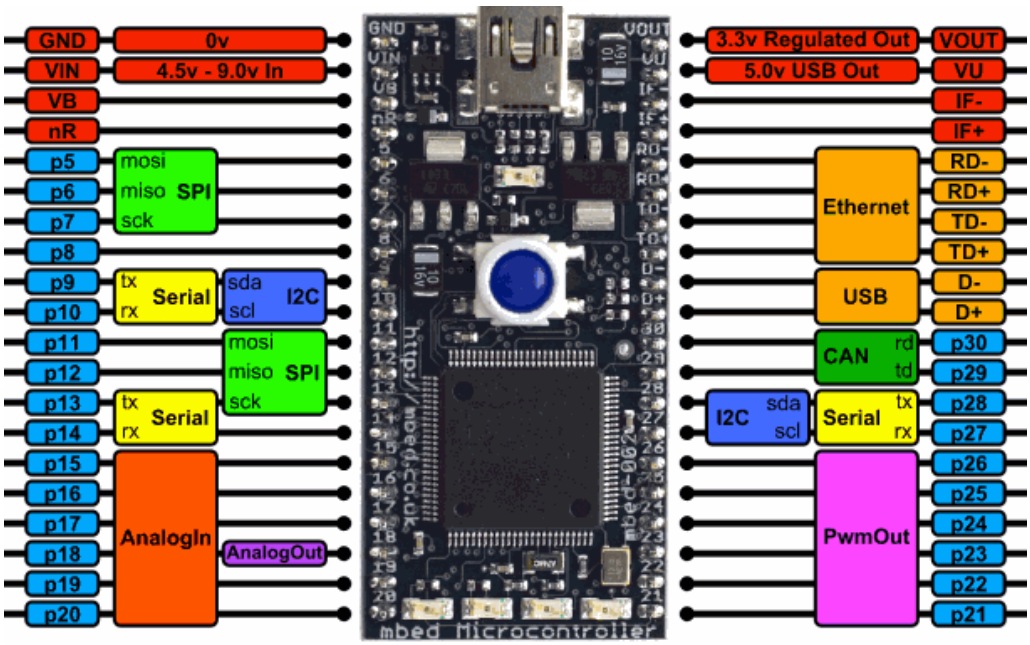
KTH Machine Design

Elektroteknik

Laboration

Experiment med en mikrokontroller



Innehåll

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 utgå från ett befintliga programskelett skrivet i C och succesivt fylla det med innehåll.

2 Förberedelseuppgifter

- Läs kapitel 9 i Elektroteknik.
- Läs igenom detta labpek noggrant.

3 Laborationsutrustningen

Laborationsutrustningen består av en PC (*Utvecklingssystemet*) och en MCU-låda (*Målsystemet*).

På MCU-lådan är en modul kallad ”mbed” monterad (se bilden på omslaget). Denna innehåller en mikrokontroller ur ARM-familjen, som används bland annat i moderna mobiltelefoner.

Programmet man skriver på PC:n laddas ner till mikrokontrollern via en USB-sladd som även ger spänningsmatning till modulen.

MCU-lådan använder vi för att komma åt mikrokontrollerns inbyggda enheter och lättare ansluta kringkomponenter, t ex tryckknappar och lysdioder. Dessutom har den en LCD där ni kommer att skriva ut data från era program. Kringkomponenter ansluts med hjälp av de vanliga labsladdarna med 2mm-kontakter.

Plocka fram:

1 st MCU-låda

2 st 10-plintar

1 st motorbänk

1 st H-brygga

1 st potentiometer

1 st tryckknapp

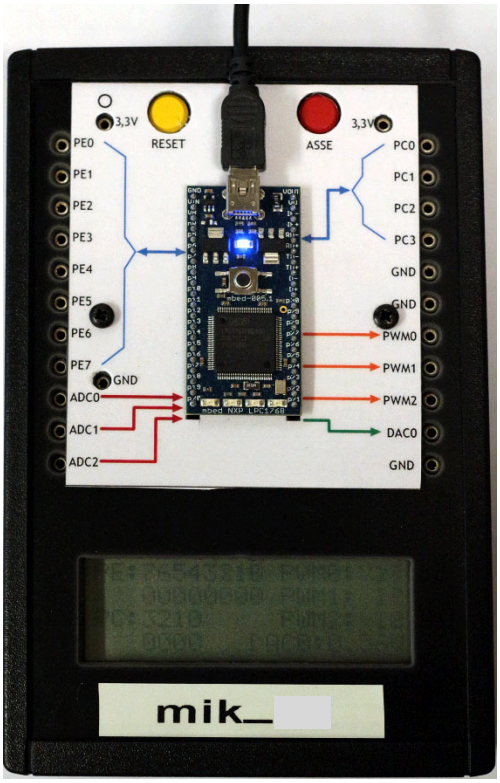
1 st gul lysdiod

1 st dubbelriktad röd/grön lysdiod (den med ofärgad kapsel)

