Mikrokontroller styrteknik

Version 2016

Göran Hult

Innehållsförteckning. Sid

Kontrollerkretsen PIC 16F1827

1 Introduktion

Denna beskrivning av kontrollerkretsen PIC 16F1827 från Microchip är en kortfattad beskrivning avsedd för personer med grundläggande kunskaper inom mikrodatorteknik. Beskrivningens syfte är att snabbt ge nödvändig information för att komma igång med konstruktioner som innehåller en PIC-krets. Ett avsnitt i denna kompendiedel tar upp utvecklingsmiljön MPLAB IDE, MPLAB SIM samt C-kompilatorn MPLAB XC8. Datablad, MPLAB IDE (inklusive MPLAB SIM) samt C-kompilator finns att ladda ner från Microchips hemsida: www.microchip.com

2 PIC 16F1827:s uppbyggnad

Kretsen är uppbyggd som en 8-bitars RISC-processor med Harvardarkitektur för att erhålla korta enhetliga instruktionscykeltider T_{CYC}. De flesta instruktionerna utförs på en instruktionscykel, ($T_{CYC} = \frac{1}{F_{CYC}}$)

Instruktionsfrekvensen F_{CYC} erhålls genom att dividera klockoscillatorfrekvensen (F_{OSC}) med 4:

$$F_{CYC} = \frac{F_{OSC}}{4}$$

Maximala klockoscillatorfrekvensen F_{OSC} = 32 MHz ger instruktionscykeltiden 125 ns. I aktuellt projekt används F_{OSC} = 4 MHz vilket ger instruktionscykeltiden 1 µs.

Blockscheman för kretsen finns på de följande 2 sidorna samt på sidorna 10 och 16 i databladet. PIC 16F1827 innehåller de enheter som behövs för att fungera som ett litet men komplett mikrodatorsystem. Kretsen består av bland annat blocken i följande avsnitt där sidhänvisningar är till databladet för PIC 16F1827.

2.1 Centrala block

Klockoscillator sid 51-65
Man kan välja en intern oscillator med frekvens mellan 31.25 kHz och 32 MHz. Om man vill ha 32 MHz intern oscillator måste man använda den inbyggda 4xPLL:n (Phase Locked Loop) som multiplicerar frekvensen med 4. Man kan välja en oscillator där frekvensen bestäms externt med en kristall, en keramisk resonator eller en RC-krets. Det går att använda kristaller med frekvens upp till 20 MHz. Om man vill ha oscillatorfrekvens över 20 MHz med en yttre kristall måste man använda den inbyggda 4xPLL:n. Val av oscillatortyp görs med konfigurationsbitar sid 44-46. Val av oscillatorfrekvens vid intern oscillator görs i OSCCON sid 65.

Resetkretsar sid 73
Power-On Reset kan göras internt eller externt via MCLR-benet. Extern Reset är nödvändig om matningsspänningen till kretsen stiger för långsamt vid spänningspåslag.

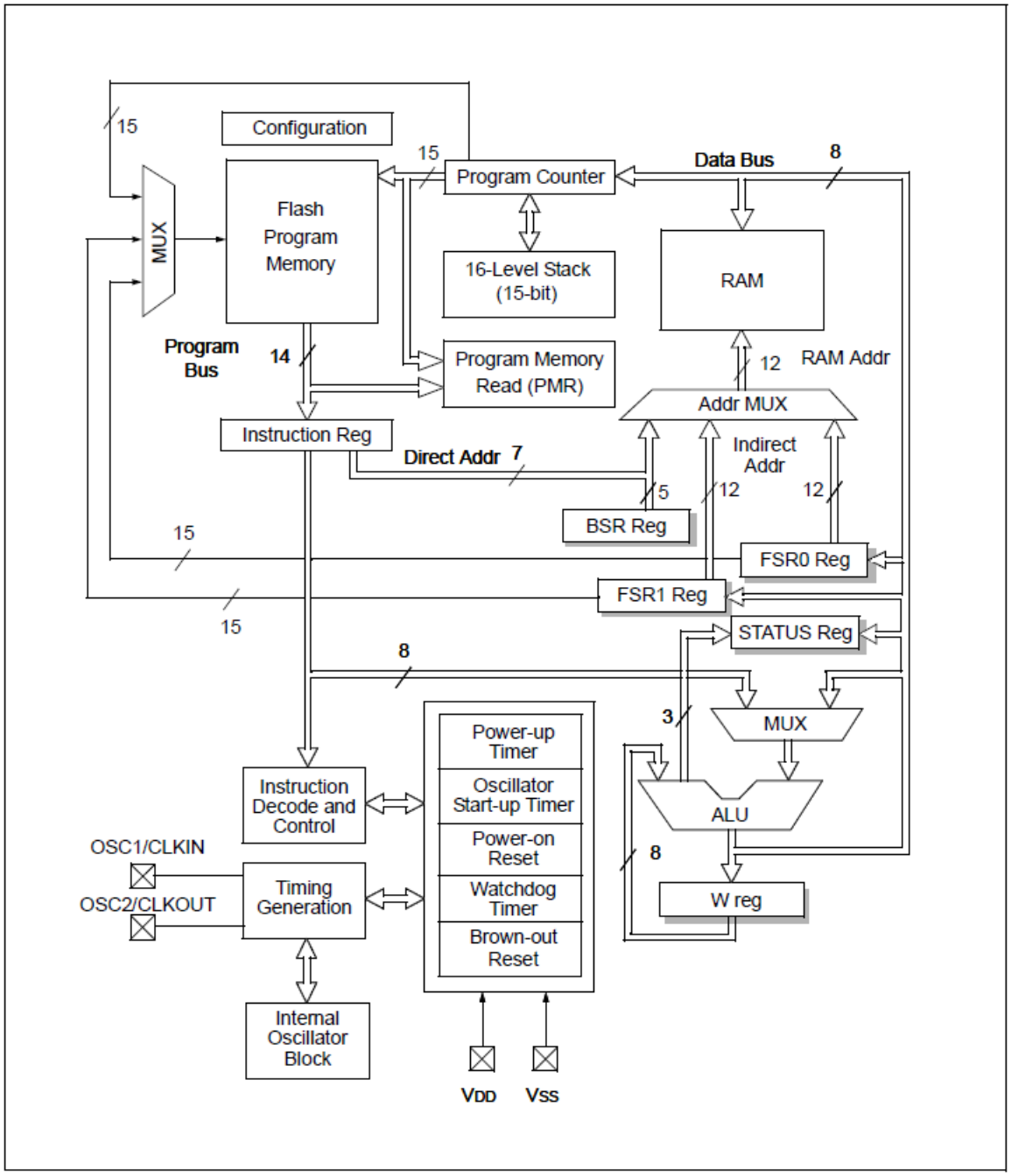
Power-up Timer ger en fördröjning på ca 65 ms innan kretsen lämnar Resettillståndet vilket ger matningsspänningen tid att stabilisera sig.

Watchdog Timer är en Timer som genererar Resetsignal om den inte nollställs med jämna mellanrum. Tiden mellan nödvändiga nollställningar av Watchdog Timern kan ställas mellan 1ms till 268 s.
Används för att processorn ska omstartas om programmet ”hänger sig”. Används normalt inte vid grundläggande utvecklingsarbete.

Brown-out Reset ger en Resetsignal om matningsspänningen är mindre än V_{BOR} under längre tid än ca 3 µs. V_{BOR} ställs in med konfigurationsbitar. Används för att processorn ska omstartas om man får en spänningsdipp på matningsspänningen som skulle kunna göra att programmet inte fungerar som avsett.

Konfigurering av Resetkretsar görs med konfigurationsbitar sid 44-46.

FIGURE 2-1: CORE BLOCK DIAGRAM



RAM, Dataminne, File Registers, sid 20-35
Registren är organiserade i 32 olika minnessidor, Bank 0 - Bank 31. Val av bank görs i **BSR**-registret. De 12 lägsta adresserna i respektive bank är s k Core Register vilka är samma i samtliga banker och innehåller ofta använda register. I varje bank finns därefter 20 stycken 8-bitars Special Function Registers (**SFR**) som används för att styra kretsens funktion. De högre adresserna i respektive bank är 8-bitars General Purpose Registers (**GPR**) eller RAM som används för tillfällig lagring av egna data. Totalt finns 384 byte RAM för egna data.

