Digital skala

Af OZ1DV John Gregersen, Huldbergs Allé 46, 2800 Lyngby

Byggeprojekt

En mere passende titel var måske 'Digital frekvensudlæsning endnu en gang', og en forklaring er bestemt påkrævet:
I EDR Gladsaxe afdeling besluttede vi os for et byggeprojekt: En 80 meter SSB transceiver. Efter søgen i litteraturen faldt valget på OZ1UMs Mikrotransceiver beskrevet af OZ9ZI i OZ nr. 3, 4 og 5, 1994. En halv snes af afdelingens medlemmer (undertegnede inklusive), er nu i fuld sving med at bygge hver sit eksemplar.

Frekvensudlæsning

Konstruktionen indeholder ikke en egentlig frekvensudlæsning, men foreslår et voltmeter tilsluttet VFO'ens reguleringsspænding. Alternativt kan en programmerbar tæller, vistnok oprindelig konstrueret af OZ8AO i 1991 til et 10 GHz projekt, anvendes.
Efter at have set lidt på denne og andre af fortidens konstruktioner måtte vi sande, at udviklingen i den digitale verden går meget hurtigt for tiden, og at en væsentlig nemmere og ikke mindst billigere konstruktion var mulig ved anvendelse af en mikroprocessor.
Det blev derfor besluttet at forsøge med konstruktion af en skala til netop OZ1UMs transceiver baseret på en mikroprocessor. En sådan blev løseligt designet efter følgende kriterier:

Design

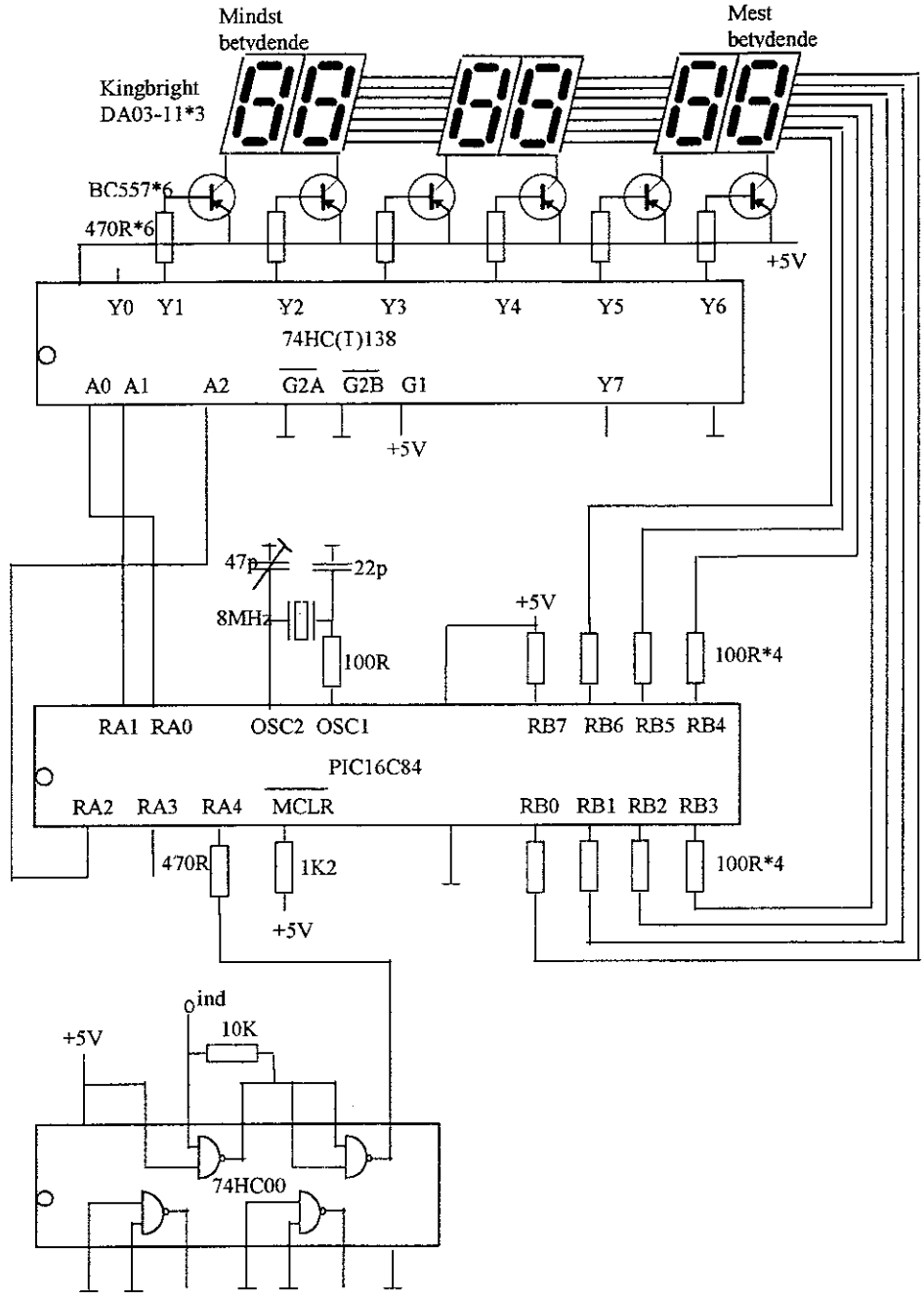
Konstruktionen skal udlæse modtage- og sende-frekvensen baseret på den overliggende VFOs frekvens; mellemfrekvensen skal altså fratrækkes. Frekvensen vises på et antal (vi startede med fire) syvsegment displays. Displayene parallelforbindes, og data multiplexes ud ved angivelse af ciffer nr. og af hvilke segmenter i cifferet, der tændes. Det enkelte ciffer tændes mindst 60 gange pr. sekund for at undgå flimmer
Vi var egentlig indstillet på kun at vise ciffer 2, 3, 4 og 5 (f. eks. vise 3,123456 MHz som 1234) af sparehensyn, men efterhånden som konstruktionen skred frem (og vi fandt frem til, at vi alligevel måtte indføre en ekstern dekodning af cifrene), kom der et ekstra ciffer på i hver ende (så vi nu viser 312345) Når vi ikke tog det sidste ciffer med, skyldes det, at vi (foruden at spare) hellere vil aflæse frekvensen 10 gange i sekundet end have dette ciffer med.

