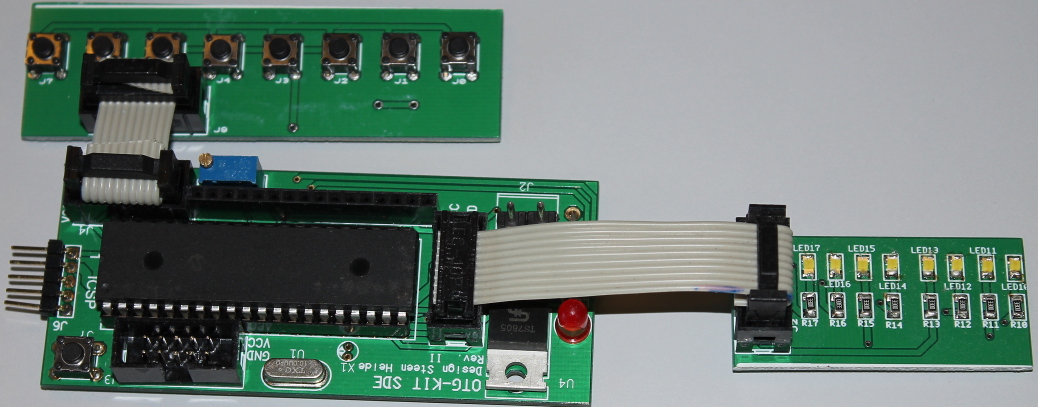
# Introduktion til øvelsen

I laboratorie øvelse 3 lavede vi delay funktioner, ved hjælp af flere løkker inden i hinanden, og spildte en masse instruktioner, det kan, i de fleste situationer, ikke svarer sig, fordi man bruger for mange af microcontrolleren’s kræfter på at udmåle en tid! Der findes imidlertid en anden metode, vi kan anvende når vi skal udmåle tid. Inden i vores Micro Controler ligger der tre timere, T0, T1, T2, de kan bruges til mange ting hvor vi skal udmåle tider eller tælle events. I denne øvelse skal vi arbejde med Timer2.

# Ekstra materiale til denne øvelse

Her får vi for første gang behov for databladet, dvs. jeg har tygget det meste. Men der vil være klip og register informer fra databladet. Husk registerinformationer skal med i jeres dokumentation når i har anvendt ”[special funktions registre](#_Special_funktions_registre)”  
det giver også mening at kikke i den fil, vi inkluderer i starten af alle vores programmer, [”p16f877a.inc”](#_p16f877a.inc) den er vi gået stille hen over i de foregående øvelser.

