Contents

1 Frequentie teller met PIC 16F5X volgens AN592

Omdat dit idee zo bijzonder is heb ik er wat nader op in gegaan, vermits ik ondervond dat vele hobbyisten, die bij het doornemen van document AN592, het toch nog niet goed begrepen hadden.

De PIC 16C5X heeft een 8 bit timer (TIMER 0) welke kan gebruikt worden als een PRESCALER (van bv.) 256).
Die PIC heeft een input die klokpulsen van 10ns kan volgen op input TOCK1. Dus maximaal 50 MHz, maar praktische proeven hebben aangetoond dat als de input pulsen rechthoekig zijn deze PIC's zelfs pulsen tot 70 MHz kunnen verwerken.

Wanneer nu in TOCK1 meer dan 256 periodes (externe klokpulsen) zijn binnengekomen dan komt er een overflow die de timer TRM0 steeds met één eenheid optelt.
Na een zekere tijd zijn er dus Fin/256 pulsen geteld in TRM0, en zit de rest van de pulsen (tussen 0 en 255) nog in de PRESCALER. In het totaal is er dus een gezamenlijk getal aanwezig van 8bits + 8 bits = 16 bits, zoals is te zien in fig.2.

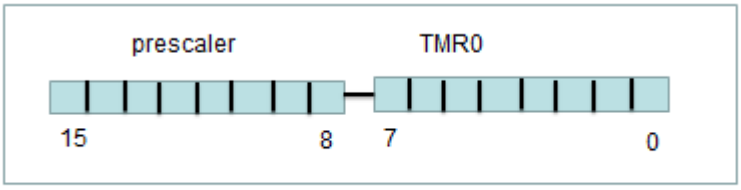

Fig. 2

Dit maakt een mogelijke uitlezing van maximaal 2^{16} = 65.536 klokpulsen/ sec.
Indien deze klokpulsen binnenkomen gedurende 1 ms. Dan kan ik maximaal 65.536 MHz tellen of bij een input van 50 MHz kan ik 50 MHz ± 1kHz.

Het probleem is echter dat de PRESCALER niet rechtstreeks kan uitgelezen worden. Maar men kan wel via instructies BCF en BSF (Bit Clear F en Bit Set F) nadat de meting gedaan is (bv. na 1 ms). Extra toggles bijvoegen aan de PRESCALER totdat deze een overflow geeft, en dan weet men dat 256 –N = inhoud van de PRESCALER was (N het aantal toggles tot overflow).

Een voorbeeldje om dit duidelijk te maken.
In fig.1

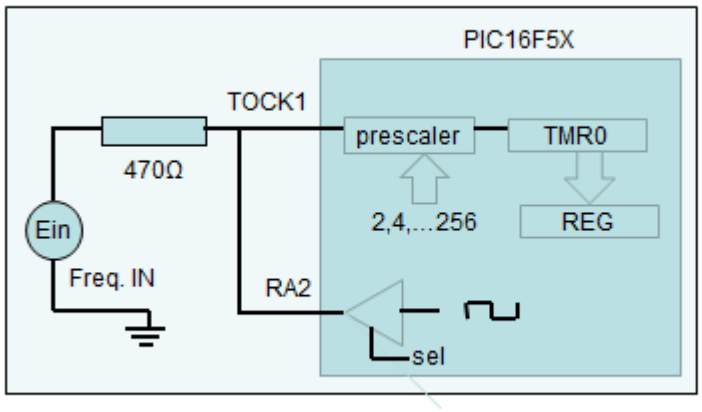

Fig.1

De frequentiegenerator is via 470Ω verbonden aan de inputs TOCK1 en RA2 van de PIC16F5X.
En veronderstel dat Fin = 21.586.244 Hz in 1 sec., en veronderstel dat de leestijd is 1ms, dan zijn er 21586,244 pulsen in RA2 aangekomen (TOCK1 is afgesloten).
De fractie van een impuls (nl. 0,244) wordt niet geteld en valt dus weg.

Software:
De PRESCALER van TRM0 staat op 256, en dus is er steeds een overflow –impuls van TRM0 naar TIMER0 na iedere 256 impulsen.
De TIMER0 staat na 1 ms. Op 21.586/256 = 84,320 pulsen. Ook hier valt de fractie na de komma weg en dus TIMER0 staat na 1 ms. Op 84.
Nu stopt de input naar RA2 door TOCK1 laag-ohmig te maken (ongeveer 50Ω)
Vermits RA2 steeds een 256 teller is staat er na 1 ms. de rest van 256 x N + R = 21.586 met N = 84 is R = 21.586 – 84 x 256 = 82 en dan is de gemeten frequentie gelijk aan N x 256 +R = 84 x 256 + 82 = 21.586 p/1 ms. .Dit is dus gelijk aan 21,586 MHz, en dus is dat een meting tot op 1 kHz nauwkeurig.