Hardware

Efter således at have lavet en løselig skitse gik jagten gennem de samlede deltageres rodekasser og erfaringsmateriale i gang. Valget faldt på processoren PIC16C84 fra Microchip. Kredsen indeholder (næsten) alt det nødvendige for at varetage både frekvenstælling og styring af display.

Beskrivelse af PIC16C84

I diagrammet i figur 1 kan man få et overblik over kredsens tilslutningsmuligheder. Den indeholder en mængde funktionelle moduler, hvoraf kun de relevante skal nævnes her. Fuldstændig beskrivelse kan hentes på Microchips Web med adressen http://www.microchip.com, hvor også compiler og simulator kan hentes.
Foruden CPU, registre, EEPROM til programmet og fornøden clockning har kredsen 13 I/O-pins, hvis funktioner er programmerbare. Til netop vores anvendelse er en indbygget 8-bit tæller og en 8-bit prescaler særdeles anvendelige.
CPU'en er en RISC-processor, og som følge deraf ganske hurtig: Alle instruktioner, med undtagelse af 'hop' (branch) udføres på fire clockcycles. Med et 8 MHz krystal giver det 0,5 mikrosekunder pr. instruktion eller 2 mio. instruktioner pr. sekund. Kredsen kan køre endnu hurtigere, nemlig 10 MHz, men rodekassen indeholdt altså 8 MHz krystaller.
Instruktionssættet er naturligvis ikke overvældende, men det er 14-bit instruktioner, så ved en pæn systematik er der alligevel opnået en forbløffende slagkraft, og samtidig er instruktionerne hurtig lært. Kredsen indeholder 1K*14 bit EEPROM til instruktioner, hvilket er mere end rigeligt.
De fleste funktioner (som f. eks. aflæsning af tæller, input/output o.s.v.) håndteres ved læsning/skrivning til bestemte af de til rådighed værende 48 registre. Således er 12 registre reserveret hertil, og de resterende 36 registre kan anvendes på vanlig vis til variable. Heldigvis er alle registre på 8 bit, så det med de 14 bit i instruktionerne bemærker man ikke.
Tælleren og prescaleren kan programmeres til et væld af funktioner. Her er det interessante, at prescaleren kan dele et signal på RA4 med 256 og sende det videre til tælleren. Prescaleren er en simpel ripplecounter, der går til 20 MHz, men den kan ikke aflæses. Det er naturligvis et problem, og det vender vi tilbage til.
Tælleren er en konventionel enhed, der ligeledes kan tælle til 256. Den kan sættes og aflæses i register 1, og overflow indikeres med en overflowbit. Når tælleren resettes, medfører det samtidig reset af prescaleren.
De 13 I/O-pins har en kapacitet på 25 mA, og de kan således lige drive en LED. De kan frit programmeres til input eller output.
Denne korte beskrivelse burde være tilstrækkelig til, at vi nu kan se nærmere på konstruktionen:



Figur 1, diagram

Beskrivelse af konstruktionen

Når man skal lave denne slags programmerbare systemer, sker det naturligvis i en interaktiv proces mellem software- og hardware-konstruktion. Da konstruktionen nu er lavet, tager vi hardwaren først:

Rodekasserne afslørede nogle fortrinlige syv-segments displays med fælles anode, men man kan naturligvis også anvende enheder med fælles katode hvis man ændrer konstruktion og program på behørig vis.

