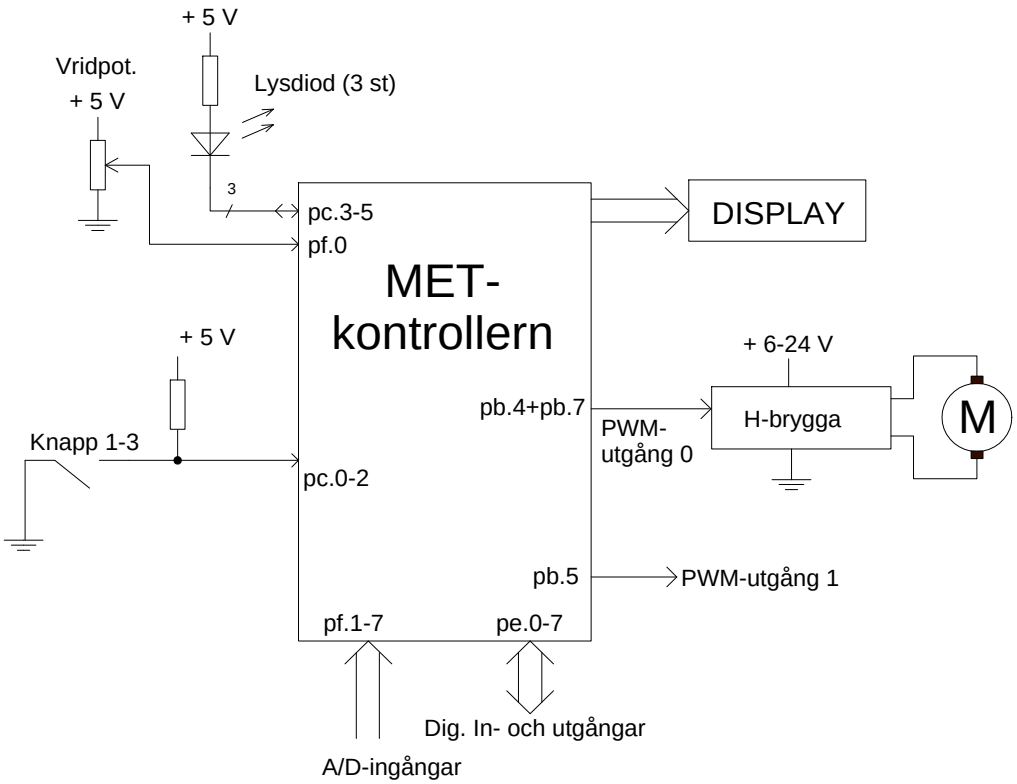
KTH Machine Design

Elektroteknik

Laboration

Experiment med mikrokontroller



Innehåll

1 Laborationens syfte 3
2 Förberedelseuppgifter 3
3 Laborationsutrustningen 3
4 Att komma igång med mikrokontrollern 5
4.1 Hämta drivrutiner och exempel 5
4.2 Starta utvecklingsmiljön 5
4.3 Ett första program. Knappar och display. 8
5 Experiment med digitala portar, AD-omvandlare och PWM-utgång 10
5.1 Realisera grindnät 10
5.2 Fortsättning på Kortlaboration i Digitalteknik 10
5.3 AD-omvandlaren 10
5.4 Använda AD-omvandlaren 11
5.5 PWM-utgången 11
5.6 Variera PWM-signalen 12
6 Motorstyrning 12
6.1 Styr motorn med vridpotentiometern 12
6.2 Styr motorn enligt medelspänningsprofil 13
6.3 Skapa ett nödstopp 13
Bilaga 1 Labplattan 14
Bilaga 2 Lite C-syntax 15
Operatorer och uttryck 15
C-syntax och egna kommandon 16
Bilaga 3 Programutskrifter 19
met.c 19
grind.c 19
hiss.c 20
vridpot.c 22
pwm.c 23

1 Laborationens syfte

Denna laboration är tänkt att introducera en mikrokontroller och tillhörande utvecklingsmiljö för dig. Målsättningen är att ge en bild av vad en mikrokontroller kan användas till, och en idé om hur en sådan programmeras.

I laborationen skall du provköra några befintliga program, skrivna i C, och modifiera dessa enligt specifikationer, samt avslutningsvis skriva ett par egna program där en likströmsmotor först skall styras manuellt från en potentiometer via mikrokontrollerns AD-omvandlare och PWM-utgång och sedan automatiskt.

2 Förberedelseuppgifter

- Läs kapitel 9 i Elektroteknik.
- Läs igenom detta labpek.
- Titta igenom programlistningarna i bilagan. Jämför detta med vad du läst i litteraturen och fundera på hur du skall gå vidare med programmeringsuppgifterna i labben. Bra förberedelse och god planering är nödvändigt för att du skall hinna färdigt laborationen under labpasset.
- Moment 5.3 bygger på kortlabb Dik eller Dis. Repetera och medtag din lösning till hissuppgiften.