1.1 Uitlezen van TMR0

Als ik nu de uitgang TOCK1 (die na de meting op 0 level gezet wordt) gedurende een korte tijd (100 ns bv.) terug op 1-level zet (met een output van 50Ω, dan zal de output op TOCK1 eruit zien (zie fig.2))
Niettegenstaande dat de frequentiegenerator nog altijd blijft toggelen op 21.586 MHz zal dat weinig invloed hebben op de input RA2.
Immers als TOCK1 = 5V en frequentiegenerator = 0V dan staat op RA2 =(5V x 470Ω)/(470Ω + 50Ω) = 4,519 V en natuurlijk met TOCK1 = 5V is RA2 = 5V.
Dit geeft maximaal een rimpel van 0,5 V terwijl de input gate omslaat bij een spanning lager dan 1V, en naar level 1 gaat bij een level van 3 V.
Omgekeerd als TOCK1 op 0V staat en de frequentiegenerator = 5V dan zal op RA2 0,48 V maximaal staan.
Door voortdurend TOCK1 te toggelen tussen 0 en 5V worden er bij de teller TMR0 steeds pulsen bijgeteld, totdat deze de waarde van 256 bereikt heeft, en een overflow aan TIMER0 geeft. Dan stopt men met toggelen, en men weet dan dat 256 –N = inhoud van TMR0, waarbij N het aantal pulsen zijn die men er bij heeft gevoegd.

In ons voorbeeld stond na de meting van 1 ms de TMR0 teller op 84 en het aantal pulsen dat er nog in de PRESCALER zaten is 21.586 –(256 x 84) = 82 of anders gezegd het aantal pulsen N die men er nog moest bijvoegen om een overflow te krijgen is gelijk aan N = 256 – 82 = 174. En de totale frequentie is dan TIMER0 x 256 + (256 –N) = 84 x 256 + (256 – 174) = 21.586 MHz wat een nauwkeurigheid geeft tot op 1kHz.

2 De subroutine 1 ms timer

De meettijd van 1ms moet zeer nauwkeurig gemaakt worden door een willekeurig programma te maken dat juist zoveel instructies bevat zodat het totale programma 1ms duurt zonder dat het wezenlijk niets doet dan een wachttijd introduceren.

Wanneer de frequentie lager is dan 1 MHz dan moet de meettijd van 1 ms verhoogd worden tot 10 ms. En dus moet de delay van 1ms 10 maal herhaald worden.

Alle instructies in een PIC hebben een aantal klok impulsen nodig om een instructie uit te voeren. Eenvoudige instructies (zoals NOP) hebben 4 klokpulsen nodig, maar meer ingewikkelde instructies hebben 8,12 of 16 klokpulsen nodig (zoals GOTO heeft 8 klokpulsen nodig).
Met een kristal klok van 4 MHz heeft dus een NOP instructie 4/4MHz = 1µs nodig en GOTO 8/4MHz = 2µs nodig.
Een mogelijk programma (maar er zijn oneindig varianten mogelijk) is als volgt:

```
Begin     Movlw     .197      ;put 197 in work register
          Movwf     count     ; the contents is moved to “count”,
                              ; if data = 0 set status flag, else continue
          Nop
          Goto  $ + 1         ;go to next
          Goto  $ + 1         ;go to next
          Goto  $ + 1         ;go to next
          Goto  $ + 1         ;go to next
          Goto  $ + 1         ;go to next
          Goto  $ + 1         ;go to next

Dly1ms    Goto  $ + 1         ;go to next
          Decfsz    count,F   ;decrement (-1) the contents of register W,
                              ; if 0 then skip next instruction else continue
          Goto Dly1           ; jump to Dly1ms
          Retlw               ; end subroutine, return to mainstream
```

Een beetje uitleg. Iedere instructie wordt uitgevoerd met een aantal klokpulsen;

Movlw → 4cl → 1µs
Movwf → 4cl → 1µs
Decfsz → 4cl → 1µs
Nop → 4cl → 1µs

Goto → 8cl → 2µs

Het “Begin” programma duurt (1 + 1 +1 + 6* 2) = 15µs
En het Dly1ms subroutine, dat 197 wordt doorlopen, duurt (2 + 1 + 2)*197 = 985µs
Dit geeft een totaal van (985 + 15)µs = 1000µs = 1ms

Jan Spaenjers