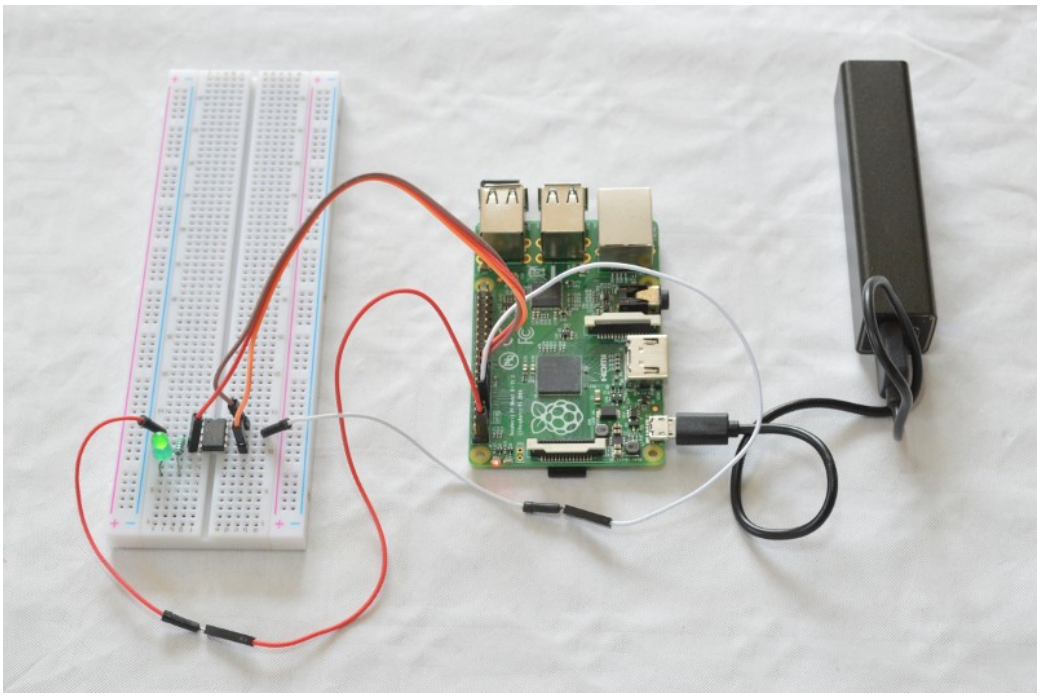
Test af
PIC 10F322



Oversigt: En grundig indføring i hvordan man kommer i kontakt med en PIC10F32x og hvordan man programmerer den i assambler fra en Raspberry Pi (RPi).

Der beskrives hvordan man gør en RPi til en SPI programmeringsenhed. Og hvordan man kan sende data ud af PIC10F32x til en RPi.

Done By: Henrik Kressner
Denne og andre kan findes på: https://synkro.dk/bog

Table of Contents

1. Intro

Dette er mine private notater omkring PIC 10F32x serien, alle er velkommen til at lade sig inspirere.

Jeg forventer et generelt kenskab til Raspberry Pi, UNIX CLI og C programmering.

Husk at læse manualerne (RTM), Der er to, en for hardware og en for software, hardware manualen er den vigtigste. Den kan findes ved at søge på: DS40001585B.

Det er hensigten at skrive kommentarer på dansk, men jeg holder mig mest til noget engelsk lignende inde i koden, fordi det ligesom ”rimer” med assambler nMonic.

December 2020: Mindre opdatering.

Maj 2023: Gennemgået for småfejl, udvidelse af kapitlet om timere, og tilføjelse af appendix A2.

Oktober 2023: Udviddet med kommentar om indirecte addressering.

Henrik Kressner
kressner@synkro.dk
April 2020

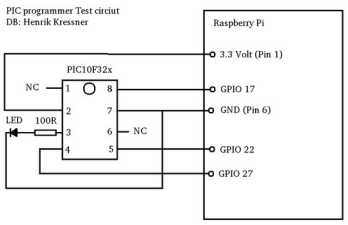
2. RPi som programmer

Selve progarmmeren består af 5 ledninger, et program skrevet i C (Se Appendiz A.1) og en Raspberry Pi (RPi) med nogle hjælpeprogrammer.

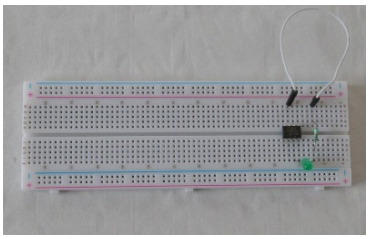
1 RPi
5 kabler
1 breadbord, eller what ever.
1 gpasm The GNU assambler for microchip controllers

Kildeteksten er vist i appendix A.1.

Bemærk: Jeg har benyttet en RPi version 2, hvis du bruger en anden udgave skal du tilpasse ofsettet til I/O ens registre som beskrevet i kildeteksten.



Diagram



Photo

Jeg har lavet et selvstændigt dokument der beskriver programmeren i detaljer:

https://www.synkro.dk/bog/PIC/PIC-programmer.pdf

3. Komme igang

3.1 port1.asm

HER SKAL INCLUDE FORKLARES !!

```
;       To compile this program:
;       # gpasm -a inhx8m port1.asm
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc


        clrf    PORTA
        bsf     PORTA,2         ; = PORTA,RA2
        movlw   B'00000011'     ; Port 2 is activ
        movwf   TRISA

        end
```

Dette program tænder en LED tilsluttet port2 (RA2), husk en faldmodstand.

Programmet starter med at cleare alle bits i port A, derefter sætter det bit 2 (husk det starter med bit nul) i Port A, så LED kan tænde.

De næste 2 linier:

```
        MOVLW   B'00000011'
        MOVWF   TRISA
```

Gør port 2 til output og port 0 , 1 og 3 til input. Port 3 er altid input, der kan ikke skrives til bit 3.

Programmet virker men … det kræver ekstern clock, ellers sker der intet. Se eksekverings instruks nedenfor.

I følge manualen burde controlleren kunne køre under en hertz, men jeg har ikke kunne få den til at køre stabilt under 1 Khz.

This program works, but it can not run as is, it need external clock on DATACLK. In my experience, the clock frequency must be up around 1 Khz or above to work.

10F322 defaulter til at kræve extern clock.

(RTM 2.2.3)

For at downloade programmet til PIC: `# cat  port1.hex | ./pic`

For at eksekvere programmet kan du enten tilslutte en extern clock, eller du kan bruge pic programmet med parameteren c: # ./pic c

3.2 port2.asm

```
;       gpasm -a inhx8m port2.asm
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc

        org             0x0000          ; Reset vector
        goto            start
        org             x0004           ; Interrupt vector
        goto            0x0000

start   clrf            PORTA
        decf            PORTA,1         ; Port 2 is output  ????
        movlw           B'00000011'
        movwf           TRISA           ; Port 0 and 1 is tristate
        goto            start

        end
```

Dette program gør det samme som port1.asm, men er lidt pænere

Vi starter med at initialisere reset vektoren og interrupt vektoren, der sender os til start hvis der kommer et interrupt.

Det gør der forhåbentlig ikke, for interrupt er deaktiveret som default. Men det er pænere, og hvis vi altid gør det, glemmer vi det ikke når vi skal bruge det.

Hvis vi undlader linien: `goto start` vil programmet stadig virke, men det kommer til at løbe gennem en masse NOP'er, før PC kommer tilbage til 0000.