Adresserna 70h-7Fh är så kallat "Common RAM" som är åtkomligt direkt oberoende av i vilken bank man befinner sig i. Användandet av "Common RAM" ger snabbare åtkomst av data eftersom man inte behöver ändra bank i BSR-registret innan åtkomst.

BSR Reg Bank Select Register. Väljer Bank. Innehåller 5 MSB i adressen till RAM minnet

Då man använder ett högnivåspråk såsom C behöver man inte bry sig om val av bank eftersom kompilatorn sköter detta.

Programminne, Flash Program Memory sid 17-18
4K x 14 bitar organiserat i 2 sidor om vardera 2K. Val av sida görs i PCLATH-registret
Då man använder ett högnivåspråk såsom C behöver man inte bry sig om val av sida eftersom kompilatorn sköter detta.

Programräknare (15 bitar)

16-Level Stack 16 x 15 bitar där endast programräknarens värde kan lagras. Vid avancerade program kan detta utgöra en begränsning då man bara kan anropa subrutiner i maximalt 16 nivåer.

FSR0, FSR1 Reg File Select Register, register vid indirekt adressering.

ALU (8-bitars)

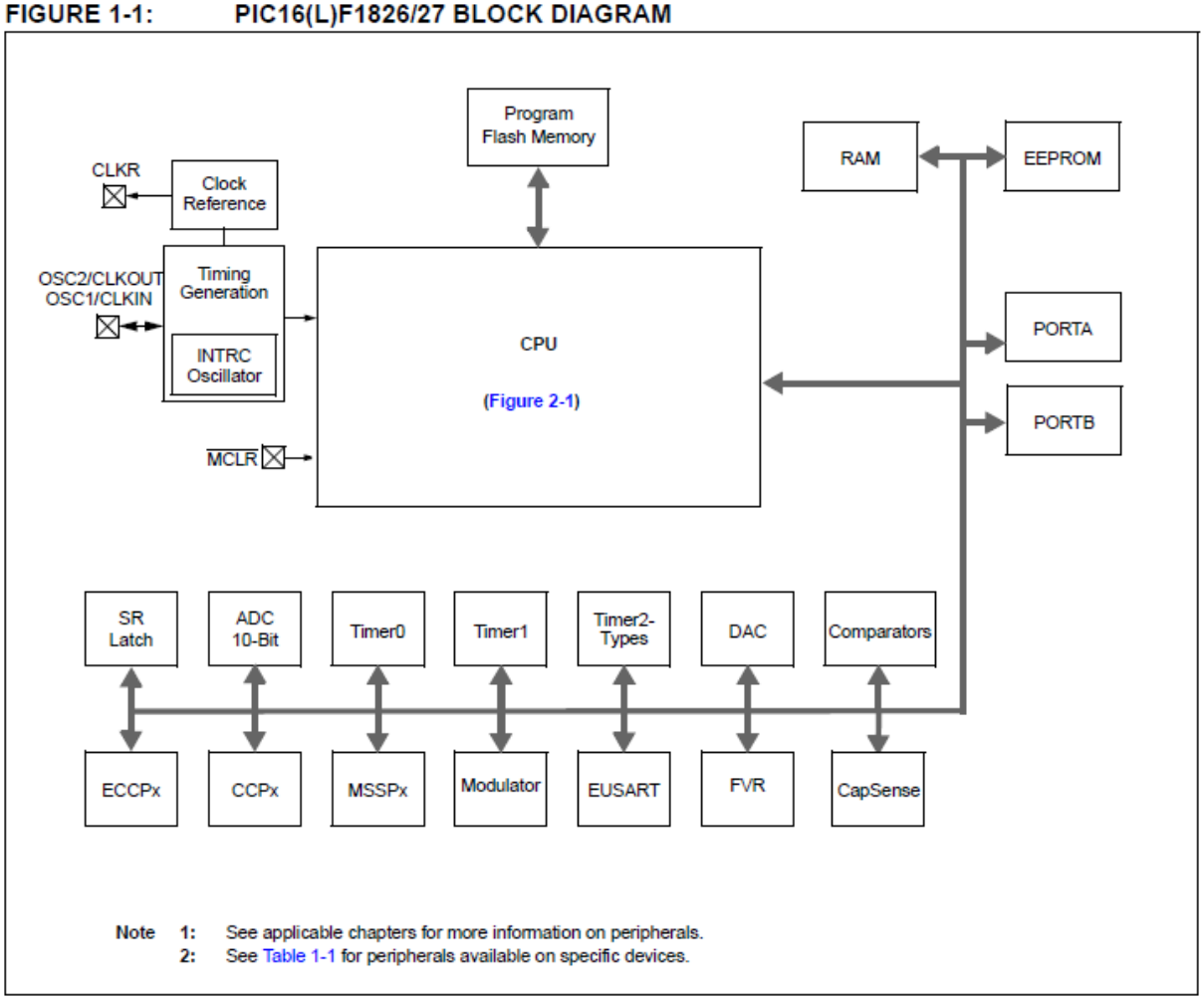
Arbetsregistret W-registret. Resultat och arbetsregister om 8 bitar. De flesta operationer görs via W-registret.

Statusregistret 8-bitars register med bl a Carryflagga, DigitCarryflagga, Zeroflagga

2.2 Periferienheter

Nedan visas ett blockschema över periferienheterna hos PIC 16F1827.
Vilka ben på PIC16F1827 som periferienheterna använder beskrivs i tabellen på sidan 6 i databladet.
Enheter som är aktuella i projektet beskrivs på följande sidor.

FIGURE 1-1: PIC16(L)F1826/27 BLOCK DIAGRAM



PORTA sid 117-124

PORTA är en port med maximalt 7 in/utgångar samt 1 ingång, RA5.
Maximalt antal in/utgångar erhålls om Intern Master Clear Reset och Intern klockoscillator används.
Fem av benen kan användas som analoga ingångar till AD-omvandlaren.
I ANSELA sid 123 väljs med ”0” om benen ska vara digitala och med ”1” om benen ska ha analogfunktion.
I TRISA sid 122 väljs med ”0” om benen ska vara utgångar och med ”1” om benen ska vara ingångar.
OBS Defaultinställning är analoga ingångar.

PORTB sid 125-129

PORTB är en port med 8 in/utgångar där 7 av benen kan användas som analoga ingångar till AD-omvandlaren.
I ANSELA sid 128 väljs med ”0” om benen ska vara digitala och med ”1” om benen ska ha analogfunktion.
I TRISB sid 127 väljs med ”0” om benen ska vara utgångar och med ”1” om benen ska vara ingångar.
OBS Defaultinställning är analoga ingångar.
Om man vill undvika yttre pull-up motstånd då man har strömställare anslutna till PORTB:s ingångar kan man aktivera "weak pull-up" i registret WPUB sid 128 samt OPTION_REG registrets WPUEN-bit sid 176.

Timer0 sid 173-176

8 - bitars räknare. Insignal från klockgenerator ($F_{OSC}/4$) vid timerfunktion. Pulser från yttre källa kan tas in via ben RA4/T0CKI och räknas i TMR0. Timer0 konfigureras med OPTION_REG sid 176 och CPSCON0 sid 318.

Timer1 sid 177-187

16 - bitars räknare. Insignal från intern klockgenerator (t ex $F_{OSC}/4$) eller yttre klockgenerator.
Timer1 konfigureras med T1CON sid 185 och T1GCON sid 186.

Timer2/4/6 sid 189-192

Timer2, Timer4 och Timer6 är 8–bitars räknare som får sina insignaler från klockgeneratorn ($F_{OSC}/4$) via var sin prescaler.
Timer2 konfigureras med T2CON sid 191.
Timer4 konfigureras med T4CON sid 191.
Timer6 konfigureras med T6CON sid 191.

PWM Pulse Width Modulation sid 203-227

Om man ska använda sig av en drivkrets med 2 transistorer i halvbrygga eller 4 transistorer i helbrygga så är det lämpligt att använda Enhanced Mode PWM.

Om man bara vill PWM-styra 1 transistor så är det enklare med Standard PWM sid 208-211.
Det finns 2 moduler för standard PWM, CCP3 respektive CCP4.
CCP3 konfigureras i CCP3CON och CCP4 konfigureras i CCP4CON sid 226

Till varje PWM-modul ska man koppla var sin 8-bitars Timer: Timer2, Timer4 eller Timer6.
Val av kopplad Timer görs i CCPTMRS sid 227.
Timer2 konfigureras med T2CON och PR2 sid 191.
Timer4 konfigureras med T4CON och PR4 sid 191.
Timer6 konfigureras med T6CON och PR6 sid 191.