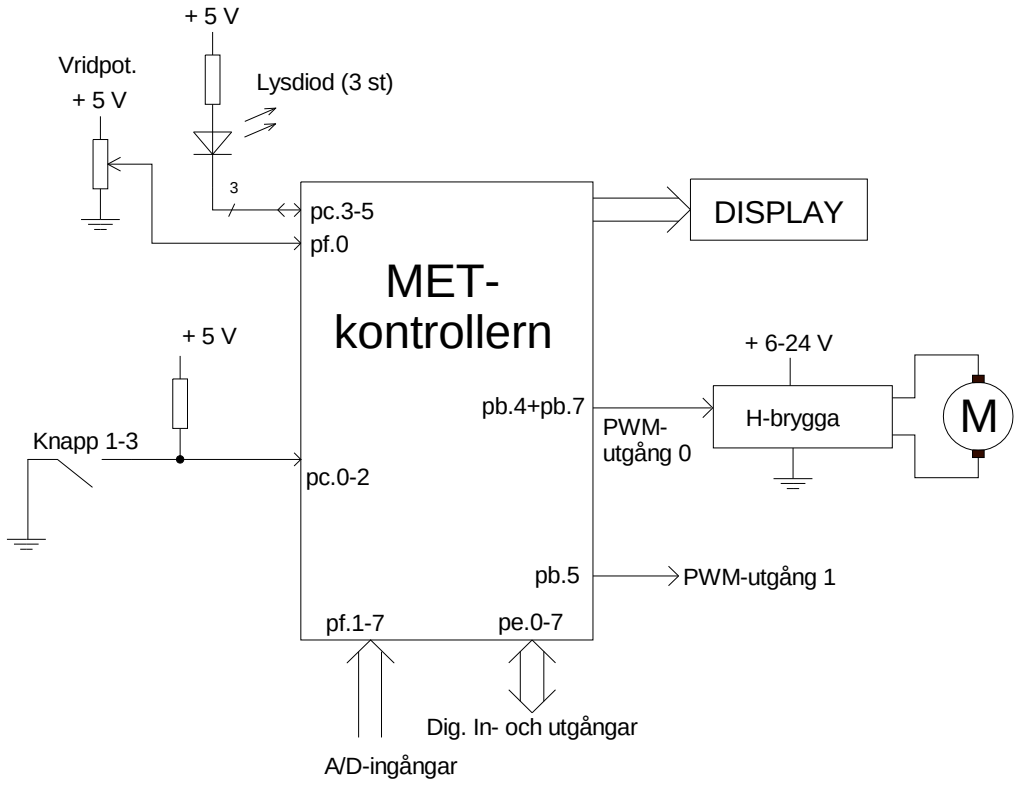
3 Laborationsutrustningen

Laborationsutrustningen består av en PC (*Utvecklingssystemet*) och en labplatta (*Målsystemet*).

PC:n kommunicerar med mikrokontrollern via en USB-JTAG-adapter, med vilken man bland annat laddar ner det program vi skriver på PC:n till mikrokontrollerns eget minne.

Labplattan använder vi för att kunna komma åt mikrokontrollerns enheter och lättare ansluta egna moduler. Det finns en del kringkomponenter som några tryckknappar, lite lysdioder, en display och vridreglage. En likströmsmotor ansluts via kabel till labkortet. Dessa kringkomponenter är konstant anslutna till mikrokontrollern, och vi kommer i denna laboration att programmera mikrokontrollern för att mäta på och styra dessa.

Labplattans in- och utgångar finns beskrivet i bilaga 1 i detta labpek.



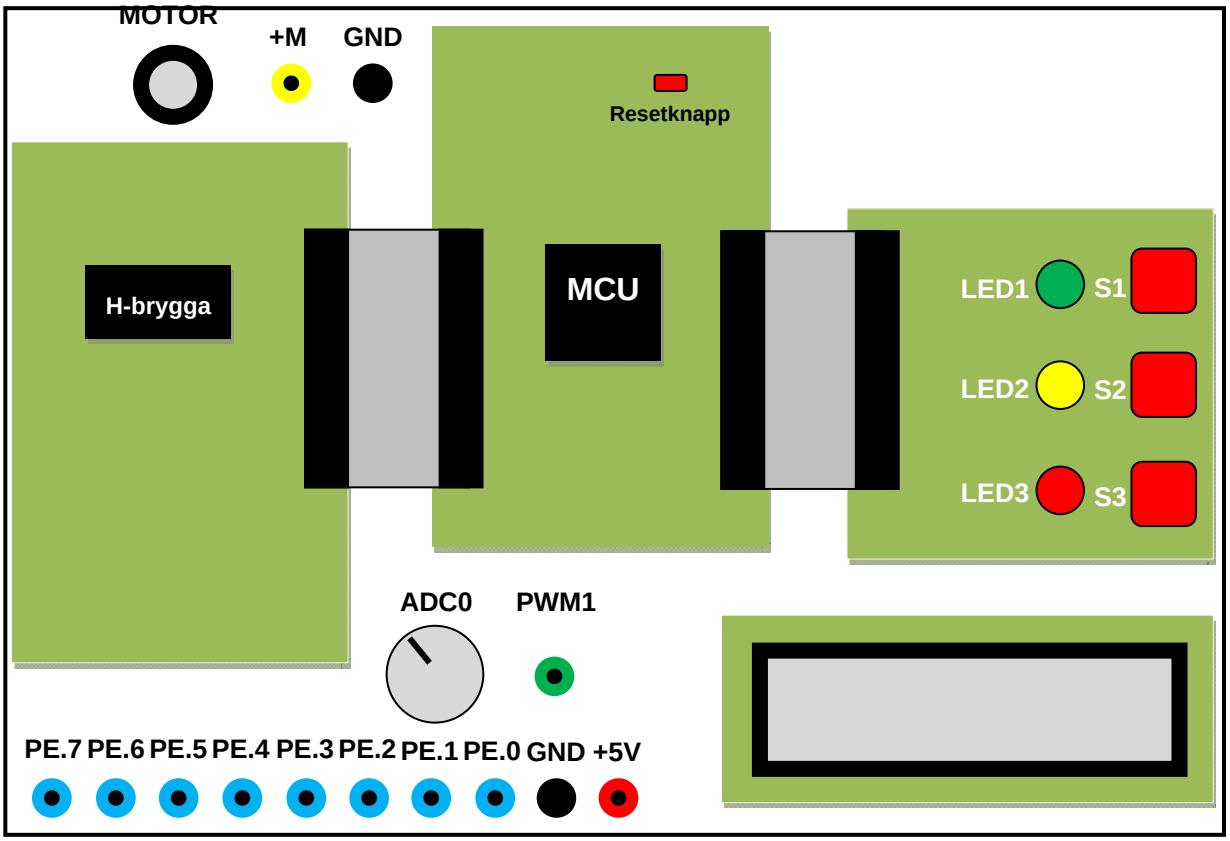
Inkopplingsanvisning.

Läs labbstationens namn t ex Wenström 04 och hämta labbplattan med samma nummer, i detta exempel MIK 04. labplattan. Hämta även en motor och en röd och en svart sladd.

- Anslut till USB
- Anslut matning (DC-plugg)
- Anslut motorn (DC-plugg)
- Anslut +12V på spänningsaggregatet matning till +M på labbplattan med röd sladd
- Anslut GND på spänningsaggregatet till GND på labbplattan (bredvid +M) med svart sladd.
- Visa för assistent.

Nedan en förenklad bild av labplattan.

Ovanstående figur visar några av de anslutningar till mikrokontrollern som finns på labplattan. Dessa kommer du att använda i laborationen. Nedan en förenklad bild av labplattan.



4 Att komma igång med mikrokontrollern

4.1 Hämta drivrutiner och exempel

Logga in som vanligt (eller efter instruktion på whiteboard i labbet).

Det behövs en del extra filer, bl a programexempel för att komma igång:

- Gå till kursens hemsida och välj kurs, omgång och labbar och ladda ner mik.zip till Skrivbordet
- packa upp zip-filen så att mappen ” C:\Documents and Settings\mikro\Skrivbord\met” skapas och kontrollera att den innehåller bl a filen **met.aps**.