# Vores MicroControler, µC, system



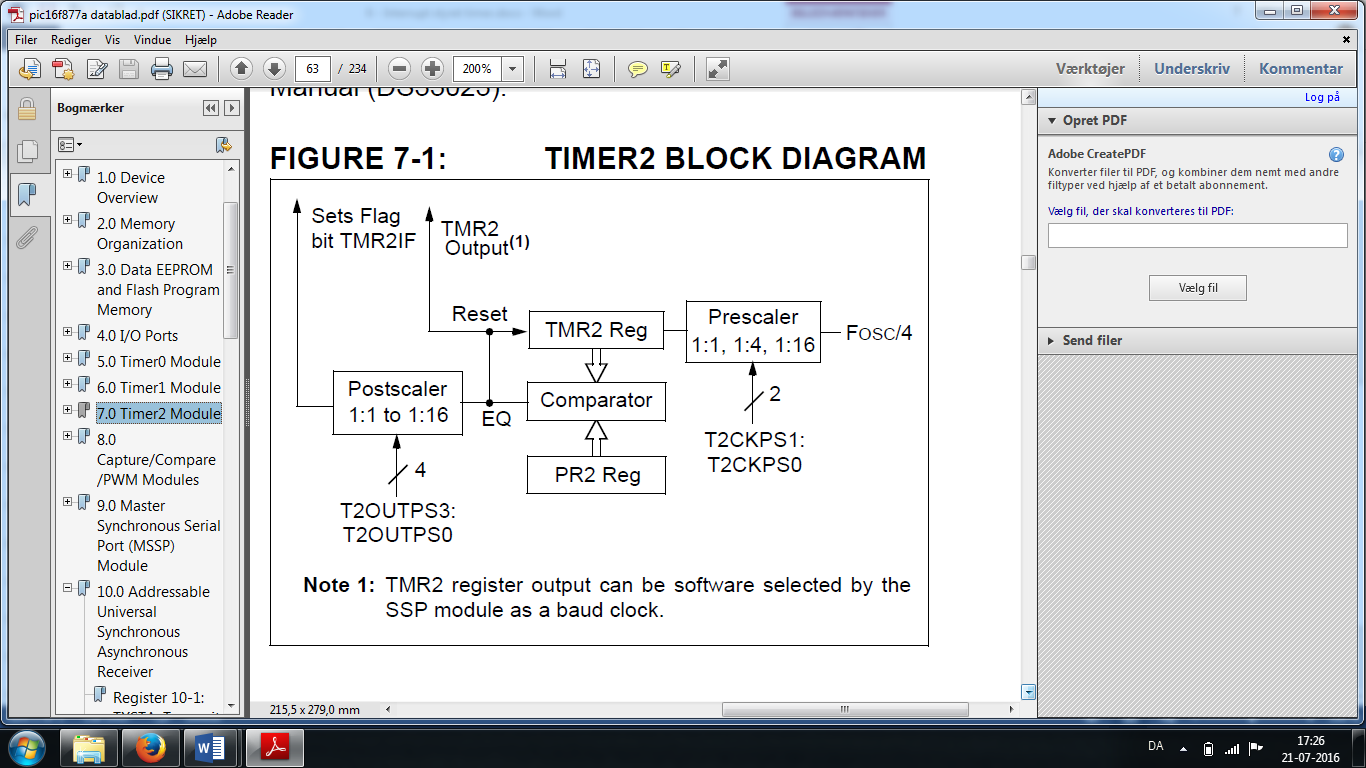
Figur 2: Vores OTGkit v2 med display keyboard og Dioder samt Programmeren

# 

# Timer2:

Timer2 er en af de 3 indbyggede timere i PIC16f877A. Den er 8 Bit, det betyder at den kan tælle til 255, da timer2 kun kan bruge clock frekvensen til, at tælle på betyder det at den har en indgang på (10\*106 Hz/4). Det betyder den burde tælle meget hurtigt, men heldig vis kan den sløves ned, med både en [prescaler](#_T2CON_(TIMER2_control) og en [postscaler](#_T2CON_(TIMER2_control) (se diagram på næste side). [Prescaleren](#_T2CON_(TIMER2_control) gør at påvirkningen fra clocken kan sløves med: 1, 4, eller 16 gange. [Postscaleren](#_T2CON_(TIMER2_control) kan vælges frit mellem værdien 1 og 16 gange. Det betyder at den længste tælleperiode vi kan opnå med timer2, er 255\*255/(10\*106 Hz/4), eller hvis vi sætter på ligning.

.



Timeren fungerer sådan, at hver gang [Prescaleren](#_T2CON_(TIMER2_control) er færdig med at tælle, tæller TMR2 registeret en op.  
Resultatet i TMR2 registeret sammenlignes nu i Comparatoren med værdien i PR2 registeret, [Postscaleren](#_T2CON_(TIMER2_control) tælles en op når værdierne er ens.  
Det hele gentages til både Pre- og Postscaleren er kommet til den indstillede værdi.

# LAB6\_1.ASM

Beregn med ligningen øverst på side1 hvad værdien af PR2 skal være, for at vi kommer meget tæt på 20mS.  
Indsæt værdien lige før ”movwf PR2” og kør øvelsen.  
se programmet i gennem læg specielt mærke til de linjer hvor programmet søger efter om timeren er blevet færdig med at tælle

[Læs afsnittet om Interrupts](#_Interrupt:)

# LAB6\_2.ASM

Hent og prøv ”LAB6\_1.ASM”   
programmet tæller dioderne på PORTC en op hver gang der er gået 20 ms

# LAB6\_3.ASM

Nu skal vi tilbage og have fat i lab3 øvelsen, du ved der hvor vi spildte tiden, men vi skal benytte det vi lærte til noget nyt, for nu skal vi have PORTC,0 til at skifte mellem tændt og slukket hvert halve sekund. Men du skal huske at vi skal have nulstillet Interruptet hver gang, lige som vi skal forlade Interruptet hvert 20mS.