Pulsbredden för CCP3 styrs med CCPR3L och CCP3CON <5:4>
Pulsbredden för CCP4 styrs med CCPR4L och CCP4CON <5:4>
Ben RA3/CCP3 respektive RA4/CCP4 används som PWM-utgångar.
Övrig konfigurering samt val av PWM-period mm sid 209.

AD-omvandlare sid 139-151

AD-omvandlaren är en 10-bitars successiv approximationsomvandlare och kan kopplas till 12 olika ben på kretsen (Kanal AN0-AN11).

ADC Clock Period, T_{AD}, erhålls vanligen genom en frekvensdelning av klockfrekvensen F_{OSC}. För korrekt AD-omvandling krävs att: 1,0 µs $\leq$ $T_{AD} \leq$ 9,0 µs.

Från det att en kanal valts krävs en viss tid (acquisition time, T_{ACQ}) innan man startar A/D-omvandlingen. Denna tid gör bland annat att omladdning av AD-omvandlarens sample & holdkondensator C_{HOLD} hinner ske. Minsta T_{ACQ} är ca 5 µs enligt databladet sid 149.

Som referensspänning till AD-omvandlaren kan man vid mindre kritiska applikationer använda matningsspänningen (V_{DD} resp V_{SS}).

Val av analog/digital görs i registren ANSELA sid 123 och ANSELB sid 128.
Konfigurering av AD-omvandlaren görs i ADCON0 och ADCON1 sid 145-146.
Resultatet av en AD-omvandling finns i registren ADRESH och ADRESL.

3 Instruktionsuppsättning och arbetssätt

3.1 Instruktionsuppsättning

En listning av samtliga 49 assemblerinstruktioner finns i databladet på sidan 327-328.
Instruktionsuppsättningen innehåller bl a instruktioner för att flytta data:
movlw, movwf, movf, movlb

Exempel:
```
movlw  k              ;Flyttar konstanten k till W
movwf  f              ;Flyttar innehållet i W till registret f
movf   f,W            ;Flyttar innehållet i registret f till W
movlb  k              ;Flyttar konstanten k till BSR-registret
```

Instruktioner finns för att addera, subtrahera, skifta samt utföra logiska operationer:
addwf, subwf, incf, decf, rlf, rrf, andwf, iorwf, xorwf

Exempel:
```
addwf  f,W            ;Addition W+f, resultatet placeras i W
addwf  f,F            ;Addition W+f, resultatet placeras i f
```

Bland bithanteringen återfinns:
bit clear (**bcf**), bit set (**bsf**)

Exempel:
```
bcf  f,0x2            ;Bit 2 i register f nollställs
bsf  f,0x0            ;Bit 0 i register f ettställs
```

För att skapa programstruktur finns bl a följande instruktioner:
goto label
bra label
call subrutinnamn resp **return** för återhopp
btfsc bit test f skip next instruction if clear.
btfss bit test f skip next instruction if set
decfsz decrement f skip next instruction if zero
incfsz increment f skip next instruction if zero

3.2 Adressering

Kretsen stödjer endast 3 adresseringsmoder, direkt, indirekt respektive relativ adressering. Indirekt adressering görs genom att ange en adress i indirekt adresseringspekare (FSR0 och FSR1) och sedan adressera register INDF0 och INDF1.

3.3 Avbrottshantering

Kretsen har en avbrottsvektor i programminnet (adress 0x0004) samt 23 avbrottskällor. Sid 81-94.
Bland annat finns följande avbrottskällor:

Externt avbrott via RB0/INT

Förändring PORTB bitar

Timeravbrott

AD-omvandling klar

Samtliga avbrott har samma prioritet och så fort ett avbrott hanteras så maskeras automatiskt alla andra avbrott tills återhopp från avbrottsrutinen sker. Maskering av alla avbrott sker genom nollställning av GIE-biten i INTCON. GIE-biten ettställs automatisk vid återhopp från avbrottsrutinen så att nya avbrott kan betjänas. Avbrottsflaggor måste nollställas i avbrottsrutinen innan återhopp från avbrottsrutinen sker.

Varje avbrottskälla kan maskas bort med enable-bitar i registren INTCON, PIE1, PIE2, PIE3 och PIE4.
Avbrottsflaggor finns i registren: INTCON, IOCBF, PIR1, PIR2, PIR3 och PIR4.

Då avbrott sker sparas följande register automatisk till så kallade skuggregister:
W, STATUS, BSR, FSR och PCLATH.
Vid återhopp från avbrottsrutinen återläses registren automatiskt.

En avbrottsrutin bör ha följande principiella uppbyggnad:

isr_rutin
Kontrollera vilken avbrottskälla som är aktuell
Hoppa till aktuell avbrottsrutin
Nollställ aktuell avbrottsflagga
Återhopp från avbrott

3.4 Read Modify Write problemet

PIC 16Fxx familjen använder en metod som kallas Read-Modify-Write (RMW) när man skriver till en enskild bit i ett register. Vid skrivning av enskild bit i t ex PORTB *läser* processorn först tillståndet på alla 8 benen i PORTB som sedan mellanlagras temporärt i ett internt register. Därefter *modifieras* den aktuella biten i det temporära registret. Slutligen *skrivs* det temporära registret till PORTB. Detta kan ge upphov till svårfunna fel om något ben i PORTB lastas så hårt att spänningen på benet inte motsvarar det önskade logiska tillståndet. I PIC 16F1827 har man infört registren LATA och LATB för att komma runt problemet. LATA och LATB är latcharna som är kopplade till respektive port. Om man skriver till en bit i LATB så läser processorn tillståndet i latchen i stället för tillståndet på benen vilket gör att man är oberoende av belastningen på benen.
Följande kan rekommenderas:

Vid skrivning till PORTA och PORTB rekommenderas användning av registren LATA och LATB.
Vid läsning av PORTA och PORTB måste man använda registren PORTA och PORTB.

4 C-kompilatorn MPLAB XC8

Olika C-kompilatorer kan ha olika storlek på sina datatyper så det är viktigt att ta reda på vad som gäller för den aktuella kompilatorn. I tabellen nedan syns storleken på datatyperna i MPLAB XC8.
Observera att variabeltyperna *bit* och *short long* är en avvikelse från ANSI-standarden.

Eftersom PIC 16F1827 är en 8-bitars processor ska man försöka att använda heltalstyperna *char, signed char* respektive *unsigned char* om det går. Om man väljer större heltalstyper där det inte behövs blir programmet onödigt stort och långsamt.

Typ	**Storlek (bitar)**
bit	1
char (1)	8
signed char	8
unsigned char	8
short	16
unsigned short	16
int	16
unsigned int	16
short long	24
unsigned short long	24
long	32
unsigned long	32
long long	32
unsigned long long	32
float (2)	24 eller 32
double (2)	24 eller 32
long double	Som double

Not (1): *char* är unsigned som standard.
(2): Storlek för *float o*ch *double* väljs under: **Project>Build Options>Project>Global**

I MPLAB XC8 finns följande makron för att erhålla fördröjningar (OBS! Två understreck i början):

```
__delay_us(x);         //Ger en fördröjning på x mikrosekunder, x heltal
__delay_ms(y);         //Ger en fördröjning på y millisekunder, y heltal
```

För att kunna använda makrona ovan krävs följande definition av aktuell klockoscillatorfrekvens:

```
#define _XTAL_FREQ 4000000      //Klockoscillatorfrekvens Fosc=4 MHz
```

Man kan infoga assemblerinstruktioner i sin C-kod, vilket man endast bör göra i undantagsfall.
Exempel:

```
asm("nop");           //Assemblerinstruktionen ”no operation”
```

5 Programutveckling för PIC 16F1827

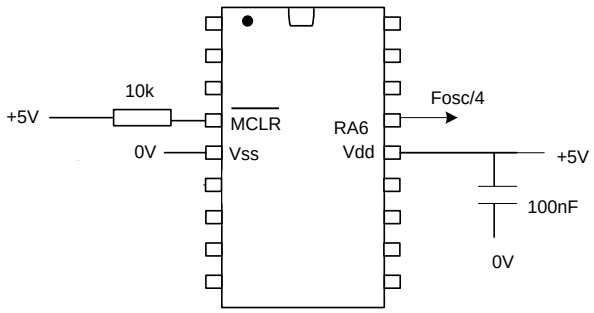
Vid utveckling av program till microcontrollers används ofta språket C.
Om man vill lära känna sin processor i detalj eller om det är tidskritiska delar i ett program är assembler att föredra. Nackdelen med assemblerprogrammering är att det tar längre tid att koda samt att felsökning är svårare.
I fallet med PIC-processorer har man också problemet med att hålla ordning på i vilken bank ett register finns i samt vilken som den aktuella sidan i programminnet. Detta håller kompilatorn reda på för oss om vi använder ett högnivåspråk.