Dette program kan ikke køre bare ved at blive downloaded, det kræver extern clock på DATACLK. (see port1.asm)

For at downloade programmet til PIC: `# cat  port2.hex | ./pic`

For at eksekvere programmet kan du enten tilslutte en extern clock, eller du kan bruge pic programmet med parameteren c: # ./pic c

3.3 Aktiver den interne clock

```
;       gpasm -a inhx8m port3.asm
;       Set port 2 ON, starting intern osc & WD timer off
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc

        ; Disable Watch Dog Timer (WDE)
        ; Enable internal clock
        __CONFIG _WDTE_OFF & _FOSC_INTOSC

        org     H'00'   ;Reset vector
        goto    run

        org     H'05'
run     movlw   B'00000100'
        movwf   PORTA           ; Port 2 is output
        movlw   B'11111011'
        movwf   TRISA           ; Port 0, 1 and 3 is tristate
        setup   run     ; ?????

        end
```

Dette program er næsten det samme som port2.asm. Nu kører vi bare med den interne clock.

Det sker i konfigurationslinien:
__ CONFIG …...

Det er 2 (to) understregninger før C i CONFIG, og det skal skrives med store bogstaver.

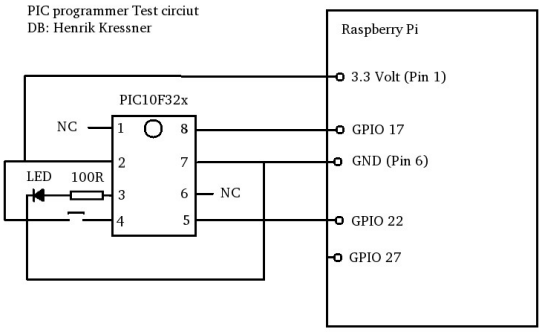
I konfigurationslinien disabler vi Watch Dog Timeren (WDT) og enabler den interne clock.

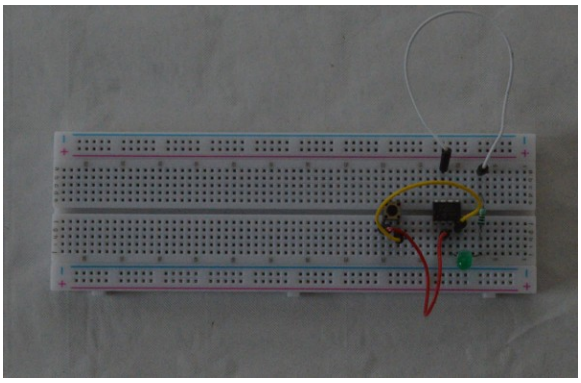
Clockfrekvensen rører vi ikke ved, den er som default 8 Mhz.

For at downloade og eksekvere PIC: `# cat  port3.hex | ./pic`

3.4 Input/Output

Dette program tænder en LED når der trykkes på en knap, når knappen slippes slukkes LED. Det er altså en lille øvelse i Input/Output (I/O).





```
;       gpasm -a inhx8m port4.asm
;       Port 2 = Port 1
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc

        __CONFIG _WDTE_OFF & _FOSC_INTOSC

        org             H'00'   ; Reset vector
        goto            init    ; Pass interrupt vector

        org     H'05'
init    clrf    ANSELA          ; All ports are digital I/O
        movlw   B'00000011'
        movwf   TRISA           ; Port 0 and 1 are input

run     btfsc   PORTA,          B'00000001'     ; Read RA1
        goto    on
        goto    off

off     bcf     PORTA, RA2
        goto    run

on      bsf     PORTA, RA2
        goto    run

        end
```

When the program run, it will test if bit 1 in PORTA is clear. If it is, the button is not pushed, and the program skip `goto on` and execute `goto off`.

If the button is pushed, bit 1 in PORTA is 1, this means we do not skip the next instruktion, so we go to on, and turn the LED on.

Der er fejl i min manual 10.2 PORTA, der tælles digitalt, IKKE som vist. (DS40001585B-page 68)

3.5 Blink1.asm

Et blinky progarm omskrevet fra en PIC16F624, derfor lidt rodet.

```
;       gpasm -a inhx8m blink1.asm
;       Port 2 ON/OFF
;       DB Henrik Kressner

processor       10F322
include         p10f322.inc

        __CONFIG _WDTE_OFF & _FOSC_INTOSC

;       Variables in ram
i       org     0x40    ; General purpose registers start at 0x40
j       org     0x41
k       org     0x42

        org     H'00'   ; Reset vector
        goto    setup

; Pass interrupt vector
                org     H'05'

setup   movlw   B'11111011'
        movwf   TRISA           ; Port is output

run     movlw   B'00000100'     ; Port 2 ON
        movwf   PORTA
        call    delay
        movlw   B'00000000'     ; Port 2 OFF
        movwf   PORTA
        call    delay
        goto    run

delay   clrf    i
        clrf    j
        movlw   0x08
        movwf   k
loop    decfsz  i,1
        goto    loop
        decfsz  j,1
        goto    loop
        decfsz  k,1
        goto    loop
        return

        end
```

This program starts by defining 3 variables (i, j and k), place them in the RAM, that start at 40H.

After initialisation, we turn the LED on, take a pause (calling delay), after that we turn the LED off, take a pause and return to run.

The delay is made by 3 loops in each other. We start by setting i = 0, so when we run `decfsz i,1`, the result is FFH, wich is not zero, therefore we call loop, until we reach zero.

When we reach zero, we go into the next loop and ask: `decfsz j,1`. This time j = FFH after the decrementation, so we jump to loop, and rund the first loop 256 times, before we try again, and so on.

The inner loop is where we fine tune the delay. Therefore we set k = 8H.

You can calculate the exact delay by counting clockpulses. With this setting, it should blink around ½ Hertz.

3.6 delay1.asm

A blinker with delay by timer0 **(Se timer kapitlet)**

```
;     gpasm -a inhx8m delay1.asm
;     Port 2 ON/OFF
;     DB Henrik Kressner

;     Delay with timer0

processor   10F322
include     p10f322.inc

      __CONFIG _WDTE_OFF & _FOSC_INTOSC

      org   H'00'             ; Reset vector
      goto  setup

      org   H'05'
setup clrf  ANSELA            ; All ports are digital I/O
      movlw B'11111011'
      movwf TRISA             ; Port 2 is output
      movlw B'00000111'       ; Fosc/4 and PSA and 1:256
      movwf OPTION_REG        ; Prescaler settings

run   bcf   PORTA, RA2        ; Port2 off
      call  delay
      bsf   PORTA, RA2        ; Port2 on
      call  delay
      goto  run

delay movlw 0x00
      movwf TMR0              ; Reset timer0
      bcf   INTCON, TMR0IF    ; Enable timer0
l1    btfss INTCON, TMR0IF    ; Timer overflow ?
      goto  l1
      return

      end
```

3.7 delay2.asm

Delay (forsinkelse) med variabel clocksetting..

```
;     gpasm -a inhx8m delay2.asm
;     Port 2 ON/OFF
;     DB Henrik Kressner

;     Adjustable delay with timer0

processor   10F322
include           p10f322.inc

      __CONFIG _WDTE_OFF & _FOSC_INTOSC

i     org   0x40  ; General purpose registers start at 0x40

      org   H'00'       ; Reset vector
      goto  setup

      org   H'05'
setup clrf    ANSELA    ; All ports are digital I/O
      movlw B'11111011'
      movwf TRISA       ; Port 2 is output
      movlw B'00000111' ; Fosc/4 and PSA and 1:256
      movwf OPTION_REG  ; Prescaler settings

run   bcf   PORTA, RA2  ; Port2 off
      call  delay2
      bsf   PORTA, RA2  ; Port2 on
      call  delay2
      goto  run

delay2      movlw 0xf
            movwf i
l2          call  delay
            decfsz i,1
            goto  l2
            return

delay clrf TMR0
      bcf     INTCON, TMR0IF  ; Enable timer0
l1    btfss   INTCON, TMR0IF  ; Timer overflow ?
      goto  l1
      return

      end
```

3.8 tx.asm

En enkel 250 Khz transmitter.

```
;       gpasm -a inhx8m tx.asm
;       Port 2 ON/OFF
;       DB Henrik Kressner

; Test PIC as Tx

processor       10F322
include         p10f322.inc

        __CONFIG _WDTE_OFF & _FOSC_INTOSC

;       Variables in ram
i       org     0x40    ; General purpose registers start at 0x40
j       org     0x41
k       org     0x42

        org     H'00'   ; Reset vector
        goto    setup

; Pass interrupt vector
                org     H'05'

setup   movlw   B'11111011'
        movwf   TRISA           ; Port is output

run     movlw   B'00000100'     ; Port 2 ON
        movwf   PORTA
        nop                     ; goto is 2 cycle, therefore 2 nop
        nop
        movlw   B'00000000'     ; Port 2 OFF
        movwf   PORTA
        goto    run


        end
```

Dette program lægger en firkant på port 2 med en frekvens tøt på 250 Khz.