# LAB6\_4.ASM

I “LAB6\_3.ASM” fik du dioden til at skifte status hvert halve sekund. Denne gang skal du lave en lysdiode der tænder når SW1 har været aktiveret i 1 sekund.

# LAB6\_5.ASM

Lav selv en ”LAB6\_5.ASM” med et ur der tæller sekunder, minutter og timer

# Interrupt:

Da timeren ligger udenfor selve Micro Controlleren, men stadig inden i vores 40 benede kreds, skal der et eller andet til for, at se at timeren nu er blevet færdig med at tælle. Måden det sker på er at der sættes et flag i et register, Et Flag er en enkel bit der fortæller om, i dette tilfælde, timeren er blevet færdig med at tælle. Men flag kan fortælle om en hvilken som helst status.   
Det flag der skal læses på i denne omgang er TMR2IF, eller når det læses bagfra, Interrupt flaget for timer2.

I Lab6\_1.asm aflæser programmet selv TMR2IF en gang i mellem, for at se om timeren er færdig. Man kan imidlertid også lade aflæsningen af TMR2IF ske automatisk, men så skal man som det sker i Lab6\_2.asm også ind og sætte en Interrupt styring op.

Når en Micro controller får et Interrupt, vil den:

1. Afslutte den instruktion den er i gang med
2. Gemme den øjeblikkelige Instruktionsadresse ”Værdien i program tælleren”
3. Hoppe til Interrupt vektoren (se næste afsnit)
4. Hoppe til Interrupt service rutinen (se næste afsnit)
5. Udføre Interrupt service rutinen
6. Læse RETFIE og returnerer til den linje den var kommet til, før der kom Interrupt

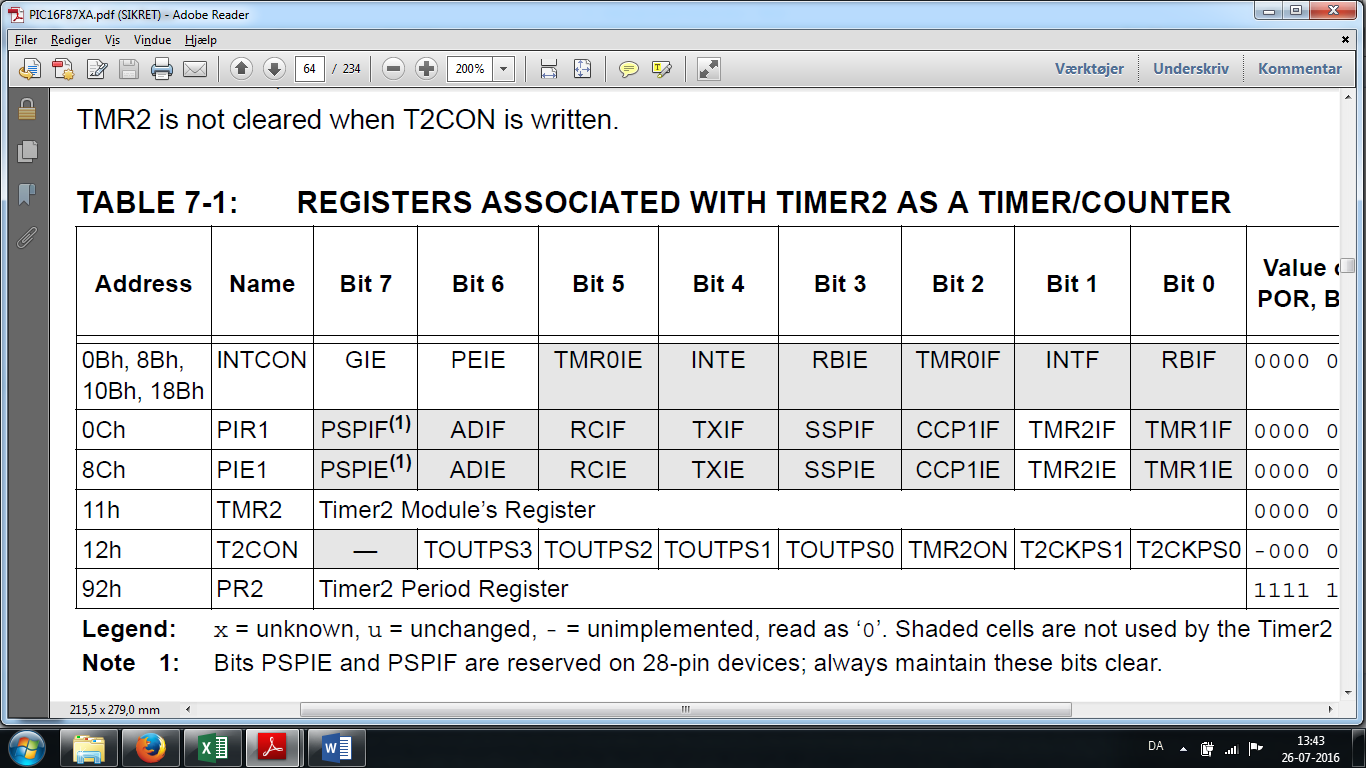
Når vi skal sætte Interrupt op, skal der ske noget 4 steder i programmet:

1. Normalt har vi set en linje ”org 0000” efterfulgt af ”GOTO init” i starten af programmet. Disse to linjer kaldes for en reset vektor, og aktiveres hvis der sættes strøm til kredsen, eller hvis nogen trykker på reset. Efter disse to linjer skal der være to nye linjer ”org 0004” efterfulgt af ”GOTO ISR”, det er Interrupt vektoren. Her står ISR for ”Interrupt Service Rutine” det er den procedure/metode der skal udføres når timeren er færdig med at tælle.   
   Det er vigtigt når du arbejder med Interrupt du får skrevet ”org 0004”. Micro controlleren vil nemlig, når den får et Interrupt, hoppe til programadresse 0004, her skal du henvise til det sted hvor tingene sker din ISR procedure
2. Det der skal aktiverer dit Interrupt skal sættes op, det vil i dette tilfælde sige timer2, og dens Interrupt.
3. Det er bare ikke nok, at opsætte timer2 controlleren skal også have at vide at den skal modtage Interrupts. Her skal sættes 2 bits i INTCON registeret nemlig GIE ”Global Interrupt Enable” og PEIE ”Peripheral Interrupt Enable”. Hvis ikke disse to bits er sat vil Controleren aldrig opfatte at timer2 er blevet sat
4. Sidst skal du konstruerer din ”Interrupt Service Rutine”. her du skal lave det program der udfører din opgave ISRen afsluttes med RETFIE, hvorefter programmet returnerer til det sted det var før der blev interruptet.

# 

# Registre:

I databladet er der en oversigt over alle de registre, der påvirker anvendelsen af timer2. der er en tilsvarende tabel i forbindelse med alle special funktioner der findes i vores controler, men i dette afsnit vil jeg kun lægge registre ind der hører til PIC16F877A der er den Controler vi anvender i øjeblikket , og i disse registre har jeg kun taget de bits med der er relevante for timer2, dvs. de hvide felter i den nedenstående tabel.



### INTCON registeret er præsenteret I alle 4 banker

bit 7 **GIE:** Global Interrupt Enable bit

1 = Enables all unmasked interrupts

0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit

1 = Enables all unmasked peripheral interrupts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TOUTPSnr | | | | POST |
| 3 | 2 | 1 | 0 | SCALE |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | 4 |
| 0 | 1 | 0 | 0 | 5 |
| 0 | 1 | 0 | 1 | 6 |
| 0 | 1 | 1 | 0 | 7 |
| 0 | 1 | 1 | 1 | 8 |
| 1 | 0 | 0 | 0 | 9 |
| 1 | 0 | 0 | 1 | 10 |
| 1 | 0 | 1 | 0 | 11 |
| 1 | 0 | 1 | 1 | 12 |
| 1 | 1 | 0 | 0 | 13 |
| 1 | 1 | 0 | 1 | 14 |
| 1 | 1 | 1 | 0 | 15 |
| 1 | 1 | 1 | 1 | 16 |

0 = Disables all peripheral interrupts

### PIR1 (peripheral interrupts registeret) ligger I bank0

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit

1 = TMR2 to PR2 match occurred (must be cleared in software)