5.1 Programskelett

Programskelett C-kod och minimal hårdvarukoppling

Kondensatorn på 100 nF (keramisk för låg impedans vid höga frekvenser) är en så kallad avkopplings-kondensator som ska placeras nära PIC-processorn med så korta anslutningsledningar som möjligt.
RA5/MCLR-ingången måste vara ansluten till +5V via ett motstånd på 10 kohm. Om man vill använda RA5 som ingång måste man ändra `MCLRE=ON` till `MCLRE=OFF` i koden nedan.
Notera att klockoscillatorfrekvensen delat med fyra, Fosc/4, finns tillgänglig på ben RA6. Om man vill använda RA6 som in/utgång måste man ändra `CLKOUTEN=ON` till `CLKOUTEN=OFF` i koden nedan.
Om man vill använda en yttre kristall som klockoscillator måste man ändra `FOSC=INTOSC` till `FOSC=LP`, `FOSC=XT` eller `FOSC=HS` beroende på frekvensen hos kristallen.

```
/* Programskelett MPLAB XC8 */

//Headerfiler-------------------------------------------------------------
#include <xc.h>

//Configuration Bits------------------------------------------------------
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
               WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Funktionsprototyper-----------------------------------------------------
void init(void);


//Huvudprogram------------------------------------------------------------
void main()
{
    init();
    while(1)
    {
        //PROGRAMKOD
    }
}


//Funktioner--------------------------------------------------------------
void init()
{
    //PROGRAMKOD
}
```

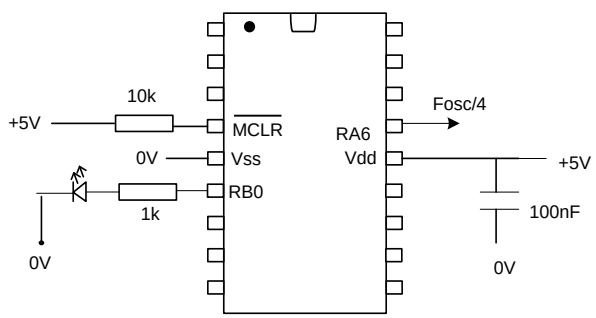


5.2 Programexempel

Programexempel 1

Nedan syns ett minimalt program för att tända en lysdiod på PORTB bit0 (RB0) och skriva talet 5 till PORTA. Notera hur man tilldelar värde till en enskild bit i ett register (LATBbits.LATB0) samt hur man kan skriva data på hexadecimalform som 0xNN, på binär form som 0bNNNNNNNN och på vanlig decimal form som NNN.

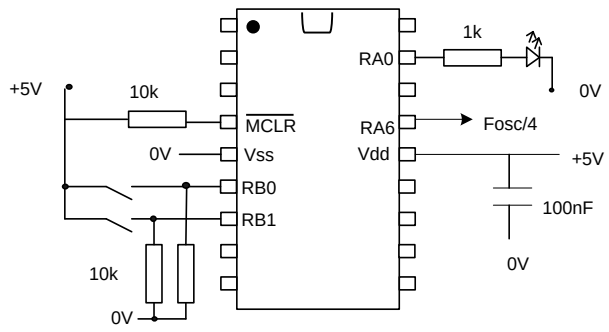
```
/* Programexempel 1 MPLAB XC8 */

//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
               WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2


//Funktionsprototyper-----------------------------------------------------
void init(void);

//Huvudprogram------------------------------------------------------------
void main()
{
    init();
    while(1)
    {
        LATBbits.LATB0=1;  //PORTB bit RB0 ettställs
        LATA=5;            //PORTA bit RA0 och bit RA2 ettställs
    }
}

//Funktioner--------------------------------------------------------------
void init()
{
    OSCCON=0b01101000;     //Intern klocka 4 MHz
    ANSELA=0b00000000;     //PORTA alla bitar digitala
    ANSELB=0b00000000;     //PORTB alla bitar digitala
    TRISA=0b00100000;      //PORTA bit 5 ingång resten utgångar
    TRISB=0x00;            //PORTB alla bitar utgångar
}
```



Programexempel 2

Följande program tänder en lysdiod på RA0 då de båda strömställarna är i till-läge, en AND-funktion.

```
/* Programexempel 2 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
        WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2


//Funktionsprototyper-----------------------------------------------------
void init(void);

//Huvudprogram------------------------------------------------------------
void main()
{
    init();
    while(1)          //Oändlig huvudloop
    {
      LATAbits.LATA0 = PORTBbits.RB0 && PORTBbits.RB1;
    }
}

//Funktioner--------------------------------------------------------------
void init()
{
    OSCCON=0b01101000;//Intern oscillator Fosc = 4 MHz
    LATA=0x00;        //Nollställer alla bitar i PORTA
    LATB=0x00;        //Nollställer alla bitar i PORTB
    ANSELA=0b00000000;//PORTA alla bitar digitala
    ANSELB=0b00000000;//PORTB alla bitar digitala
    TRISA=0b00100000; //PORTA bit 5 ingång resten utgångar
    TRISB=0b00000011; //PORTB bit0-1 ingångar, bit2-7 utgångar
}
```



Programexempel 3
Följande program initierar I/O-portarna. PORTA får 7 digitala utgångar och en ingång. PORTB får 8 digitala utgångar. Till PORTA bit 0 är en lysdiod ansluten till jord via en resistor vilket innebär att lysdioden tänds då RA0 går hög. Programmet tänder och släcker lysdioden med en fördröjningsrutin på ca 1 sekund mellan händelserna, vilket gör att lysdioden blinkar med frekvensen 0,5 Hz.

```
/* Programexempel 3 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
               WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Definitioner av konstanter----------------------------------------------
#define _XTAL_FREQ 4000000       //Klockoscillatorfrekvens Fosc=4 MHz
                                 //Krävs för macro    "__delay_us(x)"
                                 //respektive         "__delay_ms(x)"


//Funktionsprototyper-----------------------------------------------------
void init(void);


//Huvudprogram------------------------------------------------------------
void main()
{
    init();
    while(1)
    {
        LATAbits.LATA0=1;
        __delay_ms(1000);         //fördröj 1000ms
        LATAbits.LATA0=0;
        __delay_ms(1000);         //fördröj 1000ms
    }
}


//Funktioner--------------------------------------------------------------
void init()
{
    OSCCON=0b01101000;     //Intern klocka 4 MHz
    ANSELA=0b00000000;     //PORTA alla bitar digitala
    ANSELB=0b00000000;     //PORTB alla bitar digitala
    TRISA=0b00100000;      //PORTA bit 5 ingång resten utgångar
    TRISB=0x00;            //PORTB alla bitar utgångar
}
```

Programexempel 4
I detta program räknas pulserna som mottagits på T0CKI/RA4. Då 3*8=24 pulser mottagits blir RB2 hög under ett mycket kort ögonblick. Om man har problem med störningar kan man lägga en kondensator på 100 nF mellan RA4-ben och jord.

```
/* Programexempel 4 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
             WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Funktionsprototyper-----------------------------------------------------
void init(void);


//Huvudprogram------------------------------------------------------------------
void main()
{
  init();
  LATBbits.LATB2=0;
  TMR0=0;

  while(1)
  {
    if(TMR0==3)                 //Om 3*8=24 pulser mottagits
    {
      TMR0=0;
      LATBbits.LATB2=1;
      LATBbits.LATB2=0;
    }
  }
}


//Funktioner--------------------------------------------------------------------
void init()
{
  OSCCON=0b01101000;     //Intern klocka 4 MHz
  LATB=0;                //Nollställer alla bitar i PORTB
  ANSELA=0b00000000;     //PORTA alla bitar digitala
  ANSELB=0b00000000;     //PORTB alla bitar digitala
  TRISA=0b00110000;      //PORTA bit 4-5 ingångar resten utgångar
  TRISB=0x00;            //PORTB alla bitar utgångar
  OPTION_REG=0b00100010; //Initiera Timer0, Prescaler = 1:8,räknar på pos flank
  CPSCON0bits.T0XCS=0;   //Initiera Timer0 för pulsräkning på T0CKI/RA4
}
```