Diagrammet i figur 1 viser den samlede konstruktion. Det er tegnet lidt specielt (kredsenes fysiske layout er anvendt i stedet for det logiske), fordi det så er muligt at bygge direkte herefter på et stykke hulprint. Vi har overvejet at lave et print til enheden, men det er næppe ulejligheden værd.

Der er lige et lille problem, som vi må have fixet først: Den valgte processor har ikke nok i de 13 I/O-pins. Vi skal bruge 6 I/O til ciffer valg, 7 I/O til segmenter og 1 I/O til input, i alt 14 pins. Man kunne overveje at lade første ciffer være fast sat til et tretal, hvorved vi lige præcis kunne klare os med de 13 I/O-pins. Vi valgte i stedet en løsning, hvor ciffervalget repræsenteres binært, og en 74HC(T)138 anvendes til dekodning, hvorved hele otte cifre kan vises med kun 3 I/O-pins, og vi har to I/O-pins 'til fremtidig brug'.

Herefter anvendes RB0-6 til at vælge segment i displayet. RB7 anvendes til input, idet dette ben aflæses i takt med at cifrene aktiveres. Derved kan otte tilstande (høj eller lav) checkes og anvendes til at styre programmet. RA0-2 anvendes til at angive hvilket ciffer, der skal være tændt. Angivelsen er

binær, og vi må, som nævnt, benytte en dekoder, nemlig 74HC(T)138 til at dekode dette signal. 74HC(T)138 har som input 3 bit og som output otte bit, hvor bit 0 er lav hvis input er 000, bit 1 er lav hvis input er 001, bit 2 er lav hvis input er 010, ... og bit 7 er lav, hvis input er 111.

Den indbyggede oscillator tilsluttes krystallet på standard vis, dog med en trimmekondensator, så frekvensen kan finindstilles.

Tællerens input er på RA4, men der var jo lige det lille problem med prescaleren, der ikke kunne aflæses. I kredsen findes en exclusive-or gate foran prescaleren; der har til formål at gøre det muligt at vælge, om tælleren skal trigge på den positive eller den negative flanke. Det gøres med statusbitten T0SE. Vi anvender denne bit til at skifte prescaleren ud på følgende måde: Når vores 100 millisekunder periode er slut, sættes RA4 til output og lav, hvorved tælleren stopper. Derefter tæller vi, hvor mange gange vi skal vippe med T0SE, før tælleren skifter værdi. Således finder vi ud af, hvad prescaleren stod på, da vi stoppede, og dermed har vi frekvensen, som vi ønskede at finde ud af. En modstand i inputledningen sikrer, at vi kan foretage denne operation uden at ødelægge noget i kredsløbet foran tælleren. For god ordens skyld skal jeg lige gøre opmærksom på, at denne funktion ikke virker korrekt på simulatoren.

Da der ikke anvendes funktioner i processoren, der kan lide under manglende initialisering ved power-up, har vi ikke gjort noget ud af dette, men blot forbundet MCLR på pin 4 til +5 V via en modstand på 1,2 kohm.

Transistorerne er nødvendige, da en 74HC(T)138 ikke kan trække de op til syv samtidige LEDs, der vil være tændt, når vi skal vise et ottetal.

Vi fandt nogle dobbelte syvsegments enheder af typen Kingbright DA03-11, og vi har anvendt tre af dem. De er monteret på et separat stykke hulprint, og katoderne fra de tre enheder parallelforbindes. Fra disse syv sammenbundne katoder fører en stump fladkabel til hovedprintet, hvor de forbindes via 100 ohm modstandene til PIC16C84. De seks anoder, en for hvert ciffer, forbindes på samme måde til driver-transistorerne på hovedprintet.

En 74HC00 anvendes til at forstærke signalet, så det forholdsvis svage niveau vi kan få fra VFO'en, bliver tilstrækkelig kraftigt til at drive RA4.