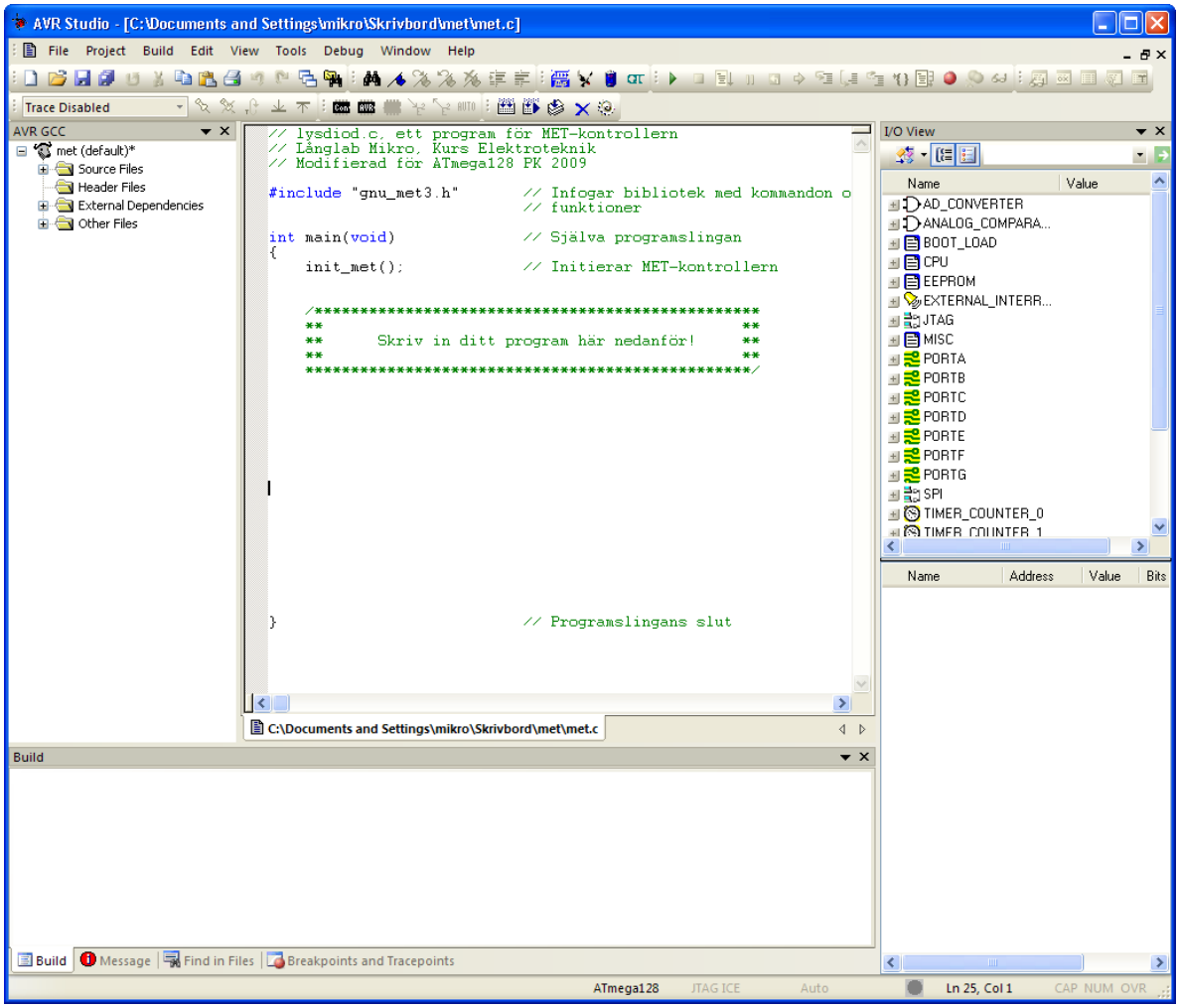
4.2 Starta utvecklingsmiljön

Vi använder oss av AVR Studio 4 för att skriva program för mikrokontrollern ATmega128. I denna laboration ska vi skriva programmen i C samt kompilera dessa med AVR GCC.

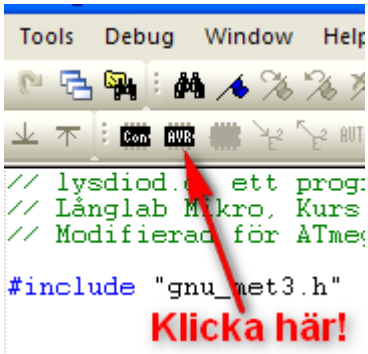
OBS! Se till att labplattan har spänningsmatning – några lysdioder på kretskorten ska lysa.

Öppna mappen MIK på Skrivbordet och dubbelklicka på **met.aps**. AVR Studio 4 startar då utvecklingsmiljön och öppnar filen **met.c** där ni ska skriva in era program.

OBS! Det är viktigt att ni alltid sparar era olika program som **met.c**. AVR Studio är nämligen konfigurerad att kompilera den filen i denna laboration.