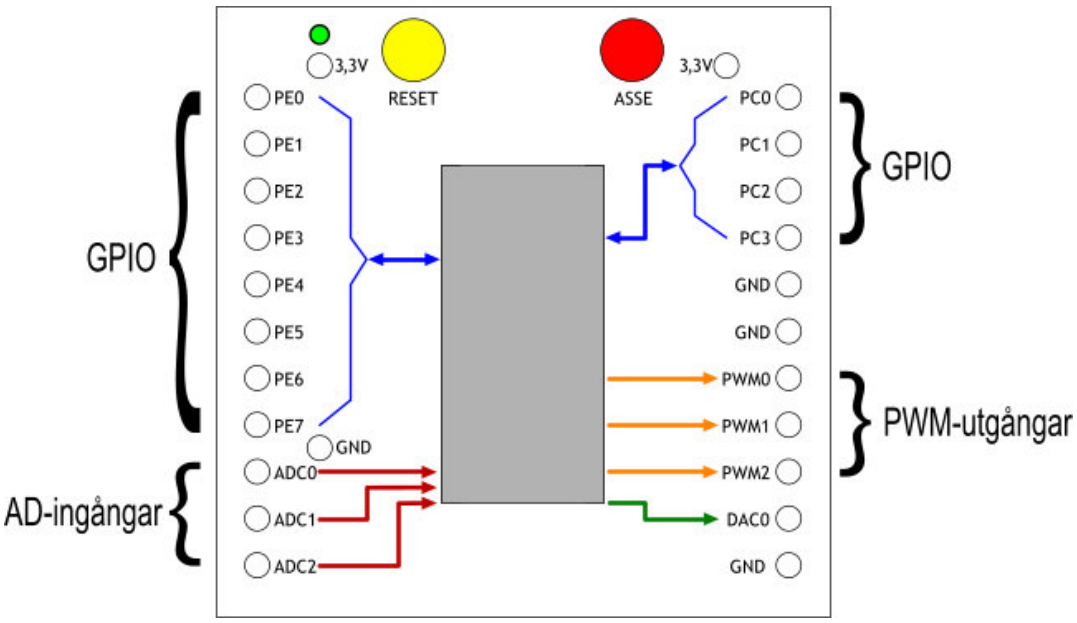
2 st motstånd på minst 1 kohm (värdet är mindre viktigt)

Labsladdar: 2 st gula, 5 st röda, 5 st svarta

Nedan ses MCU-lådan med mbed-modulen monterad i mitten.



Diverse signaler till/från mbed-modulen är utdragna till 2mm-hylsor. De som är tillgängliga i denna laboration är indikerade nedan. GND (=*ground*) är 0 V, även kallad ”jord”.



4 Att komma igång med mikrokontrollern

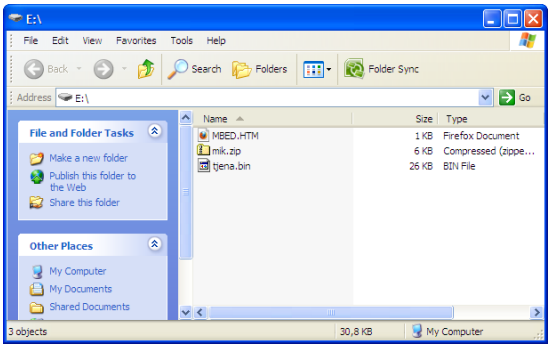
4.1 Anslut MCU-lådan till PC:n via USB

När ni ansluter MCU-lådan PC:n som finns under labbänken skall lysdioden markerad ”3,3 V” tändas och någon text synas i displayen.

