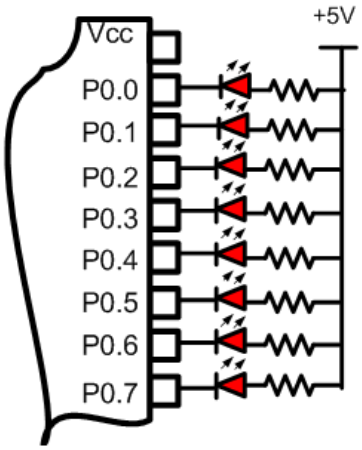
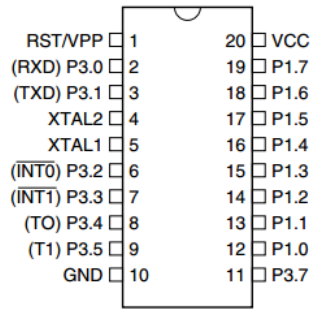
Her er en række opgaver til at lære Assemblerprogrammering.

Den første opgave går ud på at lave et program der får LED`ene et LED-kit til at lyse i flg. mønster, idet et X indikerer en tændt diode.

Fordi denne microcontroller kun kan Sink'e strøm, er dioderne monteret fra + 5 Volt og ned mod port P1!

Herunder er gengivet forskellige mønstre, der ønskes vist på lysdioderne.





Pinouts for AT89C4051

MSB	Opgave 1, Løbelys						LSB
							x
						x	
					x		
				x			
			x				
		x					
	x						
x							
							x
						x	
					x		
				x			
	Osv.						

MSB	Opgave 2, Løbelys med 2 Led						LSB
						x	x
					x	x	
				x	x		
			x	x			
		x	x				
	x	x					
x	x						
	x	x					
		x	x				
			x	x			
				x	x		
					x	x	
						x	x
					x	x	
					x	x	

MSB	Opg. 3, Vis det binære talsystem						LSB
							x
						x	
						x	x
					x		
					x		x
					x	x	
					x	x	x
				x			
				x		x	
				x		x	x

MSB	Opgave 4,						LSB
x							x
	x					x	
		x			x		
			x	x			
			x	x			
		x			x		
	x					x	
x							x
	x					x	
		x			x		

Alle mønstre fortsætter deres cyklus og starter forfra !!

Obs. Dioderne er sat fra +5V ned til porten. De lyser altså på et 0 !!

Strategi:

Først overvejes, hvad programmet skal:

- Sæt et bit-mønster ud på porten,
- Vente en tid,
- Sæt et nyt mønster ud på porten
- Vent igen en tid
- Osv.
- Efter alle mønstre er "udført", hoppes til starten igen!

Ventetiden udføres i en subrutine, kaldet "Pause" – eller Delay

Programeksempel:

```
        Org 0h                  ; Programmet skal ligge fra adresse 0 i Rom'en
Start:
        Mov p1, #11111110B      ; flyt en værdi på port 1
        Call pause              ; kald subrutinen Pause
        Mov P1, 11111101B       ; sæt en ny værdi ud
        Call pause

        Osv

        Jmp Start               ; hop op til label Start
;-------------------------------------------------------------------
; Subrutine Pause
; optager processoren ca. ¼ sek.
; ændrer værdier i reg R7 og R6
;
Pause:  mov R7, #0FFh           ; læg værdien FFh i register R7
Pause1: Mov R6, #0FFh           ; Do i reg R6
Pause2:
        Djnz R6, Pause2         ;Decrement R6, og hop hvis ikke zero til label Pause2
        Djnz R7, Pause1         ; som do, men til label Pause1
```

```
        Ret                     ; returner fra pauseprogrammet
;-------------------------------------------------------------------------------
        End
```

Programeksempel 2, gem værdi i variabel.

Strategi: Det mønster, der skal ud på porten, gemmes i en variabel, dvs. et register

```
        Org 0h

Start:  Mov a, #11111110B       ; Gem værdi I hukommelse Acc. ( Accumulator )
Start1: Mov P1, A
        Call pause
        RL A                    ; roter A-registeret til venstre
        Jmp Start1
```

Programeksempel 3

Programmet skal køre 8 løkker, hvor lyset bevæger sig til venstre, derefter 8, hvor det bevæger sig mod højre. Derefter forfra, Ala ”Knight Rider”

Strategi: Det mønster, der skal ud på porten, gemmes i en variabel, dvs. et register. Et tal i et register holder styr på hvor mange gange, der er tilbage

```
        Org 0h
Start0: Mov R0, #8d             ; R0 er en “tæller”, der har værdien 8
Start:  Mov a, #11111110B
Start1: Mov P1, A               ; send ud på port 1.
        Call pause
        RL A                    ; roter a registeret til venstre
        Djnz R0, Start1         ; Tæl R0 1 ned, og hop hvis ikke lig 0
;
        Mov R0, #8d
        Mov a, #01111111B
Start2: Mov P2, A
        Call pause
        RR A
        Djnz R0, Start2         ; Tæl R0 1 ned, og hop hvis ikke lig 0
        Jmp start0
```

Look up Table

Lav nu et program, til samme kit, der viser alle forrige 4 opgaver gennemløbet 1 gang hver.