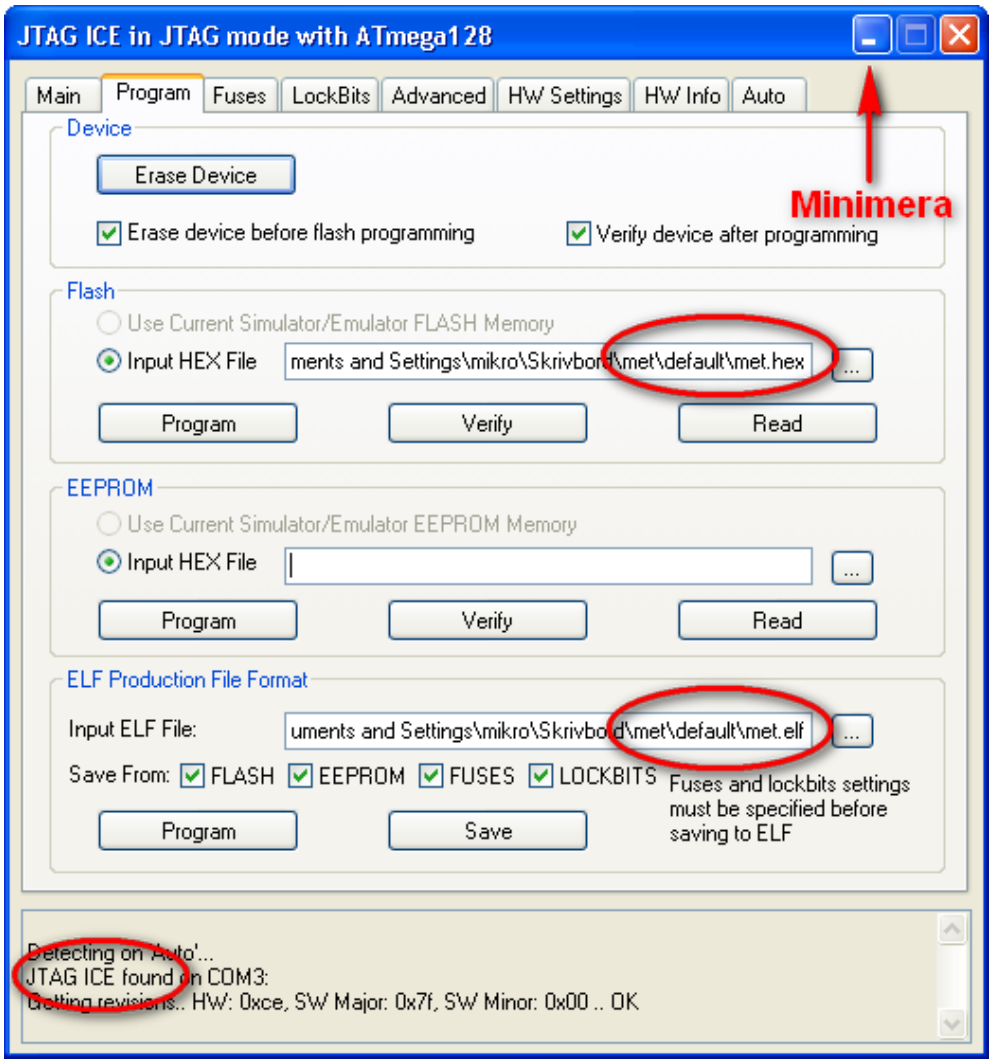


Klicka på den lilla ikonen märkt ”AVR”:



Nu öppnas ett fönster där man kan se om AVR Studio fått kontakt med labplattan och att rätt met-filer är valda i utvecklingsmiljön.

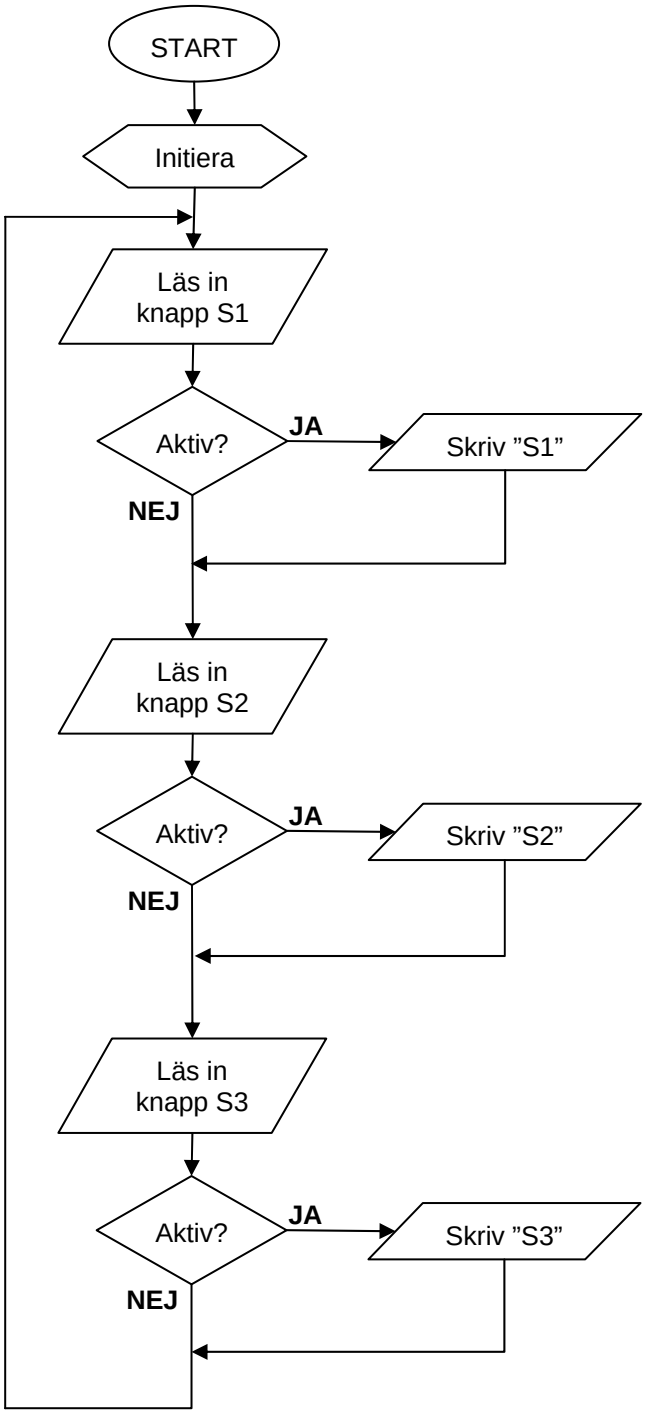
Om de inringade delarna i bilden nedan verkar stämma, **minimerar** du fönstret.



Om texten ”JTAG ICE found” inte syns har ni inte kontakt med labplattan. Antingen är någon kabel inte ansluten eller så saknas matningsspänning – eller så är det något svårare fel som labassistenten får titta på.

4.3 Ett första program. Knappar och display.

Er första uppgift blir att skriva ett program som läser in tryckknapparna S1-S3 och skriver texten ”S1”, ”S2” eller ”S3” på LCD:n då respektive knapp trycks ner.



Till er hjälp kan ni använda flödesschemat till vänster, för att strukturera ert program.

Knapparna är anslutna till den digitala porten PC:s pinnar enligt följande:

S1 : pc.0
S2 : pc.1
S3 : pc.2

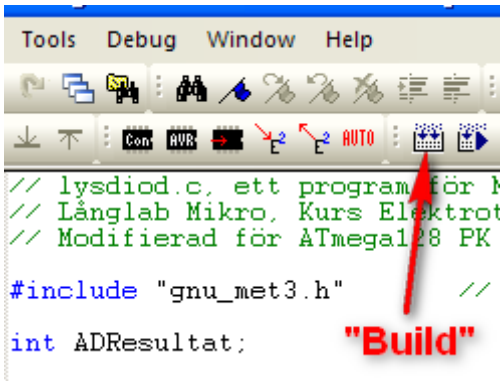
Använd funktionen `GET_BIT` för att läsa knapparna, t ex så här:

```
s1 = GET_BIT(pc, 0);
```

Använd funktionerna `move_cursor` och `dprintf` för att skriva text på displayen – de finns beskrivna i bilaga 2.