Programexempel 5

I detta program sker en AD-omvandling av den analoga spänningen på ingång RA0/AN0 till ett 8-bitars digitalt värde som läggs i variabeln AD_in. Efter val av AD-kanal finns en fördröjning, T_{ACQ} = 5µs. Denna tid gör bland annat att omladdning av AD-omvandlarens sample & holdkondensator C_{HOLD} hinner ske innan start av AD-omvandlingen. Placera ett 2,2 kΩ:s motstånd i serie med den analoga ingången för att skydda kretsen mot eventuella överspänningar. Om man har problem med störningar vid AD-omvandlingen kan man lägga en kondensator på 100 nF mellan AD-ingång och jord.

```
/* Programexempel 5 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
               WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Definitioner av konstanter----------------------------------------------
#define _XTAL_FREQ 4000000      //Klockoscillatorfrekvens Fosc=4 MHz
                                //Krävs för macro    "__delay_us(x)"
                                //respektive         "__delay_ms(x)"

//Funktionsprototyper-----------------------------------------------------
void init(void);
char AD_omv(char ADkanal);

//Huvudprogram------------------------------------------------------------
void main()
{
    char AD_in;
    init();
    while(1)               //Oändlig huvudloop
       AD_in=AD_omv(0);    //AD resultat från RA0/AN0 till AD_in
}

//Funktioner--------------------------------------------------------------
void init()
{
    OSCCON=0b01101000;     //Intern klocka 4 MHz
    TRISB=0x00;            //PORTB alla bitar utgångar
    ANSELB=0b00000000;     //PORTB alla bitar digitala
    ADCON1=0b01000000;     //Vänsterjusterat, AD-clock=Fosc/4,Vref VDD och VSS
    ANSELA=0b00000001;     //PortA bit 0 analog, övriga bitar digitala
    TRISA=0b00100001;      //PORTA bit0 och bit5 ingång, resten utgångar
    ADCON0=0x01;           //ADON
}

char AD_omv(char ADkanal)
{
    ADCON0=(ADCON0 & 0b10000011)|(ADkanal<<2);         //Val av AD-kanal
    __delay_us(5);         //Delay 5us. Macro i MPLAB XC8.
    ADCON0bits.GO=1;       //AD-omvandling startar
    while(ADCON0bits.GO);  //Vänta på att AD-omvandling är klar
    return ADRESH;         //Returnera 8 MSB av AD-omvandling
}
```

Programexempel 6
Följande program visar hur man kan erhålla en PWM-signal på ben RA3/CCP3.
Vid F_{OSC} = 4 MHz är f_{PWM} = 4 kHz.

```
/* Programexempel 6 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
               WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Funktionsprototyper-----------------------------------------------------
void init(void);

//Huvudprogram------------------------------------------------------------
void main()
{
    init();
    while(1)
    {
        CCPR3L=76;         //Duty-cycle=76/255=30%
    }
}


//Funktioner--------------------------------------------------------------
void init()
{
    OSCCON=0b01101000;     //Intern klocka 4 MHz
    ANSELA=0b00000000;     //PORTA alla bitar digitala
    ANSELB=0b00000000;     //PORTB alla bitar digitala
    TRISB=0;               //PORTB alla bitar utgångar
    TRISA=0b00100000;      //PORTA bit 5 ingång resten utgångar
    CCP3CON=0b00001100;    //CCP3 i PWM-mode
    CCPTMRS=0b01001010;    //CCP3 använder Timer2
    PR2=254;
    T2CON=0b00000100;      //Timer2 On, Prescaler=1 ger fPWM = 4 kHz
}
```

Programexempel 7

Följande visar hur avbrottshantering fungerar med MPLAB XC8. Programmet använder timeravbrott med Timer1 för att med 0.5 sekunders intervall ändra utgången RB0. På utgången RB0 kan man ansluta en lysdiod som kommer att blinka med frekvensen 1 Hz.

```
/* Programexempel 7 MPLAB XC8 */
//Headerfiler-------------------------------------------------------------
#include <xc.h>     //Definition av register och registerbitar mm.
                    //Se pic16f1827.h i kompilatorns include catalog
                    //C:\Program Files\Microchip\xc8\Vx.xx\include

//Configuration Bits------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
             WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=ON  //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF  //Config Word 2

//Funktionsprototyper-----------------------------------------------------
void init(void);
void interrupt isr(void);        //Nyckelordet interrupt ger interruptfunktion

//Huvudprogram------------------------------------------------------------
void main()
{
  init();
  while(1);          //Oändlig loop som väntar på avbrott
}

//Funktioner--------------------------------------------------------------
void init()
{
  OSCCON=0b01101000; //Intern klocka 4 MHz
  ANSELA=0b00000000; //PORTA och PORTB digitala I/O
  ANSELB=0b00000000;
  TRISB=0;           //PORTB utgångar
  TRISA=0b00100000;  //PORTA bit 5 ingång resten utgångar
  T1CON=0b00110001;  //Initiera Timer1.Prescale=8,Internal clock,Timer1 ON
  T1GCON=0b00000000;
  TMR1L=0xDF;        //Ställer TMR1 så att delay=0,5 sekunder
  TMR1H=0x0B;
  PIE1=0b00000001;   //TMR1 overflow interrupt enable
  INTCON=0b11000000; //Global och peripherial interrupt enable
}

//Interruptrutin
void interrupt isr(void)
{
  if(PIR1bits.TMR1IF && PIE1bits.TMR1IE)    //Vilken interrupt är aktuell
  {
    LATBbits.LATB0 = !LATBbits.LATB0;       //Togglar PORTB bit0
    TMR1L=0xDF;                             //Återställer TMR1
    TMR1H=0x0B;                             //så att delay=0,5 sekunder

    PIR1bits.TMR1IF=0;                      //Nollställer interruptflagga

  }
}
```

6 Utvecklingsmiljön MPLAB X IDE

6.1 Introduktion

Detta kapitel kan ses som en lathund för att snabbt komma igång med C-programmering i den aktuella utvecklingsmiljön MPLAB X IDE. Som vanligt kan det mesta annat hittas i manualer och ”hjälpfiler”.

MPLAB XC8 från Microchip är en C-kompilator för PIC-processorer i som huvudsakligen följer ANSI C90 standarden. Det rekommenderade sättet att använda kompilatorn är som ”plug-in” till MPLAB X IDE vilket ger användaren en välbekant windowsmiljö. Integrationen av kompilatorn i MPLAB X IDE gör också att det går smidigt att simulera sitt C-program i MPLAB SIM. Den installerade versionen av kompilatorn är en gratisversion vars huvudsakliga begränsning är att alla optimeringar ej går att aktivera. Det är även tillåtet att använda gratisversionen för kommersiella projekt. MPLAB X IDE och gratisversionen av MPLAB XC8 kan laddas ner utan kostnad från tillverkarens hemsida: www.microchip.com.

6.2 Öppna befintligt projekt

Starta MPLAB X IDE Välj: **File>Open Project** : zzzzzz.x

6.3 Skapa Nytt projekt

Starta MPLAB X IDE	Välj: **File > New Project**
Categories:	Microchip Embedded
Projects:	Standalone Project
	Next>
Family:	Mid-Range 8-bit MCUs (PIC10/12/16/MCP)
Device:	PIC16F1827
	Next>
Supported Debug Header:	None
	Next>
Hardware Tools:	Simulator
	Next>
Compiler Toolchains:	XC8
	Next>
Project Name:	zzzzzz (Använd ej mellanslag ääö etc)
Project Location:	Z:\yyyyy (Använd ej mellanslag ääö etc)
Kryssa i:	*Set as main project* och *Use project location as the project folder*
	Finish>