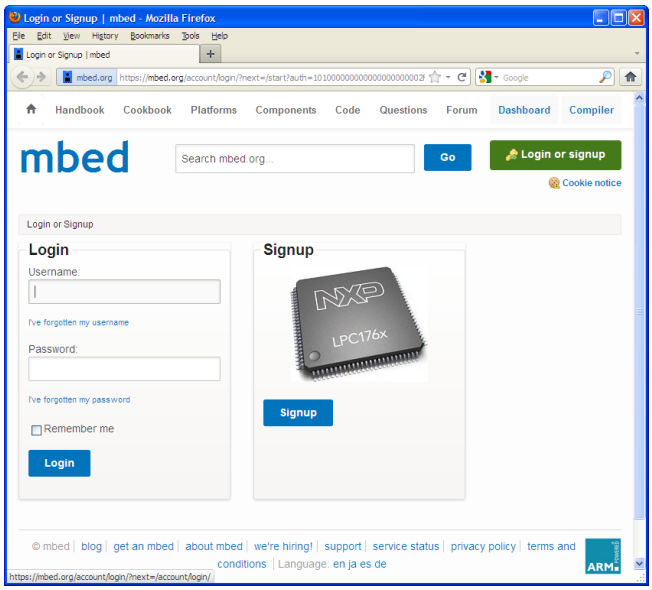
MCU-lådan spänningsmatas från USB-sladden och ingen annan matningsspänning skall kopplas in till den.

4.2 Starta utvecklingsmiljön

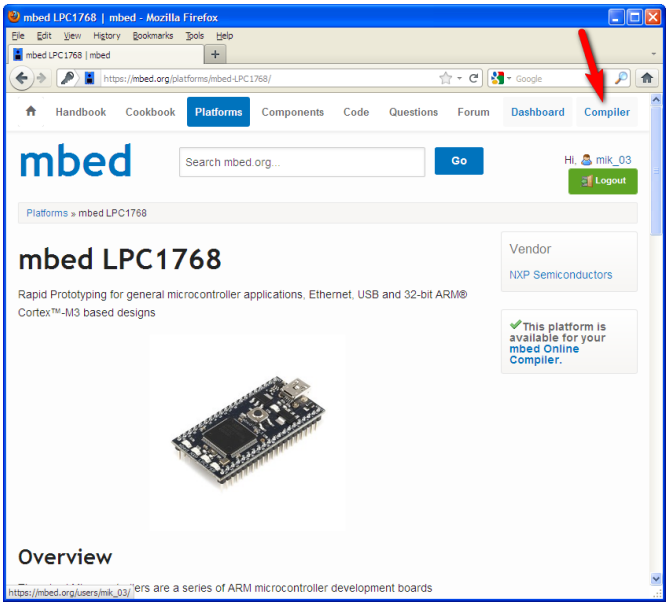
Då PC:n upptäcker mbed-modulen öppnar den en enhetsmapp (vanligen E:\) som visar filerna som är lagrade i modulen – dubbelklicka på MBED.HTM.