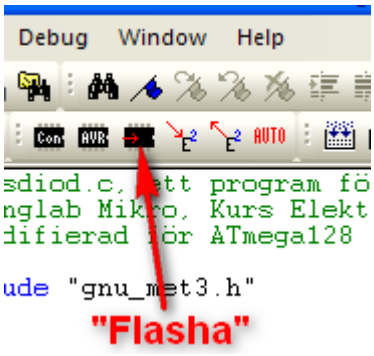
Varje gång ni ska provköra ert nya program ökar ni versionsnumret - variabeln `ver` i programmet - så att man kan se på displayen att ni verkligen laddat ner den nyaste versionen till mikrokontrollern.

När det är dags att testa programmet klickar man på den lilla ”build”-ikonen för att kompilera koden – men försäkra er om att ni sparat den nya versionen av programmet till hårddisken innan ni klickar på ”build”:



Om ni inte gjort några syntaxfel kommer AVR Studio att rapportera noll fel och varningar i rutan nere till vänster. Får vi något fel anges den rad där felet finns. Dubbelklicka på felmarkeringen, så hoppar markören till stället i programmet där felet finns. Granska koden där och försök rätta felet.

När kompileringen går igenom utan fel är det dags att ladda ner maskinkoden till mikrokontrollern – att ”flasha” den. Klicka på ”flash”-ikonen:



Efter några sekunder startar körningen av programmet.
Vill ni starta om det från början trycker ni på den lilla röda resetknappen på MCU-kortet.

I fortsättningen gör ni likadant för att kompilera och köra era program.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5 Experiment med digitala portar, AD-omvandlare och PWM-utgång

5.1 Realisera grindnät

Till detta moment av laborationen skall ni utgå från en exempelfil som heter **grind.c**.

Programmet **grind.c** realiserar mjukvarumässigt funktionen hos en AND-grind. Två portpinnar konfigureras som ingångar, pe.0 och pe.1. En portpinne konfigureras som utgång, pe.2. Kopiera in koden från **grind.c** till **met.c**.

Använd Servicekortet från tidigare labbar och koppla in två strömställare (omkopplare) till pe.0 och pe.1 samt en lysdiod till pe.2. (Glöm inte att ansluta +5 V och jord (GND) från spänningsaggregatet till Servicekortet.)
Verifiera att programmet ger AND-grindens sanningstabell.

5.2 Fortsättning på Kortlaboration i Digitalteknik

Ni som går M, P eller I-programmet (MF1016) ska utgå ifrån kortlaboration Dis och ni som går T, MEDIA eller CL-progammet (MF1017/MF1035) utgår ifrån kortlaboration Dik

I kortlaborationen gjorde ni ett styrsystem till en hiss genom att kombinera olika grindar. Utgå ifrån programmet **grind.c** eller **hiss.**c och modifiera det så att det kan användas för att styra hissen. Anslut alla sex insignaler till labplattan!

Tips: Använd gärna **switch – case**. Utgå i så fall från filen **hiss.c.** (gäller Dis labben)

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.3 AD-omvandlaren

Till detta moment av laborationen skall ni utgå från en ny exempelfil som heter **vridpot.c**.

Programmet **vridpot.c** använder A/D-omvandlaren för att avläsa den analoga vridpotentiometern på labplattan och presentera resultatet på labplattans display. Se kommentarerna i programmet samt läs bilagan om labplattan i peket. Svara på frågorna:

Mellan vilka värden varierar det A/D-omvandlade resultatet?

Svar: Mellan _____ och _____

Mellan vilka värden varierar spänningen från vridpotentiometern?

Svar: Mellan _____ och _____

Vilken är den minsta spänningsändring vi kan detektera?

Svar: _____

5.4 Använda AD-omvandlaren

Utgå från programmet **vridpot.c** som använder funktionen `GET_AD` (se bilaga 2), som läser av spänningen som vridpotentiometern levererar och returnerar ett heltal mellan 0-1023. Komplettera programmet så att ni konverterar AD-omvandlarens resultat till volt, som sen skrivs ut på rad 2 på displayen.

Lägg sen till programkod som tänder lysdioderna LED1-LED3 så att den gröna tänds när inspänningen understiger 2,5 V, den gula tänds när den är precis 2,5 V och den röda tänds när spänningsnivån överstiger 2,5 V. Se även till att lysdioderna släcks i tur och ordning, när man vrider tillbaka potentiometern.

Studera först kretsschemat på sidan 4 och svara på följande fråga:

Skall utsignalen till lysdioderna vara höga eller låga (5 volt = 1 eller 0 volt = 0) för att lysdioderna skall lysa?

Svar: _____

Lysdioderna är anslutna till mikrokontrollerns pinnar pc.3, pc.4 och pc.5.

Funktionerna `SET_BIT` och `CLR_BIT` sätter utgångspinnar höga eller låga – studera i bilaga 2 hur de ska användas för att styra lysdioderna.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.5 PWM-utgången

Till detta moment av laborationen skall ni utgå från koden i en fil som heter **pwm.c**.

Programmet använder PWM-utgång 1, som ger utsignal på portpinne pb.5 och som i sin tur är kopplad till kontakten ”PWM1” på labplattan. Kör programmet, och mät upp följande storheter med ScopeMetern:

PWM-signalens Frekvens.

Svar: _____

PWM-signalens Duty Cycle.

Svar: _____

Mät också signalens medelvärde genom att välja mätområde V_{DC} på ScopeMetern.

Svar: _____

5.6 Variera PWM-signalen

Utgå från programmet **pwm.c**. Ni skall nu modifiera programmet så att PWM-signalens Duty-Cycle varierar från 20% till 80% med en procentenhets ökning åt gången. **Tips!** För att vi skall kunna hinna se ändringen av utspänningen så är det lämpligt att göra en liten paus mellan varje ökning. En paus kan t ex skapas med kommandot `Delay(t)` (se bilaga 2), där *t* byts ut mot lämpligt värde mellan 10 och 100.

Kör programmet och mät upp signalens medelvärde genom att välja mätområde V_{DC} på ScopeMetern.

Mellan vilka värden varierar utspänningen?