6.4 Inställningar för projektet

File>Project Properties

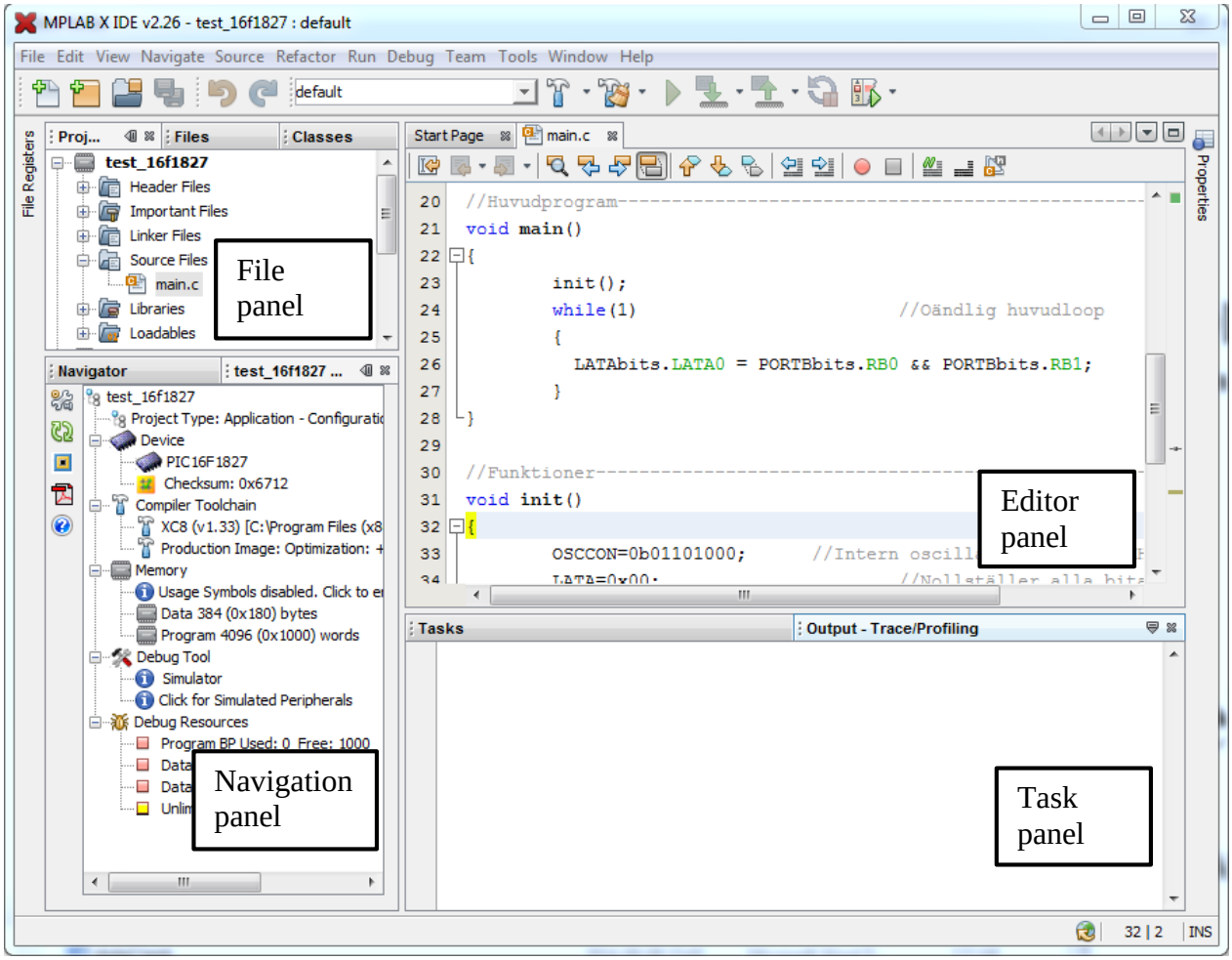
Categories**:** xc8 linker
Option categories: Memory model
Välj: *Size of Double: 32 bit* och *Size of Float, 32 bit*

Categories: Simulator
Välj Instruction Frequency: $F_{CYC} = \frac{F_{OSC}}{4} = 1\ MHz$

6.5 Skapa källkodsfil

I Project fönstret: Högerklicka på *Source Files*, Välj *New>Empty File..*
File Name: xxxxxx.c (Använd ej mellanslag åääö etc)
Finish

Skriv C-program, spara. Skrivbordet kan nu se ut som nedan:



File-panel: En panel med 3 alternativa fönster:
-Projektfönstret som visar projektträdet
-Filesfönstret som visar projektets filer
-Classfönstret som visar eventuella klasser i koden

Navigation-panel: En panel med 2 alternativa fönster:
-Navigatorfönstret visar information om valda filer samt symboler och variabler.
-Dashboardfönstret som visar information om processor, använt minne mm.

Editor-panel: För att studera och editera projektfiler. Startsida kan också visas här.

Task-panel: Visar resultat av kompilering eller simulering av program.

Om man dubbelklickar på filnamn i File-panelen så öppnas den i Editorpanelen

Om man högerklickar på en flik i ett fönster och väljer *Undock Window* så kopplas fönstret loss från panelen. För att återställa fönster till panelen, högerklicka på namn i fönster och välj *Dock Window*.

6.6 Kompilera för programmering av processor

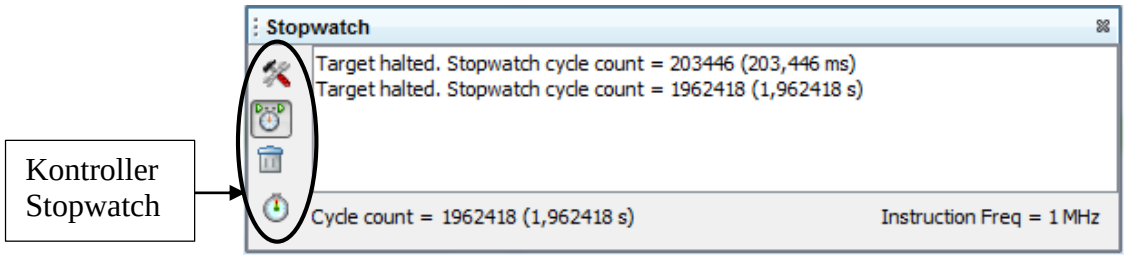
1. Kompilera hela projektet: **Run>Build Main Project**  .
   hex-fil skapas i katalogen: *projektkatalog\dist\default\production*.

2. I Output-fönstret längst ner fås resultatet av kompileringen. Man kan dubbelklicka på felmeddelandet i blå text, *Error*, för att komma till aktuell rad i källkodsfilen.

6.7 Simulera program

1. **Debug>Debug Main Project**  (Innefattar kompilering men skapar EJ hex-fil)

2. Studera tiden: **Window>Debugging>Stopwatch**.

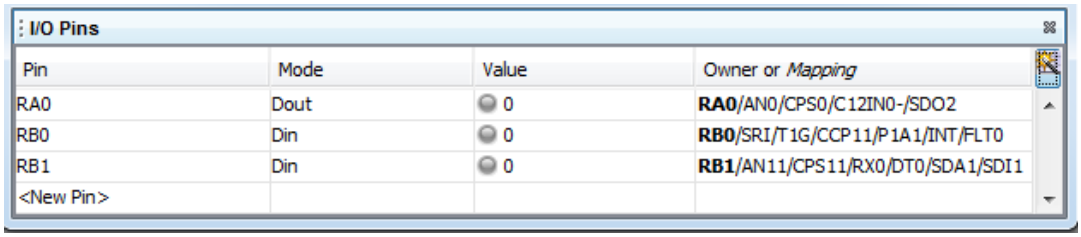


3. Studera SFR-register: **Window>PIC Memory Views>SFRs**
   I Task-panelen: Fliken SFR

4. Studera logiktillstånd på ett ben. **Window>Simulator>IOPin**
   I Task-panelen: Fliken I/O-Pins. Klicka på <New Pin>, välj ben.