0 = No TMR2 to PR2 match occurred

### PIE1 (peripheral interrupts enable registeret) ligger I bank1

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit

### TMR2 (Timer2 Module Register) ligger i bank0

Dette er det register timer 2 sammenligner, den øjeblikkelige tælle værdi med.   
Hele registeret tæller med da der er tale om en 8bits værdi (0-255)

### T2CON (TIMER2 control register) ligger I bank0)

bit 7 **Unimplemented:** Read as ‘0’

bit 6 **TOUTPS3**

bit 5 **TOUTPS2** Se den øverste tabel for

bit 4 **TOUTPS1** indstilling af Postscaler

bit 3 **TOUTPS0**

bit 2 **TMR2ON** Timer2 On bit

1 = Timer2 is on

0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

|  |  |  |
| --- | --- | --- |
| TOUTPSnr | | PRE |
| 1 | 0 | SCALE |
| 0 | 0 | 0 |
| 0 | 1 | 4 |
| 1 | 0 | 16 |
| 1 | 1 | 16 |

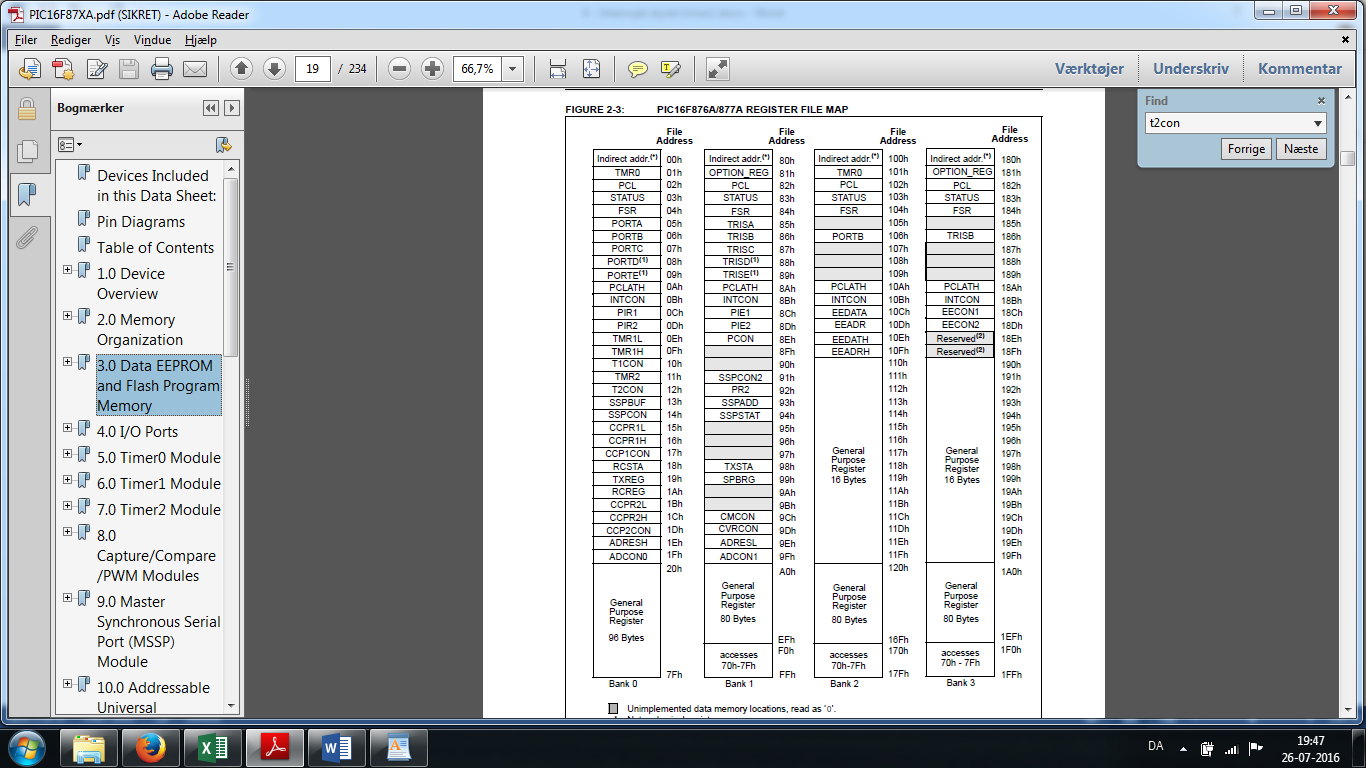
00 = Prescaler is 1

01 = Prescaler is 4 Se den nederste tabel for

1x = Prescaler is 16 indstilling af Prescaler

# Special funktions registre

Der er mange special funktions registre i PIC16F877A. De er organiseret i 4 banker, der er   
nummereret fra 0-3. De første 32 adresser i hver stak (00h-20h) er specialfunktions registre, de anvendes til at styre alt fra almindelige porte til avanceret kommunikation. Nogle af de enkle funktioner kan du læse om på en ½ time, andre som I2C har særskilte datablade på næsten 100 sider, guf for nogen men en dræber for andre.



# p16f877a.inc

Denne include fil har vi altid brugt, så længe vi har programmeret PIC controllere.  