Svar: Mellan _____ och _____

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

6 Motorstyrning

Motorn är ansluten till mikrokontrollern via en H-brygga. Denna gör så att vi kan driva motorer som kräver högre ström och spänning än de som mikrokontrollern kan leverera. H-bryggan styrs av en PWM-signal, vars värde sätts med funktionen PWM0 i programmet.

Lägg alltid ut PWM-värdet på både PWM0 (motor) och PWM1 (ScopeMetern). Detta för att samtidigt kunna studera både PWM-spänningen och den effekt som PWM-spänningen har på motorn, via motorstyrkretsen.

6.1 Styr motorn med vridpotentiometern

Ni skall, utifrån de tidigare erfarenheterna med programmen **vridpot.c** och **pwm.c**, göra ett eget program som överför vridpotentiometerns AD-omvandlade värde till ett PWM-värde till H-bryggan. (AD-resultatet skall således ge PWM-signalens Duty-Cycle.)

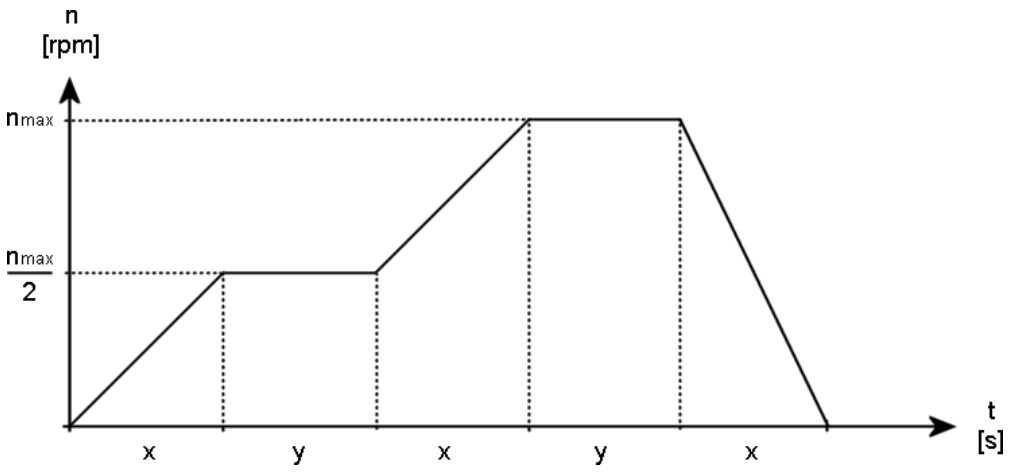
Lägg även här ut värdet från vridpotentiometern på displayen.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

6.2 Styr motorn enligt medelspänningsprofil

I denna uppgift ska ni skriva ett eget program, så att motorn tidsstyrs enligt följande varvtalsprofil:



Tiderna x och y ges av assistenten. Dessa är valda på ett sådant sätt så att varvtalsökning/minskning samt konstant varvtal tydligt kan ses med blotta ögat.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

6.3 Skapa ett nödstopp

För att kunna stoppa motorn vid en nödsituation skall ni även inkludera nödstoppsfunktionalitet i programmet.

Modifiera programmet så att ett tryck på valfri knapp stoppar motorn omedelbart!

Spara ert modifierade program, kompilera och ladda ner det till mikrokontrollern.

När ert program fungerar delredovisar ni det för labassistenten.

Nu är ni äntligen klara med laborationen!

Radera mappen MET från Skrivbordet.

Tack för en väl genomförd laboration!

Bilaga 1 Labplattan

Labkortets komponenter	Anslutningar till mikrokontroller
Inenheter	
Vridpotentiometer Lämnar analog spänning, 0 – 5 V.	`pf.0, dvs AD-ingång 0`
Analoga ingångar (används ej i denna lab)	`pf.2 – pf.7 dvs AD-ingång 2-7`
Tryckknappar Tre tryckknappar är anslutna till digitala ingångar på mikroprocessorn.	`pc.0, pc.1, pc.2`
In- och utenheter	
Externa digitala in- och utportar Åtta uttag för anslutning av separata enheter, tex digitala givare eller styrning av andra system finns på labplattan.	`pe.0 – pe.7`
Utenheter	
Lysdioder, tre st. Anod ansluten till + 5 V via resistorer (470 Ω), katod till kontroller.	`pc.3, pc.4, pc.5`
PWM-utgång Ett eget uttag för anslutning av motorer tex.	`pb.5 (PWM1)`
Display Inte helt enkel att programmera varför vi i denna kurs endast använder färdiga drivrutiner. Se kommandon i Bilaga 2 Lite C-syntax.	
Anslutning för likströmsmotor En H-brygga med separat matning. Denna styrs med pulsbreddsmodulation (PWM) från mikrokontrollern.	`pb.4, pb.7 (PWM0)`

Bilaga 2 Lite C-syntax

Det är inte vår avsikt att du skall vara tvungen att lära dig ett helt nytt programspråk. Vi använder C för att det är enklare att programmera mikrokontrollern i C, och det är en ytterst begränsad mängd kommandon vi kommer i kontakt med. Följande bör räcka väl till för labuppgifterna.

Operatorer och uttryck

Det finns ett flertal operatorer i C som utför operationer på den grundläggande datatypen int (heltal). Dessa sammanfattas i följande tabell:

+	Addition: 5 + 3 ⇒ 8
-	Subtraktion: 5 - 3 ⇒ 2
*	Multiplikation: 5 * 3 ⇒ 15
!	Logisk invers.
&&	Logisk och. I a && b utvärderas b endast om a är skild från noll.
||	Logisk eller. I uttrycket a || b utvärderas b endast om a är noll.
==	Lika med. Returnerar noll om falskt, skilt från noll om sant. Förväxla ej med tilldelningsoperatorn = !
!=	Ej lika med. Returnerar noll om falskt, skilt från noll om sant.
<	Mindre än. Returnerar noll om falskt, skilt från noll om sant.
<=	Mindre än eller lika med. Returnerar noll om falskt, skilt från noll om sant.
>=	Större än eller lika med. Returnerar noll om falskt, skilt från noll om sant.
>	Större än. Returnerar noll om falskt, skilt från noll om sant.

C-syntax och egna kommandon

Ett uttryck i C byggs upp på vanligt sätt av variabler, konstanter och operatorerna ovan, eventuellt med parenteser för att styra beräkningsordningen. Även funktionsanrop kan ingå i uttryck. Sådana anrop görs genom att skriva funktionens namn direkt följt av parametrarna inom parenteser (parenteserna måste finnas där även om funktionen saknar argument).