   Avsluta eventuell simulering  . Starta simulering: **Debug>Debug Main Project** 

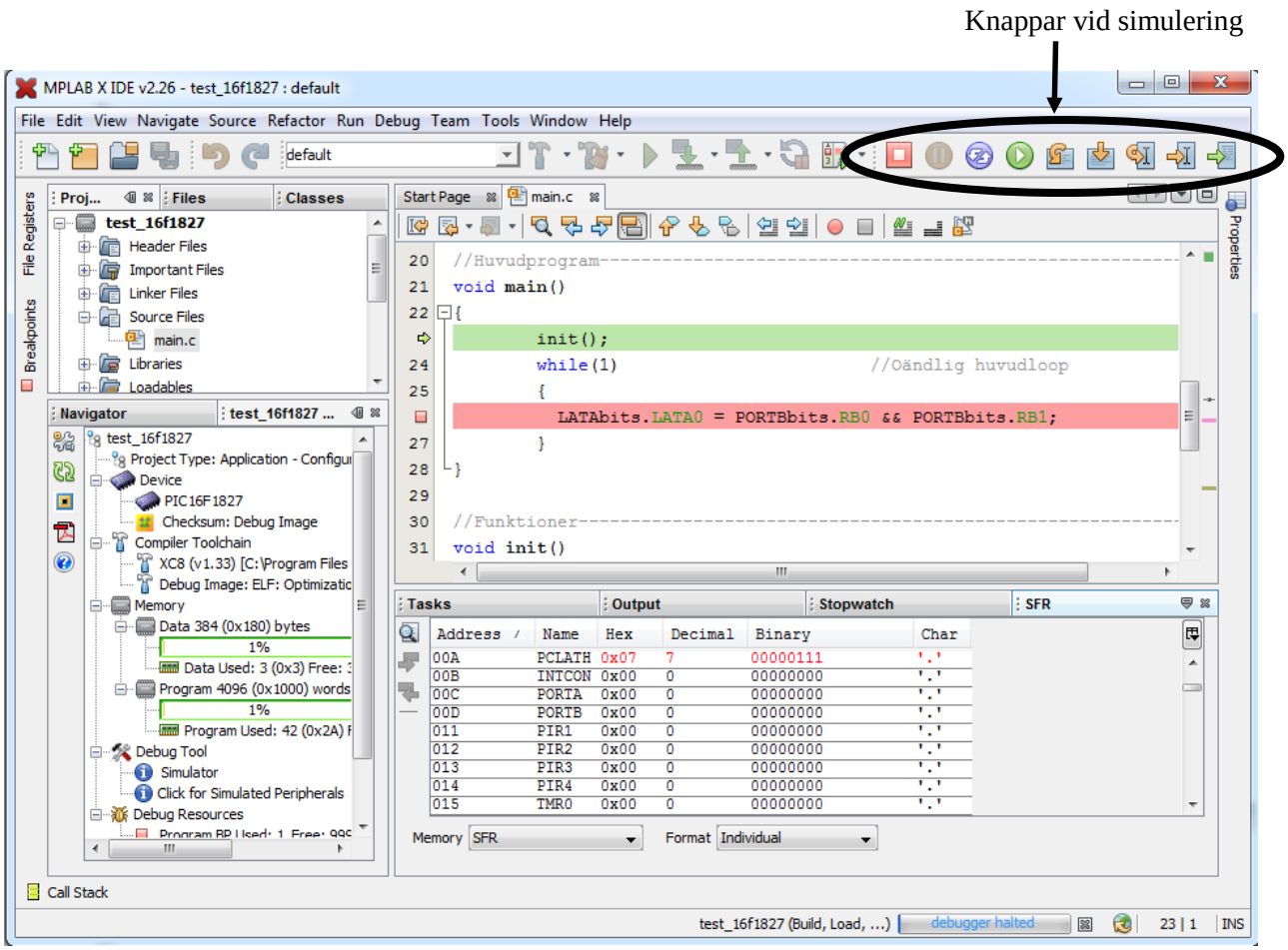
   Nedan studerar man de aktuella benen i exempel 2.



5. Studera Variabler/Register: **Window>Debugging>Watches**

   Högerklicka på Variabel/Register i källkodsfil. Välj New Watch. OK
   ELLER
   I Task-panelen: Fliken Watches.
   Högerklicka på <Enter new watch>, välj Variabel.
   OBS! Variabler/Register uppdateras bara då man pausar simuleringen

   I figuren på nästa syns ett programexempel på hur det kan se ut.
   Den gröna raden i källkoden motsvarar den rad som står i tur att utföras vid stegning av programmet.
   Den röda raden i källkoden är en rad med brytpunkt.
   De register/variabler som är markerade med röd färg i SFR-fönstret är register/variabler som har ändrats vid utförandet av den senaste raden.



Infoga/Radera brytpunkter genom att klicka på radnummer.

Observera att det måste finnas assemblerkod tillhörande den rad i C-koden som man har placerat en brytpunkt på för att simuleringen ska stoppa vid brytpunkten.

Bl a följande val finns för simulering:

1. Stegning rad för rad, stegar ej inne i funktioner: **Step Over** 
2. Stegning rad för rad, stegar även inne i funktioner: **Step Into** 
3. Simulering till nästa brytpunkt: **Continue** 
4. Pausa simulering. **Pause** 
5. Reset 
5. Avsluta simulering 

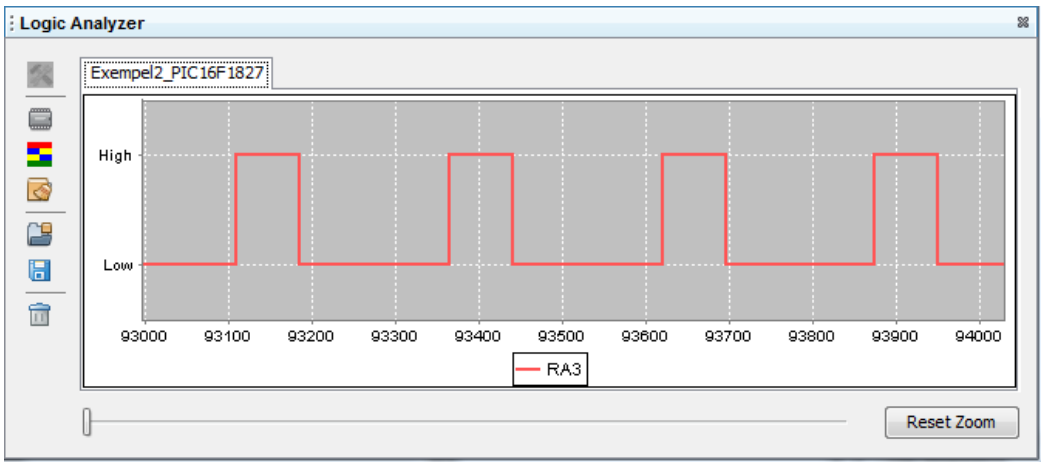
Om man under simuleringen vill ändra en variabel/registers innehåll:
Pausa simuleringen
Dubbelklicka på den aktuella variabeln/registrets värde (Hex/Decimal) i Watchfönstret eller SFRfönstret och ändra till önskat värde, avsluta med Retur.

Logikanalysator

Om man vill studera kurvformen för signaler på kretsens ben:
Window>Simulator>Analyzer

I Task-panelen: Fliken Logic Analyzer. Välj önskat ben med: 

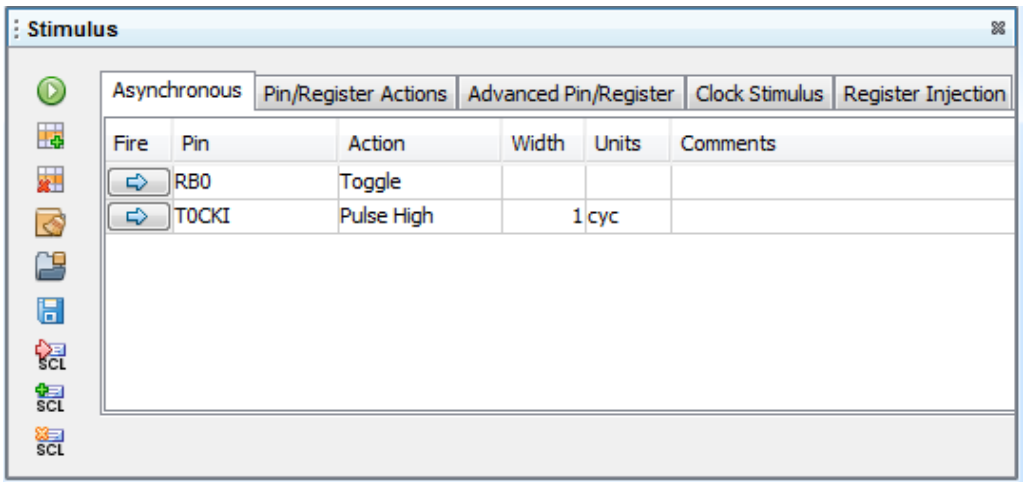
I figuren nedan har man valt att studera en PWM-signal på RA3/CCP3.



Asynkron Stimuli

Om man vid simuleringen asynkront (oberoende av tidpunkt) vill sätta en valfri ingång t ex hög eller låg gör följande:
Window>Simulator>Stimulus>
Välj fliken **Asynchronous**:
Välj **Pin** samt **Action**: Set High/Set Low/ Toggle/Pulse High/Pulse Low.

