som brukes mye i bilindustrien.

2.2.4 ANALOGE MODULER

Mange Atmel mikrokontrollere har også analoge moduler, en ADC (analog to digital converter) brukes for å måle en analog spenning, og gjøre den om til en digital verdi som kan brukes i en mikrokontroller. Noen Atmel kontrollere har også DAC (digital to analog converter) for å generere analoge spenninger, dette kan feks brukes til å lage referansespenninger, lydsignaler etc.

3 BITS og BYTES

Tall må representeres på en eller annen måte i en mikrokontroller, tall blir representert på binær[1] i hardware, det er også ganske vanlig årepresentere tall i hekadesmial[2] når man programmerer mikrokontrollere. Hvert siffer i et binært tall kalles et bit, en gruppe på 8-bit kalles en byte.

3.1 TALLSYSTEMER

Et gitt heltall i et tallsystem er beskrevet ved siffer a_n, hvert siffer har en vekt b, I desimal systemet er vekten til hvert siffer $b = 10$. For åangi tallsystem kan man for eksempel skrive det desimale tallet 128 som 128_{10} hvor $a_2 = 1$ $a_1 = 2$ $a_0 = 8$. Tallet kan deles ned i vekten til hvert siffer på denne måten: $128_{10} = 1 * 10^2 + 2 * 10^1 + 8 * 10^0$, en generell måte å uttrykke heltall på er:

$$(a_n a_{n-1} \dots a_2 a_1 a_0)_b = \sum_{k=0}^{n} a_k b^k \in [k = 0, 1, 2, \dots n]$$

Dette var jo greit, desimalsystemet er enkelt for oss fordi vi bruker det hele tiden uten å tenke over det på denne måten, hva hvis vi bruker et binært (to-)tallsystem hvor hvert siffer kun kan ha verdien 1 eller 0; 10011010_2, den desimale verdien av dette tallet blir da:

$$10011010_2 = \sum_{k=0}^{7} a_k 2^k = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 128 + 16 + 8 + 2 = 154_{10}$$

En grafisk representasjon av dette kan være:

Vekt	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^1
BIN	1	0	0	1	1	0	1	0
DEC	128	0	0	16	8	0	2	0
HEX	9				A			

Tabell 1: Visuell Tallkonvertering

[1] 2-tallssystem hvor hvert siffer kan ha verdien 1 eller 0

[2] 16-tallssystem hvor hvert siffer kan ha verdien 0-9, A, B, C, D, E eller F

På samme måte kan vi bruke et heksadesmialt (16-)tallsystem hvor sifrene 0-9 og bokstavene A-F brukes, $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$. Det vil si at den desimale verdien av det heksadesimale tallet $EC2F_{16}$ blir:

$$0C2F_{16} = \sum_{k=0}^{3} a_k 16^k = 0 \cdot 16^3 + 12 \cdot 16^2 + 2 \cdot 16^1 + 15 \cdot 16^0 = 0 + 3072 + 32 + 15 = 3119_{10}$$

Dette er en kompakt måte å representere tall på, den største fordelen er når man ser på sammenhengen mellom heksadesimale tall og binære tall. Hvert heksadesimalt siffer kan ha verdiene 0-15, det kan også et firesifret binært tall også ha, det vil si at man enkelt kan konvertere mellom binære og heksadesimale tall.

BIN				**DEC**	**HEX**
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

Vekt	16^3	16^2	16^1	16^0
BIN	0001	0101	1111	1010
HEX	1	5	F	A

Tabell 2: BIN to HEX

3.2 LOGISKE OPERASJONER

Når man programmerer mikrokontrollere får man bruk for å manipulere binære tall og ofte enkelte bit. Her der det fire logiske operasjoner som blir brukt ofte, de kalles **NOT**, **AND**, **OR** og **XOR**. Funksjonen til disse operasjonene er forklart under.

3.2.1 NOT

En NOT operasjon er den enkleste, den inverterer et enkelt bit. 1 blir til 0 og 0 blir til 1: NOT har følgende sannhetstabell.

Innputt	Resultat
1	0
0	1

Innput	1	0	1	1	1	0	1	0
Resultat	0	1	0	0	0	1	0	1

Tabell 3: Sannhetstabell: NOT

3.2.2 AND

AND operasjonen gir ut 1 hvis alle argumentene er 1, eks: 1 AND 1 AND 1 = 1, mens 1 AND 0 AND 1 = 0. AND har følgende sannhetstabell:

Innputt		Resultat
0	0	0
0	1	0
1	0	0
1	1	1

Innput 1	1	0	1	1	1	0	1	0
Innput 2	1	0	0	0	1	0	1	0
Resultat	1	0	0	0	1	0	1	0

Tabell 4: Sannhetstabell: AND

3.2.3 OR

OR operasjonen gir ut 1 hvis et av argumentene er 1, eks: 0 OR 1 OR 0 = 1, mens 0 OR 0 OR 0 = 0. OR har følgende sannhetstabell:

Innputt		Resultat
0	0	0
0	1	1
1	0	1
1	1	1

Innput 1	1	0	1	1	1	0	1	0
Innput 2	1	0	0	0	1	0	1	0
Resultat	1	0	1	1	1	0	1	0

Tabell 5: Sannhetstabell: OR

3.2.4 XOR

XOR gir ut 1 hvis et oddetall av argumentene er 1, eks: 1 XOR 0 = 1, 1 XOR 1 = 0. XOR har følgende sannhetstabell:

Innputt		Resultat
0	0	0
0	1	1
1	0	1
1	1	0

Innput 1	1	0	1	1	1	0	1	0
Innput 2	1	0	0	0	1	0	1	1
Resultat	0	0	1	1	0	0	0	1

Tabell 6: Sannhetstabell: XOR

3.3 BINÆRE OPERASJONER

RIGHT SHIFT og LEFT SHIFT er to operasjoner som brukes mye i mikrokontrollerprogrammering, disse operasjonene dytter alle bittene i et binært tall til henholdsvis høyre eller venstre. Et binært tall: 10110011 RIGHTSHIFT 2 vil dytte tallet to hakk til høyre og padde med nullere, resultatet blir 00101100. En tilsvarende LEFT SHIFT vil gi svaret 11001100.

4 PROGRAMMERING

Mikrokontrollerprogrammering handler i hovedsak om å konfigurere diverse moduler til å gjøre det man vil. Modulene lagrer all konfigurasjon i registre, registerne er 8-bit på 8-bits mikrkontrollere og hvert av bittene i registret har en spessiell funksjon. Når et register er konfigurert og strømmen skrus av, vil registeret nullstilles til standardverdier, dette betyr at programkoden som skrives må konfigurere alle modulene som skal brukes.

Hvor fins det informasjon om moduler, tilhørende register og hva de forskjellige bittene i registerne gjør? Hver mikrokontroller har sitt eget **datablad**. For å programmere mikrokontrollere er det en ting som betyr noe:

RTFD!

Den eneste måten å finne informasjonen som trengs for å programmere mikrokontrollere er i databaldet, de som har sett i et Atmel datablad kan da si, "men det er jo 600+ sider, ingen gidder å lese alt det bare for å programmere litt". Det er helt riktig, trikset er å lære seg å plukke ut informasjonen som trengs. Hver modul har et eget kapittel, det inneholder masse tekst om hvordan den fungerer og bakerst i hvert kapittel er det en seksjon som heter "Register Description". Denne delen inneholder en oversikt over alle registre som tilhører modulen og en beskrivelse av hva hvert enkelt bit gjør. De første gangene man bruker en modul kan det være greit å skumlese litt om modulen og så studere registrene nøye, hvis man lurer på hva noe i registerbeskrivelsen betyr, kan man lese mer om det i kapittlet over. Når man har brukt et par moduler ender det ofte opp med at man hopper rett på registerbeskrivelsen og i mange tilfeller er det nok!

4.1 C

Mikrokontrollerprogrammering er på et lavt nivå med bits og bytes, derfor brukes også lav-nivå programmeringsspråk. De fleste mikrokontollere (alle Atmels kontrollere) kan i dag programmeres i C, før ble det brukt mye assembly som er det nærmeste man kommer maskinkoding med 1-ere og 0-ere.

Et skall for et AVR c-program kan se slik ut;

Listing 1: main.c

```
1  // Tekst etter "//" kalles kommentarer, disse ignoreres av kompilatoren
2  /*
3         For lengre kommentarer kan man bruke denne stilen.
4         Dette gir mulighet for kommentarer over flere linjer.
5  */
6
7  #include <avr/io.h> // Inneholder definisjoner for AVR mikrokontroller
8
9  void IoInit() {
10
11        // DDRB er et register, definisjonen for dette fins i <avr/io.h>
12        DDRB = (1 << PINB0) | (1 << PINB1) | (1 << PINB2); // Sier at PINBn 0,1 og 2 skal settes som utgang,
              skriver over registret.
13        DDRB |= (1 << PINB3); //forkortelse for DDRB = DDRB | (1 << PINB3).
14 }
15
16 int main(){
17
18        uint8_t tall = 0;
19
20        // Initialiserings kode
21        IoInit();
22
23        while(1) {
24
25              // Hoved programkode
26              tall++; // forkortelse for tall = tall + 1;
27
28              PORTB ^= 0xFF; // forkortelse for PORTB = PORTB ^ 0xFF; Inverterer alle bit
29
30
31        }
32
33 }
```

Et C program må ha en main() funksjon, dette er hvor programmet starter. En typisk måte å strukturere et mikrokontrollerprogram på er å gjøre modulinitialisering, og så kjøre et program i en evig løkke slik at programmet aldrig slutter.

Data kan lagres i variabler, variabler kan brukes i regnestykker og lignende. Før en variabel kan brukes må den deklareres med en type, datatyper som er vanlig å bruke i AVR er vist under:

Vanlige datatyper	Beskrivelse
uint8_t	8-bits heltall, verdi fra 0-255
uint16_t	16-bits heltall, verdi fra 0-65 535
char	8-bits heltall, brukt til å lagre tegn (ASCII).
float	32-bits flyttall, brukes når man trenger desimaler.
bool	Sannhetsverdi, TRUE eller FALSE. (#include <stdbool.h>).
void	Tom variabel.