C-syntax	Enkla kommentarer
`#include "gnu_met3.h"`	I filen gnu_met3.h finns alla deklarationssatser som knyter ihop våra ”arbetsnamn”, tex pe.0 med mikro-kontrollerns fysiska struktur.
`int heltal;`	Variabeldeklaration för heltalsvariabel, t ex: heltal = 128; Kan innehålla både positiva och negativa tal mellan –32768 -- +32767.
`float flyttal;`	Variabeldeklaration för flyttalsvariabler, t ex: y = 47.11;
`char tecken;`	Variabeldeklaration för variabel som kan innehålla ett tecken, te x: tecken = ’t’;

`int main(void)`
`{`
`  programrader`
`}`
	Själva programmet skriver vi som en funktion. Måsvingarna definierar var programmet börjar och slutar. Funktionen måste heta main.

`for(i=1; i<8; i = i + 1)`
`{`
`  summa = summa + 1;`
`}`
	For-snurra
	Så länge räknarvariabeln i är mindre än 8 så skall variabeln `summa` räknas upp med ett.

`while(1)`
`{`
`}`
	While-snurra
	Evighetsloop! Snurra så länge 1 = 1.

`while(adam >= evert)`
`{`
`  programrader`
`}`
	Så länge variabeln `adam` är större än eller lika med variabeln `evert` genomförs programraderna mellan ”måsvingarna”.

`if(x > y)`
`{`
`  max = x;`
`}`
`else`
`{`
`  max = y;`
`}`
	Villkorssats
	Om x = 100 och y = 75 kommer max tilldelas värder på x, dvs 100. Om x = 50 och y = 75 kommer max att tilldelas värdet på y, dvs 75.

`if(elvis < 100)`
`{`
`  SET_BIT(pe, 0);`
`}`
`else`
`{`
`  CLR_BIT(pe, 0);`
`}`
	Om elvis är mindre än 100 så sätts pe.0 till 1. I annat fall sätts pe.0 till 0.

`switch (dag)`
`{`
`  case 1 : text = 'S';`
`  break;`
`  case 2 : text = 'M';`
`  break;`
`  case 3 : text = 'T';`
`  break;`
`  case 4 : text = 'O';`
`  break;`
`  case 5 : text = 'T';`
`  break;`
`  case 6 : text = 'F';`
`  break;`
`  case 7 : text = 'L';`
`  break;`
`  default : text = '?';`
`  break;`
`  }`
	En switch/case-sats fungerar så, att beroende på switch-variabelns värde så utförs en av case-satserna.
	Om dag == 4 kommer programmet att hoppa in vid "case 4" och sätta teckenvariabeln "text" till bokstaven "O". Därefter hoppar programmet ut ur switch-satsen.
	Skulle dag ha ett annat värde än 1-7 kommer programmet att gå till "default" där man kan fånga upp sådana fel och agera på lämpligt sätt.

Några kommandon specifika för vår labplatta	
`init_met();`	Initerar MET-kontrollern.
`init_pe(6,"out");`	Initierar port e pinne 6 att vara en utpinne.
`init_pe(3,"in");`	Initierar port e pinne 3 att vara en inpinne.
`SET_BIT(pe, 0)`	Portpinne pe.0 ettställs, dvs kommer att ha värdet 1 och det motsvarar 5 Volt.
`CLR_BIT(pe, 1)`	Portpinne pe.1 nollställs, dvs kommer att ha värdet 0 och det motsvarar 0 Volt
`bit_in = GET_BIT(pe, 2);`	Portpinne pe.2 avläses och resultatet lagras i variabeln bit_in.

`PWM0(20);`	PWM0(20) ger en likspänning på 1 volt på PWM-utgång 0 (20% av matnings-spänningen 5 volt). Kommandot accepterar värden mellan 0 – 100.
`PWM1(50);`	PWM1(50) ger en likspänning på 2.5 volt på PWM-utgång 1 (50% av matnings-spänningen 5 volt). Kommandot accepterar värden mellan 0 – 100.
`ADResultat = GET_AD(7);`	Läser in det värdet på en likpänning som anslutits till ingång 7 (pf.7) till variabeln ADResultat. Funktionen returnerar ett heltal mellan 0 och 1023. Spänningen måste ligga inom 0 och 5 volt.
`Delay(1000);`	Ger en paus på 1 sekund (1000 ms) vid 8MHz klockfrekvens hos CPU:n.
`clear_disp();`	Rensar displayen.
`move_cursor(1,2);`	Flyttar markören till rad 2, kolumn 1. Displayen består av två rader och 16 kolumner.
`dprintf(”Tjena!”);`	Skriver texten Tjena! på displayen.
`dprintf(”Jag har %i kr!”,peng);`	Skriver text och en variabel. I detta måste ingå ett %i, som anger att variabeln är av typen integer. Om variabeln peng har värdet 180 kommer *Jag har 180 kr!* att skrivas ut på displayen.

Bilaga 3 Programutskrifter

met.c

```
// met.c, ett program för MET-kontrollern
// Långlab Mikro, Kurs Elektroteknik
// Modifierad för ATmega128 PK 2009

#include "gnu_met3.h"              // Infogar bibliotek med funktioner

char * prog = "Knapptest";         // Textsträng med programmets namn
int ver     = 0;                   // *** ÖKA numret för varje version ni
                                   // flashar! ***

int main(void)                     // Själva programslingan
{
    init_met();                    // Initierar MET-kontrollern

    move_cursor(1,1);              // Displaymarkören till rad 1, kolumn 1
    dprintf("%s v.%i", prog, ver);  // Skriv ut programmets namn och version

    while (1)
    {

        /*********************************************************
        **                                                      **
        **        Skriv ett program som läser in knapparna      **
        **        S1, S2 och S3 och skriver vilken som är       **
        **        nedtryckt, på displayens andra rad.           **
        **                                                      **
        *********************************************************/









    }
}                                  // Programslingans slut
```