Dutycycle = 50% opnås ved at lægge et par NOP ind, så antalet af clockcykler passer for on og off.

4 Watch dog timer

En wathcdog timer (WDT) er en funktion der sikrer dit program får kontrol over CPU'en, selv om programmet skulle "hænge", altså hvis det går ind i et uendeligt loop.

Kontrolleren er default sat op med watchdog timer on, med et interval på 2 sekunder. Det betyder, hvis dit program ikke oplyser det er i live i 2 sekunder, så kommer vagthunden og tager over, det kan vi bruge til at lave en blinker.

Bemærk: I dette program mangler _WDTE_OFF i configurationslinien, det er for at aktivere WDT

4.1 wdt1.asm

```
;       # gpasm -a inhx8m wdt1.asm
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc

        ; We do not need to activate the watch dog timer
        ; it is default active
        __CONFIG _FOSC_INTOSC

        org     0x0000  ; Reset vector
        goto    init
        org     0x0004  ; Interrupt vector
        goto    init

        org     H'05'
init    clrf    ANSELA          ; All porte er digital I/O
        movlw   B'00000011'
        movwf   TRISA           ; Port 2 er output
        btfsc   PORTA, RA2      ; Hvad er der på port 2
        goto    off             ; if LED is on, goto off
        goto    on              ; if LED is off, goto on

off     bcf     PORTA, RA2
        goto    run

on      bsf     PORTA, RA2

run     goto    run             ; forever

        end
```

Når watch dog timer (WDT), bliver trigged sætter den PC til 0x0000 og lader programmet kører videre.

Vi kan lave en blinker ved at vente på WDT "slår til", så tjekker vi om RA2 er ON, hvis den er slukker vi den, ellers det modsatte.

Fordi WDT default "slår tiol" med ca 2 sekunders interval, opnår vi en blink funktion.

Frekvensen på WDT reguleres i WDTPS bitten i WDTCON registeret. Det er variabelt fra ca. 1 mS til 256 Sekund. (RTM 8.6)

5. Interrupt

Lad os lave et program der manipulerer med en LED på Port2 (RA2). Når knap 1 bliver trykket vil LED gå off, hvis den var on, ellers vil den gå on. Egentlig bare en manual flip flip, eller 2-deler.

5.1 interrupt1.asm

```
;       # gpasm -a inhx8m interrupt1.asm
;       DB Henrik Kressner
processor       10F322
include         p10f322.inc

        __CONFIG _WDTE_OFF & _FOSC_INTOSC

        org             0x0000  ; Reset vector
        goto            init
        org             0x0004  ; Interrupt vector
        goto            int

        org             H'05'
init    clrf            ANSELA          ; All ports are digital I/O
        movlw           B'00000011'     ; Port 0 and 1 is input
        movwf           TRISA
        bsf             INTCON, GIE     ; Enable global interrupt
        bsf             INTCON, IOCIE ; Enable interrupt on change
        bsf             IOCAP, IOCAP1 ; Enable interrupt on port 1
        bsf             PORTA, RA2      ; LED ON

run     goto            run

int     btfsc           PORTA, RA2
        goto            off
        goto            on

off     bcf             PORTA, RA2
        movlw           0x00
        movwf           IOCAF           ; Clear interrupt flags
        retfie

on      bsf             PORTA, RA2
        movlw           0x00
        movwf           IOCAF           ; Clear interrupt flags
        retfie

        end
```

This is not good interrupt practice.

Normaly we would start our interrupt rutine by detecting what cased the interrupt. But we are about to learn, and there is only one interrupt activated, so we can accept it. (KIS)

The 2 lines:

```
        org             0x0004
        goto            int
```

Tell that the interrupt is placed in the function called int.

After initialisation, we go into a loop that does nothing.

When a interrupt happen, it must be activety on port 1, the program will then run from int, here we test port 2, if the LED is on we want to turn it off, therfore we call the function off if that bit is not set, and call on if the bit is set.

The line:

```
movlw           0x00
movwf           IOCAF
```

Is importent, here we ckear the interrupt flag. If we do not clear the interrupt flag, the interrupt will repeat until we do.

(MÅ KUNNE LAVES PÅ EN LINIE)

clrf IOCAF ???

6 Analog til digital konverter

PIC 10F3xx har en 8 bit sample and hold analog til digital konverter (ADC), der kan benytte 3 eksterne og 2 interne porte som input.

De eksterne porte er RA0, RA1 og/eller RA2, de interne er en temperaturindikator, eller en intern konstant spænding. (FVR)

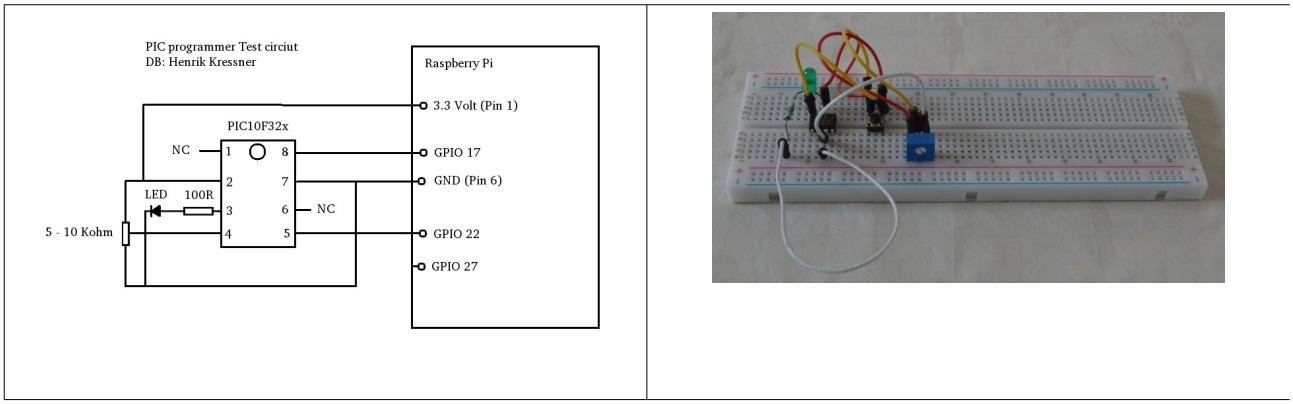
Da sample and hold kræver et stykke tid til at foretage en måling, er der af 3 måder man kan benytte ADC.

1. Overhold tidskravet beskrevet i manualen, ved at have en delay funktion.

2. Poll XXX registeret for at se om data er klar.

3. Interrupt når data er klar.

Lad os lave en program hvor outputfrekvensen på RA2 er en funktion der afhænger af spændingen på indgangen til A/D konverteren. Jo højere spænding jo høkere frekvens. Programet er adc1.asm.