I figuren nedan fås en puls på T0CK1-ingången som är en instruktionscykel lång då man trycker på dess **Fire**-knapp  samt RB0 växlar tillstånd då man trycker på dess **Fire**-knapp 



Simulera analog inspänning

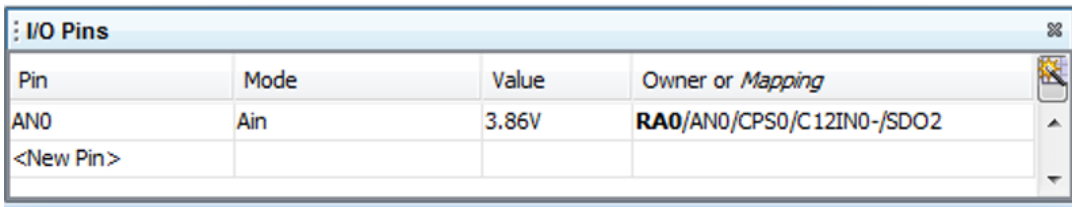
Om man vid simuleringen vill sätta inspänningen på ett analogt ingångsben gör följande:

Window>Simulator>IOPin
I Task-panelen: Fliken I/O-Pins. Klicka på <New Pin>, välj ben.
Skriv in den analoga inspänningen under ”Value”.

Avsluta eventuell simulering  . **Debug>Debug Main Project** 

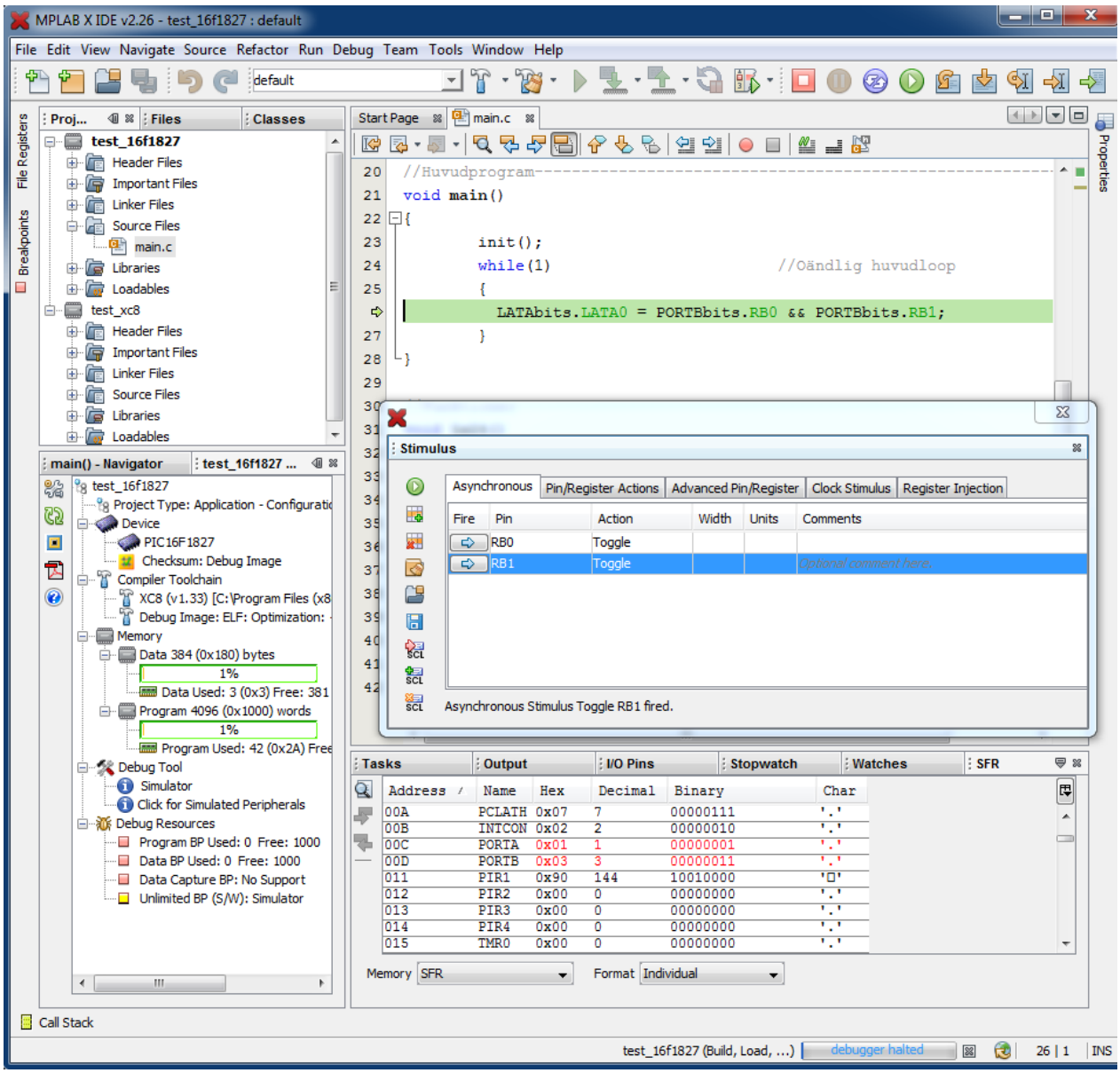
OBS!
Variabler och register som beror på den analoga inspänningen uppdateras bara då man pausar simuleringen

I figuren nedan har man simulerat inspänningen 3.86 V på ben AN0/RA0.



Exempel:

I nedanstående fönster har man simulerat programexempel 2. Genom att klicka på **Fire**-knapparna  i Stimulusfönstret kan man byta tillstånd på de 2 insignalerna på ben RB0 respektive RB1. Både insignaler på PORTB och utsignal på PORTA kan studeras i SFR-fönstret.



7. Bränna program till PIC-krets med programmeraren Dataman-40PRO.

1. Sök reda på HEX-filen med filnamnsändelse .hex som utvecklingsverktyget skapat i projektkatalogen: *\projektkatalog\dist\default\production*.
   Kopiera .hex filen till en USB-sticka. Notera eventuellt Checksum i MPLAB

2. Ladda ur eventuell statisk elektricitet som du är uppladdad med genom att hålla handen på den svarta ANTISTAT-mattan. Montera krets i brännarens sockel. Observera läge. Spänn fast. (Fäll ner ”arm”.)

3. Starta programmet Dataman-PRO Pg4uw om ej startat. Välj Dataman-40PRO.

4. Välj krets om ej vald. Alternativ 1: **Device>Select/default** 

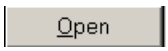
. Välj t ex PIC16F1827.

   Välj krets om ej vald. Alternativ 2: **Device>Select device** 

Search 16F1827
   Välj krets med notering*: Note: no adapter required*

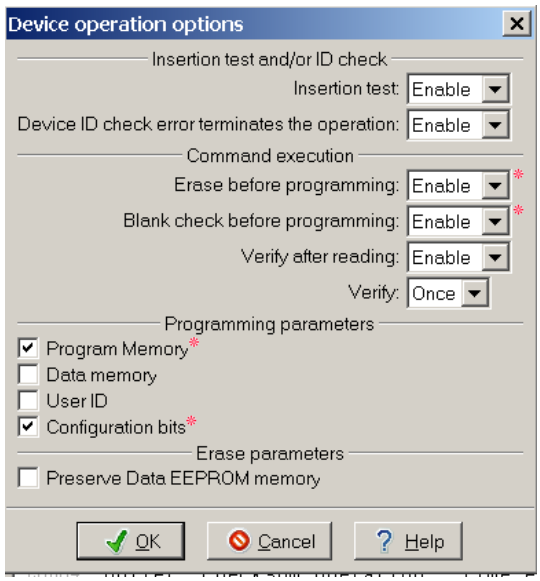
5. **File>Load** 

:
   Välj Files of type: *IntelHEX (*HEX)* Leta reda på din *zzz.hex*

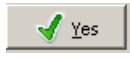
   Tryck Open 

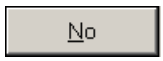
   Kontrollera eventuellt att ” PICmicro checksum” stämmer med Checksum i MPLAB

6. **Device>Device options>operation options** 



7. Programmera kretsen*:* **Device > Program** 



8 Vid frågeruta Repeat?: Tryck 

9. Ladda ur eventuell statisk elektricitet som du är uppladdad med genom att hålla handen på den svarta ANTISTAT-mattan. Plocka ur kretsen ur sockeln.