grind.c

```
// grind.c, ett program för MET-kontrollern
// Långlab Mikro, Kurs Elektroteknik
// MG & TL 2002
// Modifierad för ATmega128 PK 2009

#include "gnu_met3.h"              // Infogar bibliotek medfunktioner

char * prog = "Grind";             // Textsträng med programmets namn
int ver     = 0;                   // *** ÖKA numret för varje version ni
                                   // flashar! ***
```

```
int a, b, c;                          // Deklaration av variablerna a, b och c

int main(void)                        // Själva programslingan
{
    init_met();                       // Initierar MET-kontrollern

    move_cursor(1,1);                 // Displaymarkören till rad 1, kolumn 1
    dprintf("%s v.%i", prog, ver);    // Skriv ut programmets namn och version

    init_pe(0,"in");                  // Initierar pe.0 som inpinne
    init_pe(1,"in");                  // Initierar pe.1 som inpinne
    init_pe(2,"out");                 // Initierar pe.2 som utpinne

    while(1)                          // Evighetsloop!
    {
        a = GET_BIT(pe, 0);           // Läser av pe.0
        b = GET_BIT(pe, 1);           // Läser av pe.1
        c = a && b;                   // Logisk AND
        if(c == 1) SET_BIT(pe, 2);    // Utsignal beroende på c
        else CLR_BIT(pe, 2);
    }
}                                     // Programslingans slut
```

hiss.c

```
/* hiss.c the elevator example once again       */
/* Connect the elevator to MET-controller PORT E */
// Modifierad för ATmega128 PK 2009
/*
*****************************************************
Statechart

            ( going_up,10  )
            !Sp2 => going_up
             Sp2 => upstairs

( downstairs,00  )       ( upstairs,11            )
!Upp && !Kp2 => Z0       !Ner && !Kp1 => upstairs
 Upp ||  Kp2 => Z1        Ner ||  Kp1 => going_down

            ( going_down,01  )
            !Sp1 => going_down
             Sp1 => downstairs

*****************************************************
*/

#include "gnu_met3.h"

#define downstairs 0
#define going_up   1
#define upstairs   2
#define going_down 3

char * prog = "Hiss";                 // Textsträng med programmets namn
int ver     = 0;                      // *** ÖKA numret för varje version ni
                                      // flashar! ***
```

```
int main(void)
{
    int state = downstairs;         // Initiera state till "downstairs"
    int Upp, Ner, Kp2, Sp2, Kp1, Sp1;

    init_met();

    move_cursor(1,1);               // Displaymarkören till rad 1
    dprintf("%s v.%i", prog, ver);  // Skriv ut programmets namn och version

    init_pe(7,"out");               // Mupp
    init_pe(6,"out");               // Mner
    init_pe(5,"in");                // Upp
    init_pe(4,"in");                // Ner
    init_pe(3,"in");                // Kp1
    init_pe(2,"in");                // Kp2
    init_pe(1,"in");                // Sp1
    init_pe(0,"in");                // Sp2

    while(1)
    {
        Upp = GET_BIT(pe, 5);
        Ner = GET_BIT(pe, 4);
        Kp1 = GET_BIT(pe, 3);
        Kp2 = GET_BIT(pe, 2);
        Sp1 = GET_BIT(pe, 1);
        Sp2 = GET_BIT(pe, 0);

        switch(state)
        {
            case downstairs:
                CLR_BIT(pe, 7);                             // Mupp = 0
                CLR_BIT(pe, 6);                             // Mner = 0
                if( !Upp && !Kp2 ) state = downstairs;
                if(  Upp ||  Kp2 ) state = going_up;
                break;

            case going_up:
                SET_BIT(pe, 7);                             // Mupp = 1
                CLR_BIT(pe, 6);                             // Mner = 0
                // Ooops! vad ska stå här?
                break;

            case upstairs:
                SET_BIT(pe, 7);                             // Mupp = 1
                SET_BIT(pe, 6);                             // Mner = 1
                // Ooops! vad ska stå här?
                break;

            case going_down:
                CLR_BIT(pe, 7);                             // Mupp = 0
                SET_BIT(pe, 6);                             // Mner = 1
                // Ooops! vad ska stå här?
                break;
        }

        Delay(100);                 // 100 ms fördröjning
    }
}                                   // Programslingans slut
```

vridpot.c

```
// vridpot.c, ett program för MET-kontrollern
// Långlab Mikro, Kurs Elektroteknik
// MG & TL 2002
// Modifierad för ATmega128 PK 2009

#include "gnu_met3.h"               // Infogar bibliotek med funktioner

char * prog = "Vridpot";            // Textsträng med programmets namn
int ver           = 0;              // *** ÖKA numret för varje version ni
                                    // flashar! ***

int ad_value;                       // Deklaration av heltalsvariabeln ad_value
float volt;                         // Deklaration av flyttalsvariabeln volt

int main(void)                      // Själva programslingan
{
    init_met();                     // Initierar MET-kontrollern

    move_cursor(1,1);               // Displaymarkören till rad 1
    dprintf("%s v.%i", prog, ver);  // Skriv ut programmets namn och version

    while(1)                        // Evighetsloop
    {
        ad_value = GET_AD(0);       // AD-omvandla ingång 0 (pf.0)

        volt = ?                    // Hur ska "volt" räknas ut?

        move_cursor(1,2);           // Displaymarkören till rad 2
        dprintf("Volt: %f ", volt); // Skriv ut variabel volt


        /*********************************
        **                              **
        ** Skriv in koden för att tända **
        ** lysdioderna här nedanför.    **
        **                              **
        *********************************/









    }
}                                   // Programslingans slut
```

pwm.c

```
// pwm.c, ett program för MET-kontrollern
// Långlab Mikro, Kurs Elektroteknik
// MG & TL 2002
// Modifierad för ATmega128 PK 2009

#include "gnu_met3.h"               // Infogar bibliotek med funktioner

char * prog = "PWM";                // Textsträng med programmets namn
int ver     = 0;                    // *** ÖKA numret för varje version ni
                                    // flashar! ***

int duty_cycle;                     // Deklaration av variabeln duty_cycle

int main(void)                      // Själva programslingan
{
    init_met();                     // Initierar MET-kontrollern

    move_cursor(1,1);               // Displaymarkören till rad 1, kolumn 1
    dprintf("%s v.%i", prog, ver);  // Skriv ut programmets namn och version

    duty_cycle = 25;                // Tilldelning av variabelvärde
    PWM1(duty_cycle);               // Ändrar utspänning på PWM-utgång 1

    while(1)                        // Evighetsloop
    {
                                    // Här görs ingenting, förutom att vi
                                    // låser fast processorn i en
                                    // evighetsloop!
    }
}                                   // Programslingans slut
```

Dukningslista Klab Mik

Mikrokontroller

Antal	Utrustning	
1	Labpalattan	Står framme
1	Likströmsmotor	Står framme
1	Kopplingsbox (ansluten till mätkort)	Står framme
1	ScopeMeter	Står framme
1	Servicekort (strömbrytare, lysdioder mm)	
6	Röd laboratoriesladd	
4	Svart laboratoriesladd	