På billedet er der en trykknap, men den er ude af funktyion i dette eksempel.

6.1 adc1.asm

```
;           gpasm -a inhx8m adc1.asm
;           Analog to Digital conversion
;           DB Henrik Kressner

processor  10F322
include             p10f322.inc

                    __CONFIG _WDTE_OFF & _FOSC_INTOSC

;                   Variables in ram
i           org     0x40        ; General purpose registers start at 0x40
Result      org     0x41
k           org     0x42

            org     H'00'  ; Reset vector
            goto    setup

; Pass interrupt vector
            org     H'05'

setup       movlw   B'11111011'         ; Port 0 & 1 input Port 2 = output.  (Do NOT use RAx, has dobble meaning)
            movwf   TRISA
            movlw   B'00000010'
            movwf   ANSELA              ; Port 1 = analog input
            movlw   B'10100101'         ; Fosc/16, port 1 analog in,  GO/Done off, ADON enable
            movwf   ADCON
            clrf    i                   ; Delay until Go/Done steady
l1          decfsz  i,1
            goto    l1

run         bsf     ADCON,GO_NOT_DONE       ; Start A/D convert
            movlw   B'00000100'
            movwf   PORTA                   ; Port 2 ON
            call    delay
            movfw   Result
            movlw   0xff
            movwf   k
            movlw   B'00000000'
            movwf   PORTA                   ; Port 2 OFF
            call    delay
l2          btfsc   ADCON,GO_NOT_DONE       ; Wait until result ready
            goto    l2
            movfw   ADRES                   ; Get result -> register W
            movwf   Result
            goto    run


delay       clrf    i
            movfw   Result
            movwf   k
l3          call    l4
            decfsz  k,1
            goto    l3
            return

l4          nop
            nop
            nop
            nop
            nop
            decfsz  i,1
            goto    l4
            return

            end
```

6.2 adc2.asm

Analog til digital konvertering med interrupt. Programmet fungerer som en komporator. HvisVin er større end ½ VCC går LED LED on, ellers er den off. LED er på RA2.

```
;       gpasm -a inhx8m adc2.asm
;       Analog to Digital conversion
;       DB Henrik Kressner

processor      10F322
include        p10f322.inc

       __CONFIG _WDTE_OFF & _FOSC_INTOSC

LIMIT  org     0x40
i      org     0x41

       org     0x00
       goto    init

       org     0x04
       goto    int

       org     0x05
init   movlw   0x80
       movwf   LIMIT          ; Set treshold
       movlw   B'00000010'
       movwf   TRISA          ; Port 1 = input
       movlw    B'00000010'
       movwf   ANSELA         ; Port 1 = analog input
       movlw   B'10100101'    ; Fosc/16, port AN1 analog in, GO/Done off and ADON enable
       movwf   ADCON
       bcf     PIR1, ADIF     ; Skal cleares efter interrupt
       bsf     PIE1, ADIE     ; Enable ADC Interrupt
       bsf     INTCON, PEIE   ; Enable Peripheral interrupt
       bsf     INTCON, GIE    ; Enable global interrupt

       clrf    i       ; Delay until Go/Done steady
l1     decfsz  i,1
       goto    l1

       bsf     ADCON,GO_NOT_DONE      ; Start A/D convert

start  goto    start

int    movfw   ADRES          ; Get A/D Result
       subwf   LIMIT,w
       btfsc   STATUS,C       ; result > LIMIT ?
       goto    off
       bsf     PORTA, RA2     ; LED = On
       goto    out

off    bcf      PORTA, RA2

out    bcf      PIR1, ADIF            ; Skal cleares efter interrupt
       bsf      ADCON,GO_NOT_DONE     ; Start A/D converter
       RETFIE

       end
```

7 Timers

7.1 timer0.asm

En traditionel blinker (blinky) med timer gør programmet lidt nemmere end blink1.asm.

```
;     gpasm -a inhx8m timer0.asm
;     Port 2 ON/OFF
;     DB Henrik Kressner


processor   10F322
include     p10f322.inc

      __CONFIG _WDTE_OFF & _FOSC_INTOSC

      org   H'00' ; Reset vector
      goto  setup
      org   0x04  ; Interrupt vector
      goto  intr

      org   H'05'
setup clrf    ANSELA    ; All ports are digital I/O
      movlw B'11111011'
      movwf TRISA       ; Port 2 is output
      movlw B'00000111' ; Fosc/4 and PSA on and 1:256
      movwf OPTION_REG  ; Prescaler settings
      movlw B'10100000' ; GEI and TMR0IE
      movwf INTCON

run   goto  run

intr  bcf   INTCON, TMR0IF
;     movlw 0xf0
;     movwf TMR0
      btfsc PORTA, RA2  ; Er port 2 off ?
      goto  off
      goto  on

off   bcf   PORTA, RA2
      RETFIE

on    bsf   PORTA, RA2
      RETFIE

      end
```

7.2 timer2.asm

Timer2 er egentlig 3 tællere: En prescaler, selve tælleren, og en postscaler. Prescaleren kan dele 1:1, 1:4, 1:16 og 1:64, tælleren har 8 bit, så den kan dele op til og med 256, postscaleren kan dele med op til 16. Den maximale deling er altså 64 * 256 * 16 = 266.240.

Timer2 kan vist kun køre på interrupt, (SKAL TJEKKES) lad os udnytte det til at lave en blinky. Ved overrun får vi et interrupt.

Tælleren styres i dette tilfælde af oscilatoren, der som standard svinger på 8 Mhz, og input til tæller er oscilatorfrekvensen/4.

```
;     gpasm -a inhx8m timer2.asm
;     Port 2 ON/OFF
;     DB Henrik Kressner

processor       10F322
include         p10f322.inc

        __CONFIG _WDTE_OFF & _FOSC_INTOSC

        org     H'00'   ; Reset vector
        goto    setup
        org     0x04    ; Interrupt vector
        goto    intr

        org     H'05'
setup   clrf    ANSELA
        movlw   B'11111011'
        movwf   TRISA         ; Port 2 er output
        movlw   B'11111111'   ; 1:16, timer2 on og 1:64
        movwf   T2CON         ; Prescaler settings
        movlw   B'11000000    ; GIE og Peripheral Interrupt Enable
        movwf   INTCON
        bsf     PIE1, TMR2IE  ; Interrupt
        bcf     PIR1, TMR2IF  ; Enable timer2 interrupt

run     goto    run

intr    bcf     PIR1, TMR2IF    ; Enable timer2
        btfsc   PORTA, RA2      ; Er port 2 off ?
        goto    off
        goto    on

off     bcf     PORTA, RA2
        RETFIE

on      bsf     PORTA, RA2
```

```
        RETFIE

        end
```

7.3 timer2a.asm

Samme som timer2.asm, men med en længere periodetid, opnået ved at ændrer CPU'ens clock frekvens fra 8 Mhz (default) til 1 Mhz.

```
;     gpasm -a inhx8m timer2a.asm
;     Samme som timer2, men med længere periodetid skabt af lavere clkfrq
;     DB Henrik Kressner


processor   10F322
include     p10f322.inc

      __CONFIG _WDTE_OFF & _FOSC_INTOSC

      org   H'00'       ; Reset vector
      goto  setup
      org   0x04        ; Interrupt vector
      goto  intr

      org   H'05'
setup movlw B'00110000'       ; Clk Frq = 1 Mhz
      movwf OSCCON
      clrf  ANSELA          ; Ports 2 er digital I/O
      movlw B'11111011'
      movwf TRISA             ; Port 2 is output
      movlw B'11111111'       ; 1:16, timer2 on og 1:64
      movwf T2CON             ; Prescaler settings
      movlw B'11000000'       ; GEI og PEIE
      movwf INTCON
      bsf   PIE1, TMR2IE
      bcf   PIR1, TMR2IF

run   goto  run

intr  bcf   PIR1, TMR2IF      ; Enable timer2
      btfsc PORTA, RA2        ; Er port 2 off ?
      goto  off
      goto  on

off   bcf   PORTA, RA2
      RETFIE

on    bsf   PORTA, RA2
      RETFIE

      end
```

Man kunne også forlænge periodetiden ved at lægge en løkke ind inde i intr.