Med hardwaren på plads mangler vi 'kun' et program, der kan få liv i herligheden.

Programbeskrivelse

Man behøver ikke at studere programmet nærmere, med mindre man ønsker at ændre heri. Når det alligevel beskrives, er det fordi, det netop er tanken med en beskrivelse som denne, at den forhåbentlig vil give andre inspiration til at lave noget tilsvarende, men tilpasset netop deres behov. Altså: Hvis du skal bruge skalaen præcis, som den er, så skip dette afsnit!

Flowdiagrammet i figur 2, suppleret med programlistningen, beskriver programmet

Først erklæres en række symbolske navne for processorens fast definerede registre, f.eks. PORT_B, der tildeles værdien 5. Når der senere i koden refereres til PORT_B betyder det simpelt hen, at der læses eller skrives i register 5. Det vil ved læsning medføre, at man indlæser tilstanden på pin RB0-7, og ved skrivning at man sætter PIN RB0-7 til de udskrevne værdier.

En række symbolske navne erklæres for de registre, vi ønsker at anvende til opbevaring af mellemresultater undervejs i programmet. Anvendelsen af disse tager vi efterhånden, som programmet gennemgås.

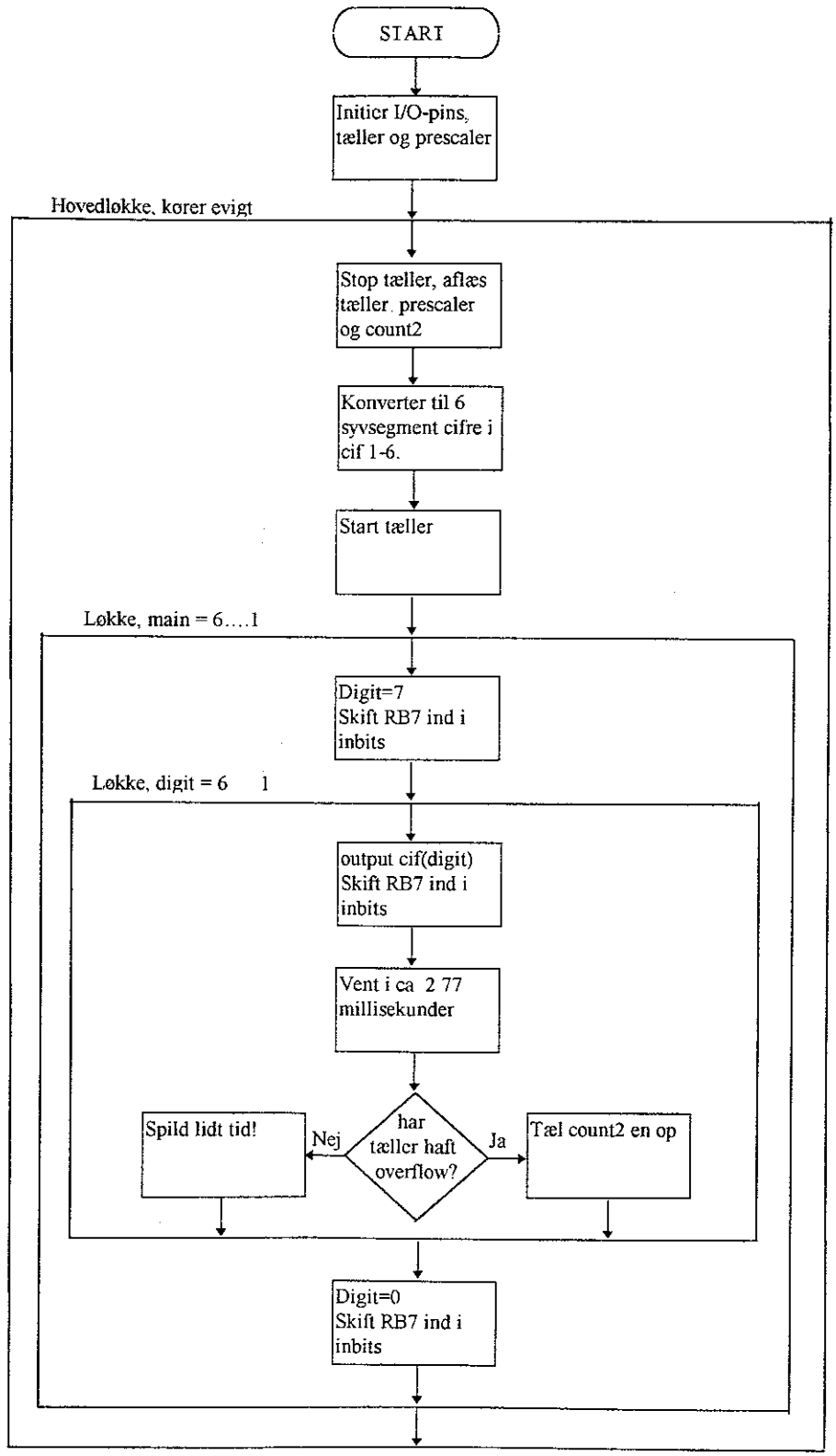
Et program som dette består af en uendelig løkke, hvor frekvensen aflæses og vises i en uendelighed. Hvor en sådan løkke starter er ret ligegyldig, og vi starter derfor tilfældigvis med at stoppe tælleren. Det gøres ved at sætte RA4 til output. Dernæst tages resultatet i count2, der er antallet af gange tælleren har været rundt, og anbringes i X_byte. Tælleren aflæses i TMR0, og resultatet anbringes i H_byte. Vi mangler nu værdien af prescaleren, der skal ende i L_byte. Prescalerens værdi kan ikke aflæses, og vi skal derfor finde ud af, hvor mange gange vi skal vippe indgangen, før prescaleren løber over, og tælleren skifter værdi. Dette tal skal fratrækkes 256, og vi har prescalerens værdi. Det gøres ved at sætte L_byte til 0, vippe T0SE, trække en fra L-byte og checke, om tælleren har skiftet. Hvis ja, er vi færdige, og prescalerens værdi står i L_byte, hvis nej vippes T0SE igen o.s.v.