Den gør at MPlab kan læse alle de forkortelser der står i manualen til PIC16F877A  
her er kun vist forkortelser for de registre vi benytter i denne øvelse men du kan   
selv tage et blik på ”p16f877a.inc”

Registrene og den adresse de ligger på, Det er når MPlab skal oversætte forkortelsen  
for registeret til adressen den anvender ”p16f877a.inc” filen

Som du kan se er de bit der er navngivet inden for et register, numre fra 0-7. Det betyder   
det reelt er ligegyldigt, om du skriver ”PIE1,TMR2” , ”PIE1,T0IF” eller ”PIE1,2” men den   
der skal læse programmet efter dig, vil nok helst have du skriver ”PIE1,TMR2” da det   
giver mest mening.

INTCON EQU H'000B'

PIR1 EQU H'000C'

TMR2 EQU H'0011'

T2CON EQU H'0012'

PIE1 EQU H'008C'

PR2 EQU H'0092'

;----- INTCON Bits ------

GIE EQU H'0007'

PEIE EQU H'0006'

T0IE EQU H'0005'

TMR0IE EQU H'0005'

INTE EQU H'0004'

RBIE EQU H'0003'

T0IF EQU H'0002'

TMR0IF EQU H'0002'

INTF EQU H'0001'

RBIF EQU H'0000'

;----- PIR1 Bits --------

PSPIF EQU H'0007'

ADIF EQU H'0006'

RCIF EQU H'0005'

TXIF EQU H'0004'

SSPIF EQU H'0003'

CCP1IF EQU H'0002'

TMR2IF EQU H'0001'

TMR1IF EQU H'0000';----- PIE1 Bits --------

PSPIE EQU H'0007'

ADIE EQU H'0006'

RCIE EQU H'0005'

TXIE EQU H'0004'

SSPIE EQU H'0003'

CCP1IE EQU H'0002'

TMR2IE EQU H'0001'

TMR1IE EQU H'0000'

;----- T2CON Bits -------

TOUTPS3 EQU H'0006'

TOUTPS2 EQU H'0005'

TOUTPS1 EQU H'0004'

TOUTPS0 EQU H'0003'

TMR2ON EQU H'0002'