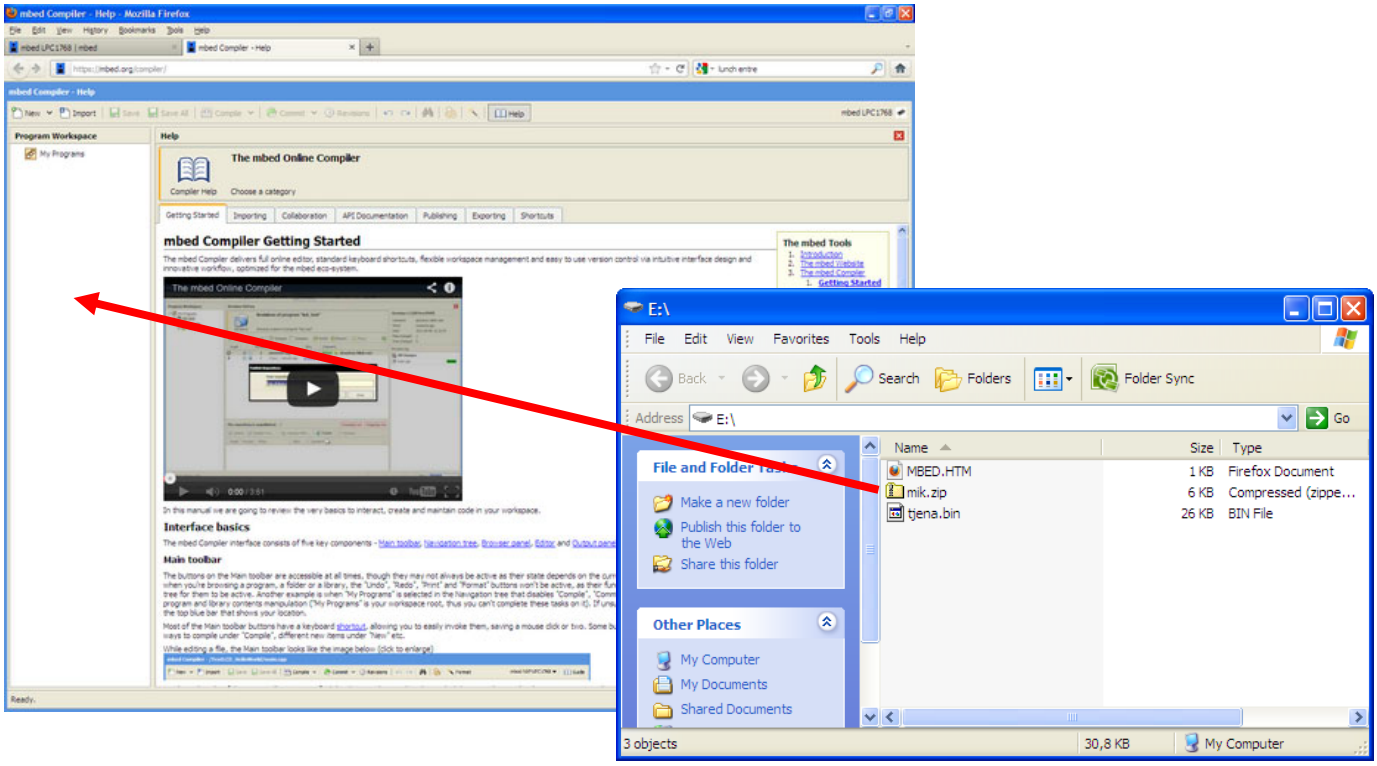
Ett browserfönster öppnas där ni ombeds logga in. Användarnamnet står på MCU-lådan – **mik_xx** – och lösenordet skall synas i displayen. Kalla på labassistenten om det krånglar.



Då ni loggat in klickar ni på ”Compiler” uppe i högra hörnet.



Nu öppnas en ny browserflik med utvecklingsmiljön och då drar ni **mik.zip** från E:\ till den vänstra spalten ”Program Workspace”:

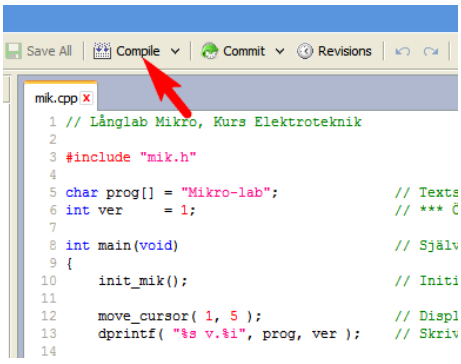


Klicka sedan på + vid mappen ”mik” och därefter på ”mik.cpp”. Då ska programmet öppnas i editeringsfönstret till höger.

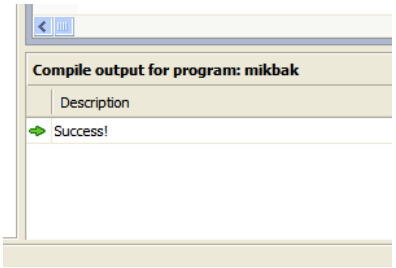
Programskelettet ser ut som nedan:

```
// Långlab Mikro, Kurs Elektroteknik

#include "mik.h"

char prog[] = "Mikro-lab";              // Textsträng med programmets namn
int ver     = 1;                        // *** ÖKA gärna numret för varje version ni testkör! ***

int main(void)                          // Själva programslingan
{
    init_mik();                         // Initiera hårdvaran

    move_cursor( 1, 5 );                // Displaymarkören till rad 1, kolumn 5
    dprintf( "%s v.%i", prog, ver );    // Skriv ut programmets namn och version

    /*********************************************************************
    **                                                                  **
    ** Deklarera variabler här nedanför vartefter som ni behöver dem    **
    **                                                                  **
    *********************************************************************/

    int state = 0;

    /*********************************************************************
    **                                                                  **
    ** Konfigurera GPIO som in- och utgångar här nedanför med init_pin  **
    **                                                                  **
    *********************************************************************/


    while( 1 )                          // Evighetsloop
    {

    }
}                                        // Programslingans slut
```