Nu indeholder de tre variable X_byte, H_byte og L_Byte vores frekvens repræsenteret binært. Mellemfrekvensen skal fratrækkes, dog kun værdien 45.350, da vi jo kun tæller i 0,1 sekund, og derfor har en tiendedel af frekvensen. PIC16C84 har ikke instruktioner til subtraktion, så vi må i stedet addere værdien -45.350. Dette tal anbringes i variablene X_add, H_add og L_add, hvorefter proceduren D_add kaldes. Ved retur fra D_add er subtraktionen foretaget, og variablene X_byte, H_byte og L_Byte indeholder frekvensen binært minus 45.350.

B2_BCD kaldes for at konvertere den binære værdi i variablene X_byte, H_byte og L_Byte til decimal repræsentation i de tre variable R0, R1 og R2. Disse indeholder herefter seks decimale cifre repræsenteret ved fire bit af en af de tre variable, sådan at forstå, at de fire mest betydende bit i R0 indeholder værdien af det decimale ciffer, der skal vises som det mest betydende ciffer i vort display.

Disse seks cifre maskes ud, og konverteres et for et til deres repræsentation i syvsegment, der placeres i de seks variable cif1-6 med det mest betydende ciffer i cif6.

Nu er vi klar til at vise frekvensen i displayet. Det skal



Figur 2, flowdiagram

ske samtidig med, at vi tæller os frem til en ny frekvens i en 0,1 sekunders periode. I en sådan periode vil vi gerne have hvert ciffer tændt 6 gange (= 60 gange i sekundet). Det styres af variablen main, der underopdeler perioden i seks dele. Disse deles igen i 6 dele, der er perioden hvori et enkelt ciffer skal være tændt, styret af variablen digit.

Tælleren resettes og startes. RA0-2 sættes til 7, og vi skifter lige RB7 ind i inbits, inden vi tæller RA0-2 en ned (til 6) og sætter det tilhørende syvsegment ciffer fra cif6 på RB0-6. For ikke at introducere en 'skyggeeffekt' i displayet, er alle cifre slukket, mens RB0-6 ændres. Det sker ved at sætte RA0-2 til 0, idet dette ciffer ikke anvendes.

Tælleren er i gang, og et ciffer er tændt. Vi har nu blot at vente i en seksogtredvtedel af 100 msek (2,7777 msek), indtil det næste ciffer skal tændes. Vi venter i en løkke, der kun har dette formål. Derefter checker vi lige, om vores tæller har været løbet over. Hvis det er tilfældet, tælles count2 en op; hvis ikke, skal vi sørge for at spilde lige så meget tid, som hvis vi havde overflow, ellers vil vores 100 msek tid variere en smule afhængig af den målte frekvens, som dermed vil blive forkert. Vi kan nøjes med denne check af overflow her, idet overflow først vil optræde efter 256 gange 256 eller 65536 Hz, svarende til en frekvens der er 36*10 gange så høj eller over 23 MHz (hvilket ikke er muligt, da prescaleren max. kan tælle 20 MHz). Herefter er vi klar til næste ciffer.
Når alle seks cifre har været tændt, skal vi have den til cifferet 0 hørende værdi skiftet ind i inbits.
Løkken, der tænder de seks cifre, køres seks gange, styret af variablen main. Når det er sket, skulle der være gået 100 msek., og vi kan igen stoppe tælleren og begynde dekodningen.
For at justere løkken helt præcist til 100 millisekunder er der anbragt nogle dummyinstruktioner undervejs i programmet. De er tydeligt markeret i programlistningen.