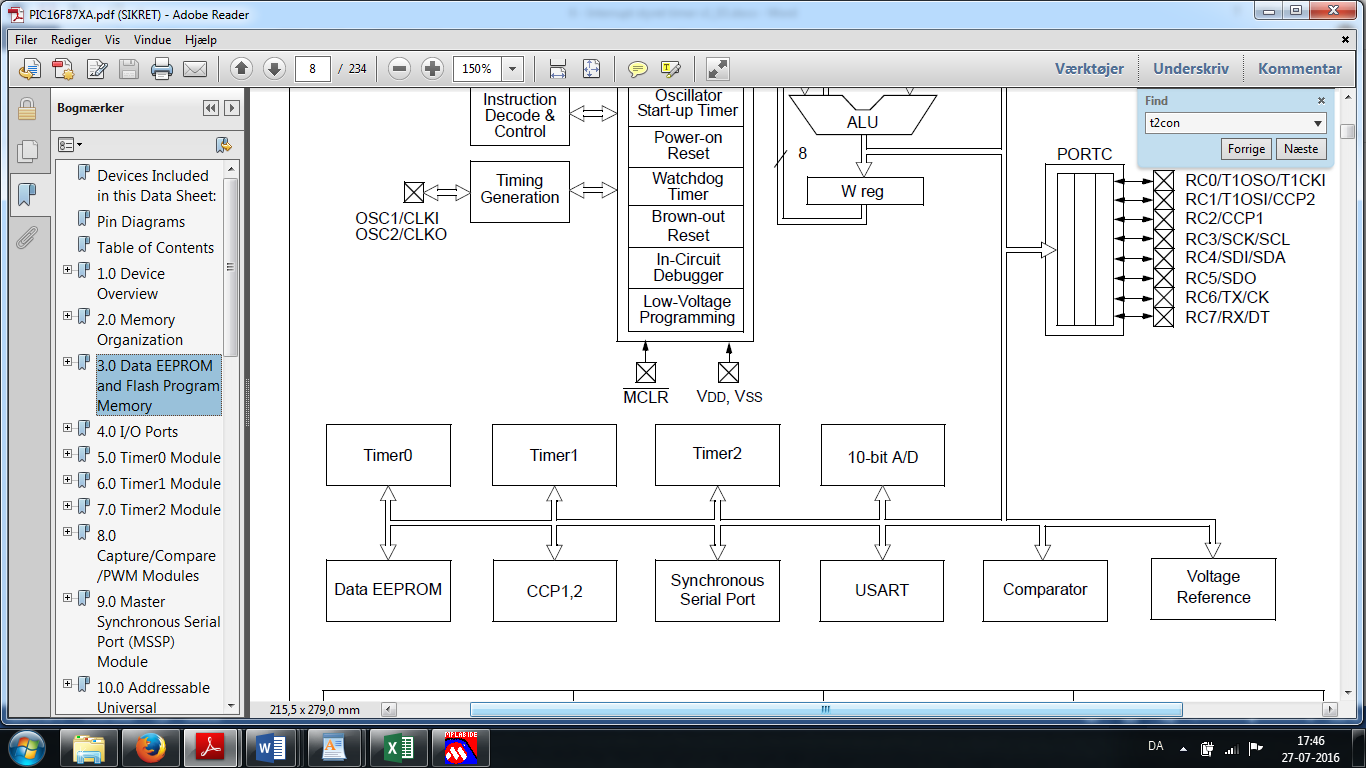
T2CKPS1 EQU H'0001'

T2CKPS0 EQU H'0000'

Da TMR2 og PR2 er fulde registre og ikke indeholder flag, er der ingen navngivne bits til dem

# Special funktioner

Der er mange specialfunktioner I PIC16F877A. dem jeg kikker på her er de perifere funktioner der ligger uden for selve µC kernen, det er de bokse der optræder nedenfor



# Timer0

Timer0 er den simpleste timer I PIC16F877A, Den køre I princippet altid, man skal bare vælge at bruge den. Lige som Timer2 er den 8 bit.  
Man kan vælge om man vil lade den tælle op fra en ekstern kilde, eller om det skal være fra krystallet. prescaleren kan gå fra 0 til 255

# Timer1

Timer1 er en 16 bit timer, den kan anvendes som timer eller som tæller, eller sammen med (CCP1,2) som PWM der er også et par andre funktioner jeg ikke tager med her. Timer1 kræver lidt læsning i databladet, men så kan den også give valuta for pengene.

# Timer2

Den har vi set på men timer2 kan også anvendes til flere ting bl.a. anvendes den også når man komunikerer via seriel. Her danner den baudraten, den hastighed man sender sine data med.

# 10-bit A/D

Der er 5 indgange til den Analog til Digital Converter der sidder i PIC16F877A, de er placeret på PORTA, men vi bruger som regel kun den indgang der sidder på PORTA,0. A/D converteren i vores µC er 10 bit men det er rigtig sjældent vi anvender mere end de 8bit det er så lidt med forskel om vi skal bruge de mindst betydende (højre justeret) eller de Mest betydende (venstre justeret) men det kan du læse mere om i datablaset.