I det nuvarande skicket gör programmet inget mer än skriver ut en text på displayen, men det går att kompilera. Gör detta genom att klicka på ”Compile”:



Resultatet av kompileringen visas i statusfönstret längst ned:



Kompilatorn frågar var den kompilerade bin-filen skall sparas – lägg den i E:\, dvs mbed-modulen. Den blå lysdioden på mbed-modulen kommer då att flimra medan filen laddas ned. När den slutat flimra trycker ni på RESET-knappen för att köra det nedladdade programmet.

Om allt fungerat skall ni nu se texten ”Mikro-lab v.1” på displayen – om ej, tillkalla labassistenten.

Tips: Varje gång man trycker på RESET startar programmet om från början.

5 Laborationen

Ni har fått i uppdrag av en kund att skriva ett program för motorstyrning. Det ska fungera så att då man trycker på en knapp skall motorn starta och öka farten till ett förbestämt värde. Därefter skall man kunna variera motorns hastighet med en vridpotentiometer. Vid ett nytt tryck på knappen skall motorn minska farten till stillastående.

Under laborationen kommer ni att bygga upp programmet gradvis till ökande komplexitet. Förhoppningsvis når ni slutmålet och har då ett fungerande och väl strukturerat program. Lycka till!

5.1 Moment 1 – GPIO

GPIO står för *General Purpose Input Output* och betecknar ett antal av mikrokontrollerns fysiska pinnar, som kan konfigureras antingen som digital in- eller utgång.

För att bestämma pinnens ”riktning” används funktionen `init_pin(n,dir)` där *n* står för pinnens namn och *dir* för riktningen.

Exempel: `init_pin(pe0,"in")` gör pe0 till ingång och `init_pin(pc0,"out")` gör pc0 till utgång.

Efter konfigureringen kan man i sitt program läsa ingången med funktionen `x=GET_BIT(pe0)`, där *x* blir 1 respektive 0 beroende på om ingången är logiskt hög eller låg.

För mikrokontrollerns GPIO-ingångar är spänningsnivåer definierade så att ”hög” ingång fås då inspänningen är >2,0 V och ”låg” som <0,8 V.

Med funktionerna `SET_BIT(n)` och `CLR_BIT(n)` sätter man utgångar höga respektive låga.

Exempel: `SET_BIT(pc0)` sätter utspänningen på pinne pc0 ”hög”, >2,0 V.

Ibland är GPIO-ingångar högohmiga, vilket medför att inspänningen inte blir väldefinierad hög eller låg, utan hamnar i det så kallade ”förbjudna området” mellan 0,8-2,0V. Då kan inte mikrokontrollern bestämma sig för ingångens logiska tillstånd och kommer slumpmässigt läsa den som 0 eller 1.

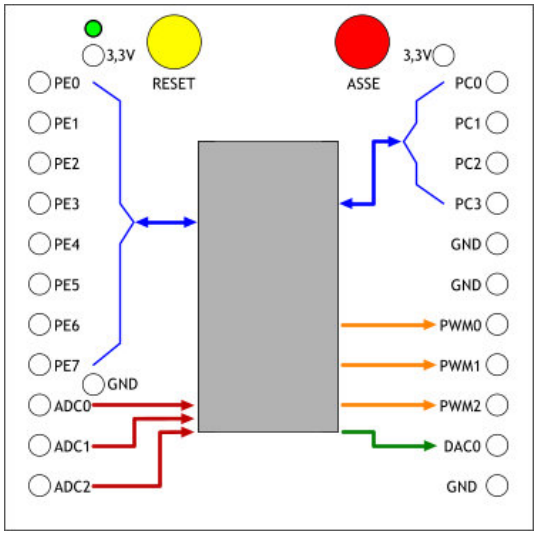
Man kan då välja att ansluta ett motstånd mellan ingången och 0V (jord, *ground*=GND) så att ingången ligger stabilt låg. Detta kallas *pull-down* (svensk term saknas).

Alternativt kan man ansluta motståndet till mikrokontrollerns matningsspänning för att lägga ingången stabilt hög. Då benämns det *pull-up*.