Programmering af processoren

Programmet, som det er vist her, er tilgængeligt fra EDRs programbank. Det består af de to filer skala.cod og skala.hex. Hvis det skal anvendes præcis, som det er, skal man kun bruge skala.hex.
Vil man ændre i programmet, er det tekstfilen skala.cod, der ændres, hvorefter programmet skal oversættes til binær form, inden det kan indlægges i processorens lager. Det klarer assembleren, som kan hentes på førnævnte Web-adresse. Den danner foruden filen ved navn skala.hex, der er vores binære fil, også filer med listning og fejlmeldinger. Inden programmet brændes ind i processoren kan man forvisse sig om at det virker efter hensigten ved at køre det på den simulator, der også kan hentes på Web'en.
For at få den dannede *.hex-fil ind i processoren, skal der anvendes en 'brænder' (som i dette tilfælde dog ikke 'brænder noget', processoren kan omprogrammeres) PIC16C84 er en meget anvendt processor, hvorfor det kan det tænkes, at en brænder allerede findes i omgangskredsen og kan lånes, da den jo blot skal bruges en enkelt gang. Ellers er den let bygget, en beskrivelse kan findes på http://www.gbar.dtu.dk/~c888600/newpic.htm. En anden adresse med mange oplysninger om emnet er http://www.man.ac.uk/~mbhstdj/piclinks.htm.
Når man har lånt eller bygget en brænder, overføres *.hex-filen til processoren, der herefter er klar til brug.

Afsluttende bemærkninger

Du har sikkert allerede indset, at konstruktionen faktisk er en frekvenstæller, og da jeg ikke i forvejen rådede over et sådan apparat, var fristelsen til modificere den beskrevne konstruktion til en frekvenstæller da også for stor.
Ved hjælp af et par ekstra kredse og et ændret program, kom der en otte-cifret frekvenstæller til 2 GHz ud af det, og en beskrivelse er på vej.