8 Pulse Width Modulation (PWM)

PIC 10F322 har 2 PWM moduler kaldet PWM1 og PWM2. PWM1 bruger RA0 som output og PWM2 RA1 som output.

PWM frekvensen afhænger af Timer2, hvilket betyder PWM er interruptdrevet, og at begge moduler vil have den samme frekvens/periodetid. Dutycyklen behøves ikke være den samme for begge PWM moduler.

Når først et PWM modul er startet, kører det til det bliver afbrudt af programmet, det stopper ikke af sig selv.

Hvis der er brug for højere præcision end Timer2 kan give, kan man benytte NCO modulet.

I dette eksemepel vil vi bruge Port0 (RA0) som output, til at regulere lysstyrken på en LED, ud fra en programmeret dutycykel.

8.1 pwm1.asm

```
;      gpasm -a inhx8m pwm1.asm
;      Port 2 ON/OFF
;      DB Henrik Kressner

;      RTM page 101 and forward

processor    10F322
include             p10f322.inc

       __CONFIG _WDTE_OFF & _FOSC_INTOSC

       org    H'00'  ; Reset vector
       goto   setup
       org    0x04   ; Interrupt vector
       goto   intr

       org    H'05'
setup  clrf   ANSELA       ; All ports are digital I/O
       movlw  B'11111111'
       movwf  TRISA        ; Disable all ports
       clrf   PWM1CON      ; Hvorfor det ????
       movlw  0xff
       movwf  PR2          ; Set PWM period
       clrf   PWM1DCH      ; Hvorfor det ????
       clrf   PWM1DCL      ; Hvorfor det ????

       ; Config timer2
       bcf    PIR1, TMR2IF        ; Clear timer2 interrupt flag
       bcf    T2CON, T2CKPS0      ; Prescaler  (Defaul, kan slettes)
       bcf    T2CON, T2CKPS1      ; Prescaler  (Defaul, kan slettes)
       bsf    T2CON, TMR2ON       ; Timer2 = on

       ; Enable PWM output pin and wait  ?????????
wait   btfss   PIR1, TMR2IF
       goto    wait

       movlw  B'11111110'         ; LED on port RA0   ???????? Se op
       movwf  TRISA
       movlw  B'11000000'
       movwf  PWM1CON
       movlw  0x20                ; On time ??
       movwf  PWM1DCH
       movlw  B'11000000'
       movwf  PWM1DCL
       ; Enable interrupt
       movlw  B'11000000'     ; GEI and PEIE
       movwf  INTCON
       bsf    PIE1, TMR2IE
       bcf    PIR1, TMR2IF

run    goto   run
```

```
intr   bcf     PIR1, TMR2IF    ; Enable Interrupt timer2
       RETFIE

       end
```

8.2 pwm2.asm en servo driver

Lad os skrive et program der gør det muligt at drive en servo. Vi placerer servoens signal på port 0 og har en trykknap på port 1 og 2. Når knap 1 er trykket skal servoen drejes 45 grader den ene vej, når den anden knap trykkes skal den dreje 45 grader den anden vej, når knapperne er sluppet skal servoen stå lige ud.

Hvad der sker hvis vi trykker begge knapper samtidig, gør vi ikke noget ved.

BEMÆRK: Fejl i manualen 10.2, PORTA registeret. Det er kun de to sidste bit der bruges, hvorfor der tælles digitalt til 4 med de 2 sidste bit.

Krav:

Periodetid 20 mS

Dutycycle:

1 mS = Helt med uret
1,5 mS = Midterstilling
2 mS = Helt mod uret

For at få periodetiden ned på 20 mS (= 1/50Hz) er vi nød til at sætte cloclfrekvensen ned til 1 Mhz, den deles med 4, hvilket giver et input til timer2 på 250 Khz. Derefter kan vi sige 250.000Hz/50Hz = 5.000, Vi skal altså dele clockfrekvensen ned 5.000 gange, for at få en periodetid på 20 mS.

Hvis vi sætter prescaleren til at dele med 64, skal timer2 tælle op til: 5.000/64 = 78,1, vi runder ned til 78.

1mS → PWM1DCH + PWM1DCL = 78/20 = 3,9, vi sætter PWM1DCH = 4 og PWM1DCL = 0

1,5 mS → PWM1DCH + PWM1DCL = 78/(20/1,5) = 5,9, vi sætter PWM1DCH = 6 og PWM1DCL = 0

2mS → PWM1DCH + PWM1DCL = 78/10 = 7,8, vi sætter PWM1DCH = 8 og PWM1DCL = 0

```
;     gpasm -a inhx8m pwm2.asm
;     DB Henrik Kressner

processor   10F322
include           p10f322.inc

      __CONFIG _WDTE_OFF & _FOSC_INTOSC

      org   H'00'       ; Reset vector
      goto  init
      org     0x04    ; Interrupt vector
      goto    intr
      org   H'05'
init  movlw   B'00110000'           ; Clock = 1 Mhz
      movwf   OSCCON
      clrf  PORTA
      movlw B'0000110'
      movwf WPUA                    ; Waek Pull Up
      bcf   OPTION_REG, NOT_WPUEN   ; Enable WPUA
      bcf   TRISA, RA0              ; Servo on RA0
      clrf  ANSELA                  ; All ports are digital
      movlw D'78'                   ; Period = 20 mS
      movwf PR2                     ; Configure the Timer2 period
      movlw   D'00000111'           ; Timer2 = on og prescaler = 64
      movwf   T2CON
      movlw B'11000000'
      movwf PWM1CON           ; Enable PWM Module output1 og pin1
      bcf   PIR1, TMR2IF      ; Clear the timer2 interrupt flag
      movlw   B'00000000'
      movwf   PWM1DCL
      ; Enable interrupt
      movlw   B'11000000'     ; GEI and PEIE
      movwf   INTCON
      bsf     PIE1, TMR2IE
      bcf     PIR1, TMR2IF

run   btfss PORTA,RA1   ; Read RA1
      goto  left
      btfss   PORTA,RA2       ; Read RA2
      goto  right
      movlw   D'6'            ; PulseWidth = 1,5 mS
      movwf   PWM1DCH
      goto  run
left  movlw   D'8'            ; PulseWidth = 2 mS
      movwf   PWM1DCH
      goto  run
right movlw   D'4'            ; PulseWidth = 1 mS
        movwf   PWM1DCH
```

```
        goto    run

intr  bcf     PIR1, TMR2IF    ; Enable timer2
      RETFIE

      end
```

9 Configurable logic cell (CLC)

Kommer nok engang.

10 Numerically controlled oscillator (NCO)

Kommer nok engang.

11 Complementary waveform generator (CWG)

Kommer nok engang

12 Sleep and wake

Kommer nok engang

13 Indirekte addressering

Her tænker jeg på brug af en pointer. Dette er angiveligt implementeret, men i praksis virker det ikke og forklaringen i manualen er vidlledene.

Det burde fungere ved at loade en adresse ind i FSR registeret, den adresse vil så fungere som en pointer, som der angiveligt kan kan læses og skrives til, **det er forkert**.