Er första uppgift blir att skriva ett program som läser av en tryckknapp och tänder en lysdiod då knappen trycks ner. Knapp och lysdiod kopplas till valfria GPIO-kontakter – **rita ett schema över er inkoppling nedan**.

Anslut knappen till pe0 – välj själva *pull-up* eller *pull-down* – och lysdioden till pc0.

Tips: Använd gärna funktionerna `move_cursor` och `dprintf` (de finns beskrivna i bilaga 1) för att skriva ut ingången pe0:s tillstånd på displayen, så att ni kan verifiera att ert program gör rätt.



När ert program fungerar delredovisar ni det för labassistenten.

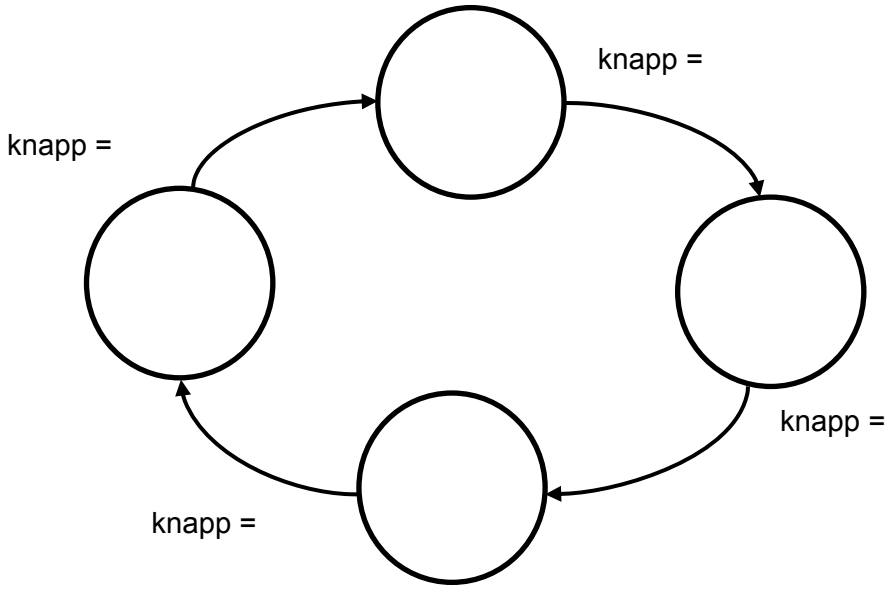
Assistentens signatur: _____

5.2 Moment 2 - Tillståndsmaskin

Ni skall nu programmera en tillståndsmaskin med tillstånden ”STOP”, ”START”, ”RUN” och ”BRAKE”.

- Tillståndsmaskinen skall starta i tillståndet ”STOP” vid reset.
- När en knapp **trycks** in skall man byta tillstånd från ”STOP” till ”START”.
- När knappen **släpps** ut i tillståndet ”START” skall man byta tillstånd till ”RUN”.
- När knappen **trycks** in i tillståndet ”RUN” skall man byta tillstånd till ”BRAKE”.
- När knappen **släpps** ut i tillståndet ”BRAKE” skall man byta tillstånd till ”STOP”.
- I ”STOP” skall ingen lysdiod lysa.
- I ”START” och ”BRAKE” skall endast en gul lysdiod lysa.
- I ”RUN” skall endast en grön lysdiod lysa.

Fyll i det påbörjade tillståndsdiagram nedan.



När ni är klara med fylleriövningen delredovisar ni den för labassistenten.

Assistentens signatur: _____

Lägg till en switch-case-funktion i ert program, så att det blir en tillståndsmaskin med dessa fyra tillstånd. Styr tillståndet med variabeln `state`, som finns fördeklarerad i programmet.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.3 Moment 3 – PWM-styrning

PWM (*Pulse Width Modulation*, pulsbreddsmodulation) används flitigt numera för att bl a varvtalsstyra motorer och variera ljusintensitet, t ex en mobiltelefons ljusstyrka.

I tillståndet ”RUN” i ert program lägger ni nu in funktionen `PWM0(x)`.

Denna gör så att utgången PWM0 matar ut en fyrkantvåg vars pulsbreddsförhållande (*duty cycle*) antar värdet x, där x kan varieras 0-100%, i heltalssteg. Detta betyder att fyrkantvågen är **hög** i x% av fyrkantvågens periodtid, och **låg** i 100-x%.