Programliste:

```
        LISI   P = 16C84.  F = INHX8M. n = 66
        __config            H'3FFE'
;*****************************************************
************
;
;Programmet tæller input frekvensen på RA4 i 100 mS  Kon-
verterer
;tallet til seks 7-segment cifre og viser dem.
;Outputs er aktiv lav
;                           RB0
;                  RB5                    RB1
;                           RB6
;                           RB4                    RB2
;                           RB3
;
;Der vises seks cifre  og nummeret på det aktuelle ciffer
sættes på
;RA0-RA2 hvor den binære værdi 1 signalerer det mindst
betydende
;ciffer  og 6 det mest betydende. 0 og 7 anvendes ikke.
;
;*****************************************************
************
;
INDF                equ       0
TMR0                equ       1
PCI                 equ       2
SIATUS    equ       3
FSR                 equ       4         ;  Definition  af
systemvariable
PORI_A              equ       5
PORI_B              equ       6
INIICON   equ       0B
;*****************************************************
************
X_byte              equ       12
H_byte              equ       13
I_byte              equ       14
R0                  equ       15          ; Definition af
arbejdsvariable
R1                  equ       16
R2                  equ       17
count               equ       19
temp                equ       1A
digit               equ       1B
count2              equ       1C
main                equ       1D
inbits              equ       1E
cif1                equ       21
cif2                equ       22
cif3                equ       23
cif4                equ       24
cif5                equ       25
cif6                equ       26
X_add               equ       27
H_add               equ       28
L_add               equ       29
;*****************************************************
************
VAROPI    equ       81
IRISA               equ       85        ;Systemvariable
på page 1
TRISB               equ       86
;
;*****************************************************
************
;                   Initialisering
;*****************************************************
************
          bsf       SIAIUS 5              ;select pg 1
          movlw     0A7                   ;sæt timer på RA4
          movwf     VAROPI      ;med prescaler på 256
          movlw     18
```

```
          movwf     TRISA               ;gør RA0 1.2 outputs og RA3,4 inputs
          movlw     80
          movwf     TRISB               ;gør  RB0-6  outputs og RB7 input
          bcf       SIATUS,5            ;select page 0
          clrf      PORT_A              ;sæt alle outputs lav
          clrf      PORI_B              ;     /
;*****************************************************************
;                   Aflæs tæller
;*****************************************************************
forever3
          bsf       SIAIUS,5            ;select pg 1
          bcf       IRISA,4             ;stop tæller
          bcf       SIAIUS,5            ;select page 0
          movlw     .6
          movwf     main                ;100   mS   løkke kører 6 gange
          movf      count2.0
          movwf     X_byte              ;aflæs  tæller  og anbring i
          movf      TMR0,0              ;X_byte. H_byte
          movwf     H_byte
          clrf      L_byte              ;aflæs prescaler
om        bsf       STATUS.5            ;select pg 1
          bsf       VAROPI.4  ;vip I0SE
          bcf       VAROPT.4  ;
          bcf       SIATUS 5            ;select page 0
          decf      L_byte 1
          movf      H_byte.0  ;samenlign IMR0 med gammel værdi
          subwf     TMR0,0              ;hvis de er forskellige
          btfsc     SIAIUS.2  ;d v s  at Z=0  skip næste instr.
          goto      om
bit1      btfss     inbits 1            ;hvis bit 1 er sat i inbits
          goto      bit2
          movlw     0FF                 ;fratrækkes
          movwf     X_add
          movlw     4E                  ;45.350  fra  den målte
          movwf     H_add
          movlw     0DA                 ;værdi
          movwf     L_add
          call      D_add
bit2
          call   B2_BCD        ;Konverter fra binær til BCD
                                ;     i R0,R1,R2
                                ; d e r herefter konverteres til
          movf      R2 0                ;7-segments    og placeres i cif1..6
          andlw     00F                 ;med mindst betydende ciffer i cif1
          call      LedIable
          movwf     cif1
          swapf     R2,0
          andlw     00F
          call      LedTable
          movwf     cif2
          movf      R1,0
          andlw     00F
          call      LedTable
          movwf     cif3
          swapf     R1,0
          andlw     00F
          call      LedIable
          movwf     cif4
          movf      R0,0
          andlw     00F
          call      LedIable
          movwf     cif5
          swapf     R0,0
          andlw     00F
          call      IedTable
          movwf     cif6                ;cif1 .6 er nu i 7-seg.
          clrf      TMR0                ;clear timer
          clrf      count2              ;og overflow tæller
bkpnt
          bsf       SIAIUS,5            ;select pg 1
          bsf       TRISA,4             ;start tæller
          bcf       SIAIUS.5            ;select page 0
          nop                           ;for at finjustere 100 mS løkke.
          nop                           ;eksekveres     1 gang
          nop
          nop
forever4
          nop                           ;for at finjustere 100 mS løkke.
          nop                           ;eksekveres     6 gange
          nop
          movlw     7
          movwf     digit               ;nummer  på  aktuelt ciffer i display
          movwf     PORI_A              ;
          rlf       PORI_B 0            ;aflæs PORI B og
          rlf       inbits 1            ;skift bit 7 ind i inbits
          decf      digit 1
forever5
          nop                           ;for at finjustere 100 mS løkke.
          nop                           ;eksekveres     36 gange
          nop
          nop
          nop
;*****************************************************************
;                   Vis et ciffer
;*****************************************************************