Man kan skrive til det FSR peger på som beskrevet i manualen 2.4 (Indirect Addressing, INDF and FSR Registers), men man kan ikke læse fra det, hvad der også står samme sted i manualen.

Som det fremgår af manualen, returnerer INDF NUL hvis man læser fra det.

Jeg har fundet nogen steder på nettet hvor det påstås, man skal benytte PR0, PR1 og/eller IRP bitne fra statusregisteret, men ifølge manuelen er alle tre bit reserverede, og læses som nul.

Så er der noget med man kan benytte RETLW (Return Literal to W) men det er så primitivt at det ikke er værd at arbejde med.

Det er en alvorlig mangel der ser ud til som minimum at hærge 10F 0g 12F serien.

Sagt på en anden måde, men kan skrive sekventielt til RAM, men man kan IKKE læse sekventielt fra RAM. **Array kan ikke implementeres.**

Appendix A

Enkelt program til at programmere en PIC10F32x kreds fra en Raspberry Py version 2. For at bruge andre versioner af RPi, læs hvor baseadressen til BCM2708_PERI_BASE defineres.

For at compilere: # cc pic.c -o pic

A.1 Kildetekst pic.c

```
//  Use a RPi 2 as PIC10xxx LVP programmer
//  For RPi 3 see comment.
//  Only testet with a 10F322

//  Inspired by: Holden ( Giorgio Vazzana ) at http://holdenc.altervista.org/rpp/
//  Done by Henrik Kressner 2020

//  This is free software, use at your own risc.

// ToDo:
// Read out in HEX
// Test for error
// Test and use arguments


//  To compile (must be root) :  cc pic.c -o pic
//  To run (must be root): cat filename.hex | ./pic


#define VERSION "0.49"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>

//  Access from ARM Running Linux
//  For Raspberry Pi 3, change BCM2708_PERI_BASE to 0x3F000000
//  My RPi 2 runs with 0x200000
#define BCM2708_PERI_BASE 0x20000000
#define GPIO_BASE (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

# define VPP 17
# define ICSPCLK 27
# define ICSPDAT 22
# define TCK 100          // Delay 100 uS
# define TDIS 100         // Delay 5 mS
# define TDLY 1           // Delay 1 uS
```

```
# define TERAB 5000 // Delay 5 mS
# define TERAR 2500 // Delay 2.5 mS
# define TEXIT 1          // Delay 1 uS
# define TPEXT 1000 // Delay 1 mS
# define TPINT 2500 // Delay 2.5 mS
// From programmers manual page 10
# define KEYSEQUENCE "01001101010000110100100001010000"

int  mem_fd;
char *gpio_mem, *gpio_map;
char *spi0_mem, *spi0_map;


// I/O access
volatile unsigned *gpio;


// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x)
// or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)
%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0
#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH


void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit (-1);
   }
   /* mmap GPIO */
   // Allocate MAP block
   if ((gpio_mem = malloc(BLOCK_SIZE + (PAGE_SIZE-1))) == NULL) {
      printf("allocation error \n");
      exit (-1);
   }
   // Make sure pointer is on 4K boundary
   if ((unsigned long)gpio_mem % PAGE_SIZE)
     gpio_mem += PAGE_SIZE - ((unsigned long)gpio_mem % PAGE_SIZE);
   // Now map it
   gpio_map = (unsigned char *)mmap(
      (caddr_t)gpio_mem,
      BLOCK_SIZE,
      PROT_READ|PROT_WRITE,
      MAP_SHARED|MAP_FIXED,
      mem_fd,
      GPIO_BASE
   );
```

```
   if ((long)gpio_map < 0) {
      printf("mmap error %d\n", (int)gpio_map);
      exit (-1);
   }
   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;
} // setup_io


void help()
{
      printf("\nPIC 10Fxxx Lov Voltage Programmer version: %s : Syntax:\n", VERSION);
       printf("pic c -> Send 512 clock pulse to 10F32x\n");
      printf("pic f <filename.hex> -> ");
      printf("Write hex file to 10F32x (Not implementet, use piping)\n");
      printf("pic rb <a> <b> -> ");
      printf("Read binary from 10F32x address a to address b\n");
      printf("pic rh <a> <b> -> ");
      printf("Read hex from 10F32x address a to address b (Not implementet)\n");
      printf("No argument -> asuming hex file is piped:  cat hexfile.hex | pic\n");
      printf("DB Henrik Kressner 2020.\n");
}


// Convert txt based Hex on pos a to b to a int
int getIntFromString(char tmp[], int a, int b)
{
        int i, sum = 0;

        for (i = a; i <= b; i++)
        {
                sum = sum * 16;
                if (tmp[i] >= '0' && tmp[i] <= '9')
                        sum = sum + (tmp[i] - '0');
                else if (tmp[i] >= 'A' && tmp[i] <= 'F')
                        sum = sum + (tmp[i] - 'A' + 10);
        }
      return sum;
}

// Convert one byte to a array of char,
// with one bit per char
void CharToBit(char x, char holder[9])
{
      char i;

      holder[8] = '\0';
      for (i = 0; i < 8; ++i)
            if ( (x << i) & 128 )
                  holder[i] = '1';
            else
                  holder[i] = '0';
}     // CharToBit
```

```
// Send a command, with data, to PIC10F32x
// Command = 00H : Load Configuration
// Command = 02H : Load Data For Program Memory
// Command = 04H : Read Data From Program Memory
// Command = 06H : Increment Address
// Command = 16H : Reset Address
// Command = 08H : Begin Internally Timed Programming
// Command = 18H : Begin Externally Timed Programming
// Command = 0AH : End Externally Timed Programming
// Command = 09H : Bulk Erase Program Memory
// Command = 11H : Row Erase Program Memory
void SendCommand(char Command[])
{
      int i;

   INP_GPIO(ICSPCLK); // Must use INP_GPIO before we can use OUT_GPIO
   OUT_GPIO(ICSPCLK);
   INP_GPIO(ICSPDAT);
   OUT_GPIO(ICSPDAT);
      GPIO_CLR = 1<<ICSPCLK;
   for (i = 7; i > 1; i--)
        {
             if (Command[i] == '1')
             GPIO_SET = 1<<ICSPDAT;
            else
             GPIO_CLR = 1<<ICSPDAT;
                          fflush(stdout);
                          usleep(TCK);
            // One clock cycle
            GPIO_SET = 1<<ICSPCLK;
            usleep(TCK);
            GPIO_CLR = 1<<ICSPCLK;
            usleep(TCK);
        }
}     // SendCommand

void IncrementPC()
{
      char tmp[9];

      CharToBit(0x06, tmp);     // Increment PC
      SendCommand(tmp);
      usleep(TCK);
}

void WriteConfig(char ConfigWord[], int Address)
{
      int i;
      char tmp[9];
      char WriteData[8];

      printf("Config address = %x\n", Address);
      GPIO_CLR = 1<<ICSPCLK;
      usleep(TPINT);
      CharToBit(0x00, tmp);     // Load Configuration
```

```
      SendCommand(tmp);                       // Now PC = 0x2000
      printf("Write config to device at address %x:\t", Address);
      // Data is 14 bit
      usleep(TPINT);
      for (i=15; i>=0; i--)
      {
            GPIO_SET = 1<<ICSPCLK;
            if (ConfigWord[i] == '1')
                  GPIO_SET = 1<<ICSPDAT;
            else
                  GPIO_CLR = 1<<ICSPDAT;
            printf("%c", ConfigWord[i]);
            usleep(TCK);
//          fflush(stdout);
            GPIO_CLR = 1<<ICSPCLK;
            usleep(TCK);