Sätt duty cycle till 20% i programmet, anslut **ScopeMetern** till utgången PWM0 och mät upp följande storheter:

PWM-signalens frekvens [HZ]. f : _____

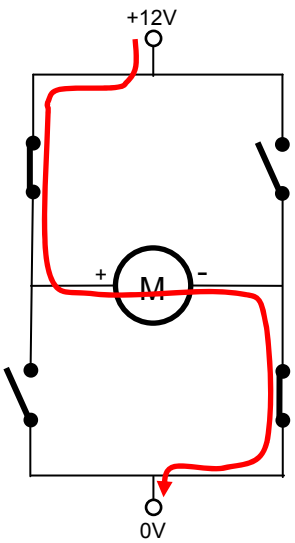
PWM-signalens till- och fråntid [s]. Till (hög): _____Från (låg):_____

Utspänningens topp- och medelvärde [V]. Û: _____ U:_____

Justera nu ert program så att duty cycle blir 50% i tillstånd ”STOP” och 100% i ”RUN”.

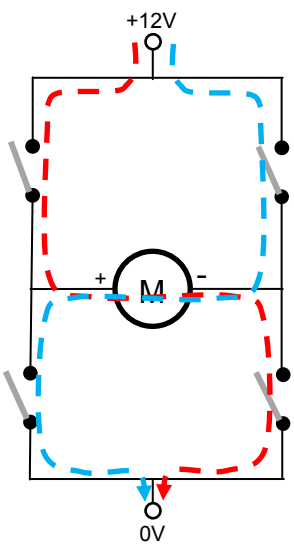
Anslut H-brygga med motorbänk till PWM0. H-bryggans schematiska funktion vid olika duty cycle beskrivs nedan.

100% duty cycle



Då duty cycle är 0% eller 100% är två diagonala transistorer (här ritade som omkopplare) till hela tiden. Motorn går då med full fart åt ena eller andra hållet.

50% duty cycle



Då duty cycle är 1%-99% turas de parkopplade diagonala transistorerna om att leda så att tiden de leder bestäms av duty cycle-värdet. Om det ”positiva” paret leder längre tid än det andra blir summaströmmen positiv och viceversa.

Då duty cycle är precis 50% är strömmen genom motorn lika lång tid positiv som negativ och medelströmmen blir därmed 0 => motorn står still.

H-bryggan behöver matas med 5V och 12V från spänningslådan på labbänken. Kör ert program och kontrollera att motorn går att starta och stoppa med tryckknappen.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.4 Moment 4 – AD-omvandling

Det finns tre AD-omvandlaringångar i MCU-lådan – ADC0, ADC1 samt ADC2. Dessa omvandlar en spänning på ingången till ett heltal.

Resultatet av en AD-omvandling läses med funktionen `GET_AD`, t ex `y=GET_AD(0)` som läser värdet för ADC0 och lagrar det till heltalet *y*. Läs på om funktionen `GET_AD` i bilaga 1 i peket och svara på dessa frågor:

Mellan vilka min- och maxvärden kan det AD-omvandlade resultatet variera?

Svar: Mellan _____ och _____

Mellan vilka min- och maxvärden får spänningen till AD-ingångarna variera?

Svar: Mellan _____ och _____

Vilken är den minsta spänningsändring vi kan detektera (1 bit ändras)?

Svar: _____

- Lägg till funktionen `GET_AD(0)` på lämpligt ställe i ert program.

  Använd `move_cursor` och `dprintf` för att skriva ut det AD-omvandlade värdet på displayen.

- Anslut vridpotentiometern på så sätt att ni kan variera spänningen på ingång ADC0 steglöst mellan 0 och 3,3 V.

  Kör programmet och verifiera att värdet på displayen varierar på rätt sätt.

- Utöka programmet så att det räknar ut vad AD-värdet motsvarar i inspänning (volt) och skriv ut även detta värde på displayen.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.5 Moment 5 – Steglös varvtalsreglering

- Modifiera tillståndet ”RUN”, så att motorns varvtal kan regleras från stillastående till full fart med vridpotentiometern.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.6 Moment 6 – Rampning av varvtal

Det är inte alltid önskvärt att en motor startar med ett ryck, så nu skall ni modifiera ert program så att motorvarvtalet rampas upp i tillståndet ”START” och ned i ”BRAKE”.