          clrf      PORT_A              ;sluk dispaly
          movf      digit.0             ;W= ciffer nr.
          addlw     20                  ;ciffer  adresseres indiekte
          movwf     FSR
          movf      INDF.0              ;W=ciffer  i  7-segment rep.
          movwf     PORI_B              ;sæt segmenter
          movf      digit.0             ;sæt  ciffer  nummer
          movwf     PORI_A              ;tænd   aktuelt ciffer
          rlf       PORI_B 0            ;indlæs PORI B og
          rlf       inbits 1            ;skift bit 7 ind i inbits
;*****************************************************************
;                   Vent i 2 777777 mS
;*****************************************************************
          movlw     .18
          movwf     count
w2        movlw      101                ;kør   rundt   en masse gange
          movwf     temp                ;for at få tiden til at gå
w1        decfsz    temp 1              ;
          goto      w1
          decfsz    count.1
          goto      w2
;************************     Slut     på     venteløkke **********************
          clrwdt                        ;sikrer  vi  ikke forstyrres !
          btfsc     INITCON 2 ;har der været overflow på tæller ?
          goto      incrcount2
          nop                           ;spild  lidt  tid for at have
          nop                           ;samme   tidsforbrug som når
          nop                           ;der er overflow
          goto      cont
cont      decfsz    digit 1
```

```
          goto      forever5
          clrf      PORI_A              ;vælg ciffer 0
          rlf       PORI_B 0            ;indlæs PORI B og
          rlf       inbits 1            ;skift bit 7 ind i
inbits
          decfsz    main 1
          goto      forever4  ;løkken er kørt 6*6 gan-
ge=100 mS
          goto      forever3
incrcount2                              ;kommer her når
tælleren har overflow
          bcf       INITCON,2 ;clear overflow bit
          incf      count2 1            ;og læg 1 til
count2
          goto      cont
;*****************************************************
************
;                   Procedure B2_BCD
;              Binary Til BCD Konverterings Routine
;     Denne routine konverterer et 24 Bit binær tal til et 6
cifret
;     BCD tal.
;
;     Det binære tal leveres i adresse X_byte, H_byte and
;     I_byte med X_byte som mest betydende byte.
;      Det 6 cifrede BCD tal returneres i R0. R1 and R2 med
det mest
;     betydende ciffer i R0 s laveste fire bit.
;*****************************************************
************
B2_BCD bcf          SIAIUS.0            ;clear carry bit
          movlw      24
          movwf     count
          clrf      R0                  ;nulstil R0. R1.
R2
          clrf      R1
          clrf      R2
loop16    rlf       L_byte  F ;roter en bit
          rlf       H_byte  F ;gennem
          rlf       X_byte  F ;de seks variable
          rlf       R2. F
          rlf       R1. F
          rlf       R0. F
;
          decfsz    count, F
          goto      adjDEC
          REILW  0
;
adjDEC  movlw       R2                  ;udpeg R2 indi-
rekte
          movwf     FSR
          call      adjBCD              ;og kald adjBCD
;
          movlw     R1
          movwf     FSR                 ;ligeså for R1
          call      adjBCD
;
          movlw     R0
          movwf     FSR                 ;og for R0
          call      adjBCD
;
          goto      loop16
;
adjBCD  movlw       3
          addwf     0.W                 ;læg 3 til Rx s 1.
cif
          movwf     temp
          btfsc     temp 3              ;hvis resultat >
7
          movwf     0                   ;er det det vi
skal bruge
          movlw     30                  ;læg 3 til Rx s 2.
cif
          addwf     0,W
          movwf     temp
          btfsc     temp.7              ;hvis resultat >
7
          movwf     0                             ;er det
det vi skal bruge
          REILW  0
;*****************************************************
************
;                   Procedure led tabel, aktiv lav
;      W indeholder et 4 bit binært tal ved kald. Det konve-
teres til
;          samme tals repræsentation i 7-segment ved retur
;*****************************************************
************
LedTable
          addwf     PCL.1               ;hop W adresser
frem
          retlw     B 01000000          ;led drive for 0
40
          retlw     B 01111001          ;led drive for 1
79
          retlw     B 00100100'         ;led drive for 2
24
          retlw     B 00110000          ;led drive for 3
30
          retlw     B 00011001          ;led drive for 4
19
          retlw     B 00010010          ;led drive for 5
12
          retlw     B 00000010          ;led drive for 6
02
          retlw     B 01111000          ;led drive for 7
78
          retlw     B 00000000          ;led drive for 8
00
          retlw     B 00011000          ;led drive for 9
18
          retlw     B 00001000          ;led drive for A
          retlw     B 00000011          ;led drive for b
          retlw     B 01000110          ;led drive for C
          retlw     B 00100001          ;led drive for d
          retlw     B 00000110          ;led drive for E
          retlw     B 00001110          ;led drive for F
;*****************************************************
************
;          Ire byte addition.
;          I.H.X_byte + L.H.X_add -> I.H.X_byte
;
;*****************************************************
************
D_add     movf      L_add.0
          addwf     L_byte.1            ;læg to mindst
betydende bytes sammen
          btfss     STAIUS.0            ;ved overflow
          goto      nocarry
          movlw     1                   ;lægges 1 til
næste byte. og ved
          addwf     H_byte.1  ;overflow på denne operati-
on
          btfsc     STATUS.0            ;lægges 1 til
højeste byte
          incf      X_byte,1
nocarry   movf      H_add.0
          addwf     H_byte.1            ;læg to mellemste
bytes sammen
          btfsc     STAIUS.0  ;ved overflow
          incf      X_byte.1            ;læg 1 til mest
betydende byte
          movf      X_add.0
          addwf     X_byte.1            ;læg de mest
betydende bytes sammen
          retlw  0

          END
```

OZ