      }

      // Click PC frem
      for (i=0; i<Address-0x2000; i++)
                  IncrementPC();

      CharToBit(0x08, WriteData);     // Begin Internally Timed Programming
      SendCommand(WriteData);
      usleep(TPINT);
} // WriteConfig

// Manual page 12
void WriteData(char x[])
{
      int i;
   char WriteData[8];

      GPIO_CLR = 1<<ICSPCLK;
      usleep(TPINT);
   CharToBit(0x02, WriteData);     // Load Data For Program Memory
      SendCommand(WriteData);
      printf("Write data to device :\t\n");
      // Selve data (14 bit)
   usleep(TPINT);
   for (i=15; i>=0; i--)
   {
      // Clock up
            GPIO_SET = 1<<ICSPCLK;
            if (x[i] == '1')
                  GPIO_SET = 1<<ICSPDAT;
            else
                  GPIO_CLR = 1<<ICSPDAT;
            usleep(TCK);
//          fflush(stdout);
            // Clock down
     GPIO_CLR = 1<<ICSPCLK;
     usleep(TCK);
      }
   CharToBit(0x08, WriteData);     // Begin Internally Timed Programming
```

```
   SendCommand(WriteData);
      usleep(TPINT);
}  // WriteDate



// Manual page 12
void ReadData(char x[])
{
       int i;
       char READDATA[8];

       GPIO_CLR = 1<<ICSPCLK;
       CharToBit(0x04, READDATA);
       SendCommand(READDATA);
       usleep(TDLY);
       printf("Reading from device :\t");
       INP_GPIO(ICSPDAT); // Dont need out, this is read.
       for (i=15; i>=0; i--)
       {
              GPIO_SET = 1<<ICSPCLK;
      usleep(TCK);
      GPIO_CLR = 1<<ICSPCLK;
      usleep(TCK);
              if (GET_GPIO(ICSPDAT))
                     x[i] = '1';
              else
                     x[i] = '0';
              if (i== 0 || i == 15)
                     x[i] = 'x';
              if (i== 7)
                     printf("%c", ' ');
              printf("%c", x[i]);
              fflush(stdout);
              usleep(TCK);
       }
}  // ReadDate


// Transmit n bit to port, one bit at the time LSB first
void ByteToPin(char x[], int n)
{
       int i;

       n--;  // Array starts at 0

       GPIO_CLR = 1<<ICSPCLK;
        for (i = n; i >= 0; i--)
        {
                           if (x[i] == '1')
             GPIO_SET = 1<<ICSPDAT;
            else
             GPIO_CLR = 1<<ICSPDAT;
//                         fflush(stdout);
```

```
                        // One clock cycle
                        GPIO_SET = 1<<ICSPCLK;
                        usleep(TCK);
                        GPIO_CLR = 1<<ICSPCLK;
            usleep(TCK);
        }
}      // ByteToPin

void ReadPIC(int from, int to)
{
       int i;
       char tmp[9];
       char holder[17];

       // Read i address
       printf("Reading from %d to %d\n", from, to);
   CharToBit(0x16, tmp);     // Reset PC
   SendCommand(tmp);
   for (i = 0; i <from; i++)
            IncrementPC();
   for (i = from; i <=to; i++)
       {
            printf("PC = %d : ",i);
            ReadData(holder);
            printf("\n");
            IncrementPC();
       }
} // ReadPIC

void WritePIC(char Line[], char BulkErase)
{
       char tmp[9], tmp1[9], tmp2[9];
       char DataWord[17];
       int i,j,k,l,m;
       int ByteCount, Address;

       i = 0;
       ByteCount = getIntFromString(Line,1,2)/2;           // Programmeres ref. page 25
       Address   = getIntFromString(Line,3,6)/2;
       if (BulkErase)
       {
          CharToBit(0x09, tmp);     // Bulk erase
          SendCommand(tmp);
              usleep(TERAB);
       }

       // Set PC to Address
       CharToBit(0x16, tmp);     // Reset PC
       SendCommand(tmp);

       if (Address < 0x2000)
              for (i=0; i<Address; i++)
                     IncrementPC();

       // Find DataWords
```

```
	for (i=0; i<ByteCount*4; i+=4)
	{
		DataWord[0] = '\0';
		j = getIntFromString(Line, i+9, i+9);
		k = getIntFromString(Line, i+10, i+10);
		j = j << 4 | k;
		CharToBit(j, tmp1);
		j = getIntFromString(Line, i+11, i+11);
		k = getIntFromString(Line, i+12, i+12);
		j = j << 4 | k;
		CharToBit(j, tmp2);
		strcat(DataWord, tmp2);
		strcat(DataWord, tmp1);

		// Move one bit left
		for (l=0; l<16; l++)
			DataWord[l] = DataWord[l+1];
		DataWord[0] = DataWord[15] = 'x';

		if (Address >= 0x2000)			// Its Config
			WriteConfig(DataWord, Address);
		else
			WriteData(DataWord);

		usleep(TDLY);
		IncrementPC();
		usleep(TCK);
	}
} // WritePIC

void InitGPIO()
{
	setup_io();
	INP_GPIO(ICSPCLK);  // ICSPCLK as input
	INP_GPIO(ICSPDAT);  // ICSPDAT som input
	INP_GPIO(VPP);		// must use INP_GPIO before we can use OUT_GPIO
	OUT_GPIO(VPP);
	GPIO_SET = 1<<VPP;
	usleep(10);
	GPIO_CLR = 1<<VPP;  // Now controller is reset, we can enable ICSPCLK & ICSPDAT
	OUT_GPIO(ICSPCLK);
	OUT_GPIO(ICSPDAT);
	usleep(10);

	// Init for lov voltage programming
	ByteToPin(KEYSEQUENCE,32);
	// KEYSEQUENCE needs 33 clockcycle
	GPIO_SET = 1<<ICSPCLK;
   usleep(TCK);
   GPIO_CLR = 1<<ICSPCLK;
   usleep(TCK);
} //InitGPIO

void writeToPIC()
{
```

```
	char test[256];

	scanf("%s", test);
	if (test[0] == ':')
		WritePIC(test,1);
	else
	{
		printf("ERROR: Missing : in hex file first line\n");
		return;
	}
	scanf("%s", test);
	while( strcmp(test, ":00000001FF") )
	{
		printf("\n! %s\n", test);
		if (test[0] == ':')
			WritePIC(test,0);
		else
			printf("ERROR: Missing :\n");
		scanf("%s", test);
	}
	printf("\n");
} // writeToPIC

void delay(int number_of_seconds)
{
    // Converting time into milli_seconds
    int milli_seconds = 1000 * number_of_seconds;
    // Storing start time
    clock_t start_time = clock();

    // looping til required time is achieved
    while (clock() < start_time + milli_seconds)
        ;
} // delay()


void TicToc()
{
	int i;

	GPIO_SET = 1<<VPP;
	  for (i=0; i<512; i++)
	  {
		printf("i = %d\n", i);
		GPIO_SET = 1<<ICSPCLK;
		delay(1);
		GPIO_CLR = 1<<ICSPCLK;
		delay(1);
	  }
}

void main(int a, char *s[])
{
	int i;
```

```
      printf("PIC10Fxxx LVP programmer version: %s\n", VERSION);
      InitGPIO();
      if ( a == 1 )
            writeToPIC();
      if ( a == 2 )
      {
            if ( !strcmp("f", s[1]) )
                  printf("Read from hex file: Not implementet\n");
            if ( !strcmp("h", s[1]) || !strcmp("H", s[1]) )
                  help();
               if ( !strcmp("c", s[1]) || !strcmp("C", s[1]) )
                     TicToc();
      }
      if ( a == 4 )
      {
            if ( !strcmp("rb", s[1]) )
                  ReadPIC( atoi(s[2]), atoi(s[3]) );
                  else if ( !strcmp("rh", s[1]) )
                        printf("Hex out: Not implementet\n");
                        else help();
      }

      // Set CLK and DAT to input and exit program mode
   INP_GPIO(ICSPCLK);
   INP_GPIO(ICSPDAT);
      usleep(TCK);
      // Now we can raise Vpp
   OUT_GPIO(VPP);
   GPIO_SET = 1<<VPP;
}
```

A.2 Serial transmission Rpi ↔ PIC32x

Et enkelt program til at sende data fra en PIC32x til en RPi.