Tabell 7: Variabeltyper

En deklarasjon av en variabel skal inneholde en variabeltype og et navn, den kan også initialiseres med en verdi;

variabeltype navn = initialiseringsverdi
eks: uint8_t minvariabel = 4;

Trenger man en liste med variabler av samme type kan man lage et Array et array deklareres nesten likt som en variabel;

variabeltype navn[n];
eks: uint8_t minvariabel[10]

Her opprettes det 10 variabler, minvariabel[0] til minvariabel[9]. Variabler kan være del av regnestykker, under er noen vanlige operasjoner:

Operasjon	C-syntax	Beskrivelse
$A + B$	$A + B$	Addisjon
$A - B$	$A - B$	Subtraksjon
A/B	A/B	Divisjon
$A * B$	$A * B$	Multiplikasjon
NOT A	$\sim A$	Bitvis NOT
A AND B	$A \,\&\, B$	Bitvis AND
A OR B	$A \mid B$	Bitvis OR
A XOR B	$A \wedge B$	Bitvis XOR
A RIGHT SHIFT B	$A >> B$	Skifter A, B possisjoner til høyre
A LEFT SHIFT B	$A << B$	Skifter A, B possisjoner til venstre

Tabell 8: C-Syntax

Eksempel på bruk av variable:

Listing 2: math.c

```
1  int main() {
2
3          uint8_t var = 0;
4
5          var = 1 + 3 + 5; // var blir 9
6          var = var - 5; // var blir 4
7          var += 3; // var blir 7
8          var = var/2; // var blir 3, desimaler forsvinner.
9          var++ // var blir 4
10
11         while(1){
12
13         }
14
15 }
```

4.1.1 MEGA AVR

$\#include < avr/io.h >$ i eksempelprogrammet for AVR inkluderes definisjoner for Registre og bit i registre. Navnene på registerne og bittene er helt identiske med det som står i databladet.

Et eksempel på et register fra en MEGA kontroller er DDRB, dette registeret bestemmer om pinner er innganger eller utganger.

Bitnummer	7	6	5	4	3	2	1	0	
Bitnavn	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	**DDRB**

Tabell 9: DDRB

Registeret er definert slik at det kan skrives til/leses fra, bitnavnene er definert som bitnummeret i registeret. Det vil si at vi kan sette bestemte bit ved å skrive slik:

Listing 3: register.c

```
1 DDRB = (1 << PINB5); //Disse to linjene er ekvivalente.
2 DDRB = 0b00100000; // Vi venstreskifter et 1-tall inn 5 posisjoner.
```

Her er eksempler på vanlige måter å manipulere registre på, for å forstå hva som skjer se på de logiske operasjonene beskrevet tidligere.

Listing 4: logikk.c

```
1 DDRB = (1 << PINB5) | (1 << PINB6); // Skriver over registeret.
2 DDRB |= (1 << PINB5) | (1 << PINB6); // Setter bit5 og bit6 til 1, beholder resten av registeret.
3 DDRB &= ~((1 << PINB5) | (1 << PINB6)); // Setter bit5 og bit6 til 0.
4 DDRB ^= (1 << PINB5) | (1 << PINB6); // Toggler bit5 og bit6, hvis det er 1 blir det 0 og omvendt.
```

5 NATURLOVER FOR DET MODERNE MENNESKET.

Her har vi tatt med noen lover som er ment å være til oppmuntring om noe av en eller annen grunn ikke skulle virke perfekt første gang:

- Murphys lov: Hvis noe kan gå galt, går det galt. Dersom det ikke går galt, viser det seg at det senere ville vært best om det gikk galt allikevel.
- Boobs law: You always find things in the last place you look.
- Finagle's fourth law: Once a job is fouled up, anything done to improve it only makes it worse.
- H.L. Mencken's law:
  - Those who can — do
  - Those who can't — teach
  - Those who can't teach — administrate
- Lowreys'law: If it jams – force it. If it breaks it needed replacement anyway.
- Harrisberger's fourth law of the lab: Experience is directly proportional to the amount of equipment ruined.
- Jone's law: The man who smiles when anything goes wrong has thought of someone to blame it on.
- Oliver's law: Experience is something you don't get until just after you need it.
- Olke's law: There is no such ting as foolproofness.
- Speer's 1st law: The visibility of an error is inversely proportional to the number of times you have looked at it.
- Loven om forelesere: Enhver foreleser antar at du ikke har noe som helst annet å gjøre enn å arbeide med det faget denne foreleseren foreleser.
- Sattinger's law: It works better if you plug it in.
- O'Toole's Commentary on Murphy's Law: Murphy was an unrealistic optimist.
- Ohm's lov sier noe om motstanden som var imot elektrisiteten til å begynne med.

Dersom disse lovene ikke skulle gi deg noe hjelp med problemet ditt så sett deg ned og spør deg selv: "Hvordan ville MacGyver håndtert denne situasjonen?"

Printet på 100% resirkulerte trær