I tillståndet ”START” skall motorn startas från stillastående och öka duty cycle med takten 1%/100ms tills det når det värde som var inställt med potentiometern i startögonblicket.

Därefter skall programmet övergå i tillståndet ”RUN”, där man fortfarande skall kunna variera varvtalet med potentiometern.

Då man ska stoppa motorn, i tillståndet ”BRAKE”, skall duty cycle minska från aktuellt värde till stillastående med takten 1%/50ms.

Tips: Koppla in ScopeMetern till PWM0, så att ni har koll på duty cycle.

När ert program fungerar delredovisar ni det för labassistenten.

Assistentens signatur: _____

5.7 Moment 7 – Nya krav

Er kund kommer plötsligt på att det behövs ett nödstopp också, så att motorn stannar direkt om en annan (röd) knapp trycks in. Ha ett projektmöte i gruppen och diskutera om det är besvärligt eller ej, att lägga till denna funktion i ert program.

Markera er slutsats med ett kryss:

- ☐ Orka
- ☐ Suck
- ☐ Like A Boss

När ni gjort ert val redovisar ni det för labassistenten.

Assistentens signatur: _____

Städning:

- Radera mik-mappen i Program Workspace.
- Kör ”rensa.bat” i E:\ - då radderas alla bin-filer ni skapat.

Bilaga 1 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”.
||	Logisk ”eller”.
==	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 "mik.h"`	I filen mik.h finns alla deklarationssatser som knyter ihop våra ”arbetsnamn”, tex pe0 med mikrokontrollerns 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, t ex: 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(pc0);`
`}`
`else`
`{`
`  CLR_BIT(pc0);`
`}`
	Om elvis är mindre än 100 så sätts pc0 till 1. I annat fall sätts pc0 till 0.

`switch ( x )`
`{`
`  case 1 : SET_BIT(pc1);`
`  break;`
`  case 2 : CLR_BIT(pc1);`
`  break;`
`  case 3 : SET_BIT(pc2);`
`  break;`
`  case 4 : CLR_BIT(pc2);`
`  break;`
`  default : CLR_BIT(pc1);`
`  CLR_BIT(pc2);`
`  break;`
`}`
	En switch/case-sats fungerar så, att beroende på switch-variabelns värde (x i detta fall) så utförs **en** av case-satserna.
	Om *x == 3* kommer programmet att hoppa in vid *"case 3"* och sätta utgången *pc2* hög (1:a). Därefter hoppar programmet ut ur switch-satsen.
	Skulle *x* ha ett annat värde än 1-4 kommer programmet att gå till "default" där man kan agera på lämpligt sätt, t ex hantera felaktiga värden.

Några kommandon specifika för vår lab	
`init_mik();`	Initerar mikrokontrollern.
`init_pin(pc0,"out");`	Initierar pinne pc0 att vara en utgång.
`init_pin(pe0,"in");`	Initierar pinne pe0 att vara en ingång.
`SET_BIT(pc0);`	Pinne pc0 ettställs, dvs kommer att få värdet 1, vilket motsvarar nominellt 3,3 V ut.
`CLR_BIT(pc1);`	Pinne pc1 nollställs, dvs kommer att få värdet 0, vilket motsvarar 0 V ut.
`bit_in = GET_BIT(pe2);`	Pinne pe2 avläses och resultatet lagras i heltalsvariabeln bit_in.

`PWM0(20);`	PWM0(20) gör så att en fyrkantsvåg med *duty cycle* 20% matas ut på PWM-utgång 0. Funktionen accepterar heltalsvärden mellan 0 – 100.
`ADResultat = GET_AD(0);`	GET_AD omvandlar en likspänning på ingång ADC0 (i detta fall) till ett heltal som lagras i variabeln ADResultat. I vårt fall omvandlas en spänning i intervallet 0-3,3 V till ett tal mellan 0 och 1023.
`Delay(1000);`	Gör en paus i programmet på 1 sekund (1000 ms).
`clear_disp();`	Rensar displayen från all text.
`move_cursor(1,2);`	Flyttar markören till rad 1, kolumn 2. Displayen består av fyra rader och 20 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 (heltal). Om variabeln peng har värdet 180 kommer *Jag har 180 kr!* att skrivas ut på displayen.

190919/PK,HJ