Programmet består af to programmer, et assambler til PIC32x og et C program til Rpi'en.

```
;     gpasm -a inhx8m transtst.asm
;     Testprogram for seriel overførsel til RPi
;     DB Henrik Kressner
;     Limit er en konstant der overføres til Result.
;     Resultat sendes serielt til en RPi

processor   10F322
include     p10f322.inc

            __CONFIG _WDTE_OFF & _FOSC_INTOSC

Result      org     0x40
i     org   0x41
Limit equ   D'99'

      org   H'00'  ; Reset vector
      goto  setup
      org   0x05
setup movlw B'01010000'
      movwf OSCCON            ; Clk Frq = 4 Mhz
      clrf  ANSELA            ; Port 0, 1 og 2 digital I/O
      movlw B'00000110'
      movwf TRISA       ; Port 0 er output, 1 og 2 er input
      clrf  PORTA
start movlw Limit       ; En hukkommelsesplads er an literal
      movwf Result
      movlw 0x9         ; Vi skal overføre 8 bit og stoppe
      movwf i
      goto  Tx

Tx    decfsz      i,1             ; Har vi talt til 8 ?
      goto  cont
      goto  start
cont  btfsc PORTA, RA1      ; Vent på RPi
      goto  $-1
      btfss PORTA, RA1  ; RPi vil have en bit
      goto  $-1
      rrf   Result,1
      btfsc STATUS,C    ; Tjek carry
      goto  on
      goto  off

on    bsf     PORTA,RA0       ; Port 0 ON
```

```
      goto    Tx

off   bcf     PORTA,RA0       ; Port 0 OFF
      goto    Tx

      end
```

Her er C programmet til Rpi'en, det bygger over samme fundament som pic.c programmet.

```
//  Use a RPi 2 til at snakke med temp måler
//  To compile (must be root) :  cc trans.c -o trans

#define VERSION "0.12"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>


//  GPIO Basic address
//  Se: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
//  For Raspberry Pi 2 BCM2708_PERI_BASE = 0x20000000 (BCM2835)
//  For Raspberry Pi 3 BCM2708_PERI_BASE = 0x3F000000 (BCM2836 og BCM2837)
//  For Raspberry Pi 4 BCM2708_PERI_BASE = 0xfe000000 (BCM2711)


#define BCM2708_PERI_BASE        0x20000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO
controller */
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

// # define VPP  17
// # define ICSPCLK  27
// # define ICSPDAT  22

# define ICSPCLK   17
# define ICSPDAT  27
# define VPP  22

# define TCK 0.         // Delay 100 nS: Ændret 2/6 2021 fra 100
# define TDIS 100 // Delay 5 mS
# define TDLY 1         // Delay 1 uS
# define TERAB 5000     // Delay 5 mS
# define TERAR 2500     // Delay 2.5 mS
```

```
# define TEXIT 1  // Delay 1 uS
# define TPEXT 1000     // Delay 1 mS
# define TPINT 2500     // Delay 2.5 mS


int  mem_fd;
char *gpio_mem, *gpio_map;
char *spi0_mem, *spi0_map;

int   OFFSET = 0x2000;


// I/O access
volatile unsigned *gpio;


// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x)
// or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?
3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0
#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH


void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit (-1);
   }
   /* mmap GPIO */
   // Allocate MAP block
   if ((gpio_mem = malloc(BLOCK_SIZE + (PAGE_SIZE-1))) == NULL) {
      printf("allocation error \n");
      exit (-1);
   }
   // Make sure pointer is on 4K boundary
   if ((unsigned long)gpio_mem % PAGE_SIZE)
     gpio_mem += PAGE_SIZE - ((unsigned long)gpio_mem % PAGE_SIZE);
   // Now map it
   gpio_map = (unsigned char *)mmap(
      (caddr_t)gpio_mem,
      BLOCK_SIZE,
      PROT_READ|PROT_WRITE,
```

```
      MAP_SHARED|MAP_FIXED,
      mem_fd,
      GPIO_BASE
   );

   if ((long)gpio_map < 0) {
      printf("mmap error %d\n", (int)gpio_map);
      exit (-1);
   }
   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;
} // setup_io


void help()
{
      printf("\nPIC 10Fxxx Lov Voltage Programmer version: %s : Syntax:\n",
VERSION);
        printf("pic c -> Send 512 clock pulse to 10F32x\n");
      printf("pic f <filename.hex> -> ");
      printf("Write hex file to 10F32x (Not implementet, use piping)\n");
      printf("pic rb <a> <b> -> ");
      printf("Read binary from 10F32x address a to address b\n");
      printf("pic rh <a> <b> -> ");
      printf("Read hex from 10F32x address a to address b (Not implementet)\
n");
      printf("No argument -> asuming hex file is piped:  cat hexfile.hex | pic\
n");
      printf("DB Henrik Kressner 2020.\n");
}


void InitGPIO()
{
        setup_io();
        INP_GPIO(ICSPCLK);      // ICSPCLK as input
        INP_GPIO(ICSPDAT);      // ICSPDAT som input
        INP_GPIO(VPP);          // must use INP_GPIO before we can use OUT_GPIO
        OUT_GPIO(VPP);
        GPIO_SET = 1<<VPP;
        usleep(10);
        GPIO_CLR = 1<<VPP;      // Now controller is reset, we can enable
ICSPCLK & ICS$
        OUT_GPIO(ICSPCLK);
        OUT_GPIO(ICSPDAT);
        usleep(10);
   usleep(TCK);
   GPIO_CLR = 1<<ICSPCLK;
   usleep(TCK);
```

```
} //InitGPIO



void main(int a, char *s[])
{
      char  tmp, ind;
      int i,h, run;
      float temp, Vsensor;

        printf("PIC10Fxxx kommunikation version: %s\n", VERSION);
        InitGPIO();

      // Set CLK and DAT to input
      INP_GPIO(ICSPCLK);
      INP_GPIO(ICSPDAT);
      usleep(110);
      // Now we can raise Vpp
      OUT_GPIO(VPP);
      GPIO_SET = 1<<VPP;
      OUT_GPIO(ICSPCLK);
      GPIO_CLR = 1<<ICSPCLK;
      usleep(1000);

      //Reset MCU
      GPIO_CLR = 1<<VPP;
      sleep(0.1);
      GPIO_SET = 1<<VPP;
      printf("Har resat\n");
      sleep(0.1);
      printf("-");
      run = 1;
      while (run)
      {
            tmp = 0;
            for(i=0; i<8; i++)
            {
                  GPIO_SET = 1<<ICSPCLK;
                  sleep(0.1);
                  h = GET_GPIO(ICSPDAT);
                  if (h)
                  {
                        printf("1");
                        tmp = tmp >> 1;
                        tmp = tmp + 128;
                  }
                  else
                  {
                        printf("0");
```

```
                        tmp = tmp >> 1;
                  }
                  GPIO_CLR = 1<<ICSPCLK;
                  sleep(0.1);
            } // for
            printf(" : tmp = %d\n", tmp);
            if ((ind = getchar()) == 's')
                  run = 0;
      }

      // RYD OP
      INP_GPIO(ICSPCLK);
      INP_GPIO(ICSPDAT);
}
```