Programmet skal udføres, så det er en lille løkke.

Lysdioderne på port 1 styres vha. Tabelopslag. Dvs. at i en tabel er gemt de data, - eller mønsteret, der skal sendes ud på porten.

Øverst i kildeteksten er løkke-programmet. Dernæst pauserutinen, og sidst tabellen.

Tabellen ligger altså i ROM, og defineres i kildeteksten med:

```
Tabel:  db 01h                  ;( "Tabel" er blot en label, "db" betyder Define Byte )
        db 02h, 04h, 08h
```

- osv.

Data hentes til Acc vha. en datapointer. Det er en 16 bit register, der "peger" på en adresse i ROM'en. Den bringes til at pege på tabellen med

```
        Mov dptr, #Tabel        ; få datapointeren til at pege på "Tabellen"
```

Data kan hentes fra tabellen til register A fra tabellen vha. følgende ordre:

```
        Mov a, #00h
        movc a, @a + dptr       ; hent tabelværdi
```

A-registeret indeholder før kaldet en værdi, der lægges til datapointeren før der hentes en værdi. Altså et offset. Dvs. næste tabelværdi kan hentes ved at øge værdien i A, eller ved at øge dptr.

Tegn flowchart !!

Lav program

Program-udbygning:

Sidst i tabellen anbringes værdien 0ffh. Og programmet indrettes således, at når en værdi er hentet til Acc, tjekkes den, og hvis lig 0ffh, hoppes til start igen !!

Ekstra 2:

Imellem "opgaverne" skal der være 3-dobbelt pause. (Kald pauserutinen 3 gange.)

Dette skal udføres, hvis der i tabellen hentes værdien 0FEh.

7-Segment Kit

Der skal skrives et program til 7-segment-kittet.

Først laves en konverter-tabel, en 4 bit til 7-segment – konverter. Den skal kunne vise fra 0 til F.

Kittet har forbundet lysdioderne a til f i 7-segmentet til port 1 som følger:

Tabel:

```
;     afecdgb*   ( P1.0 ikke forbundet, P1.1 forbundet til b, P1.2 til g osv. )
db 11111010b      ; Viser et 0-tal
                  ; db betyder Define Byte
```

Derefter skrives 4 små programmer:

1: Skriv et program, der tæller langsomt op fra 0 til 9, eller fra 0 til F, - hvorefter der startes forfra.

2: Skriv kode, så der via et tastetryk genereres et tilfældigt tal fra 0 til 9.

Lav koden om, så det virker som en elektronisk terning

3: Skriv kode, der tæller og viser antal tastetryk. Bemærk, at der skal tages højde for kontaktprel.

4: Vi skal have fundet et flowchart-program – tegn / konstruer
Se mit kompendium !!

Stepmotor styring

Lav et program til styring af stepmotor-kittet !!

Der skal laves en knap, så der kan bestemmes, hvilken vej, motoren skal køre!

Stepmotor !! Hastighed afhængig af kontakt, op/ned-knap, - og direction er afhængig af en anden knap.

Først laves programmet bare, så motoren kører

Programmet skal følges af et flowchart.

Nogle bør se lidt på Nassi Schneidermann metoden!
Øvrige: "Normale metode" !!

Når motoren kører, skal der tilføjes en knap, der ændrer omløbsretningen

Dernæst skal der bygges 2 knapper på, så der kan justeres op / ned for hastigheden

Kittet bruger en ULN2003. Undersøg dens datablad !!

Elektronisk terning

Lav et program, der styrer 7 lysdioder, og fungerer som en elektronisk terning.

Opbyg på fumlebræt.

Blandede Boolske logik-elementer

Lav et program, der på udgang P1.0 er en AND-gate funktion af P3.0 og P3.1, samtidig med at udgang P1.1 er en NANDgate-funktion af P3.2 og P3.3, og samtidig med at P1.2 er en NOR-funktion af P3.1 og P3.3

Tegn opgaven med bolske gates, Lav et program, og afprøv på fumlebræt.
Husk at udgangene ikke kan Source strøm!

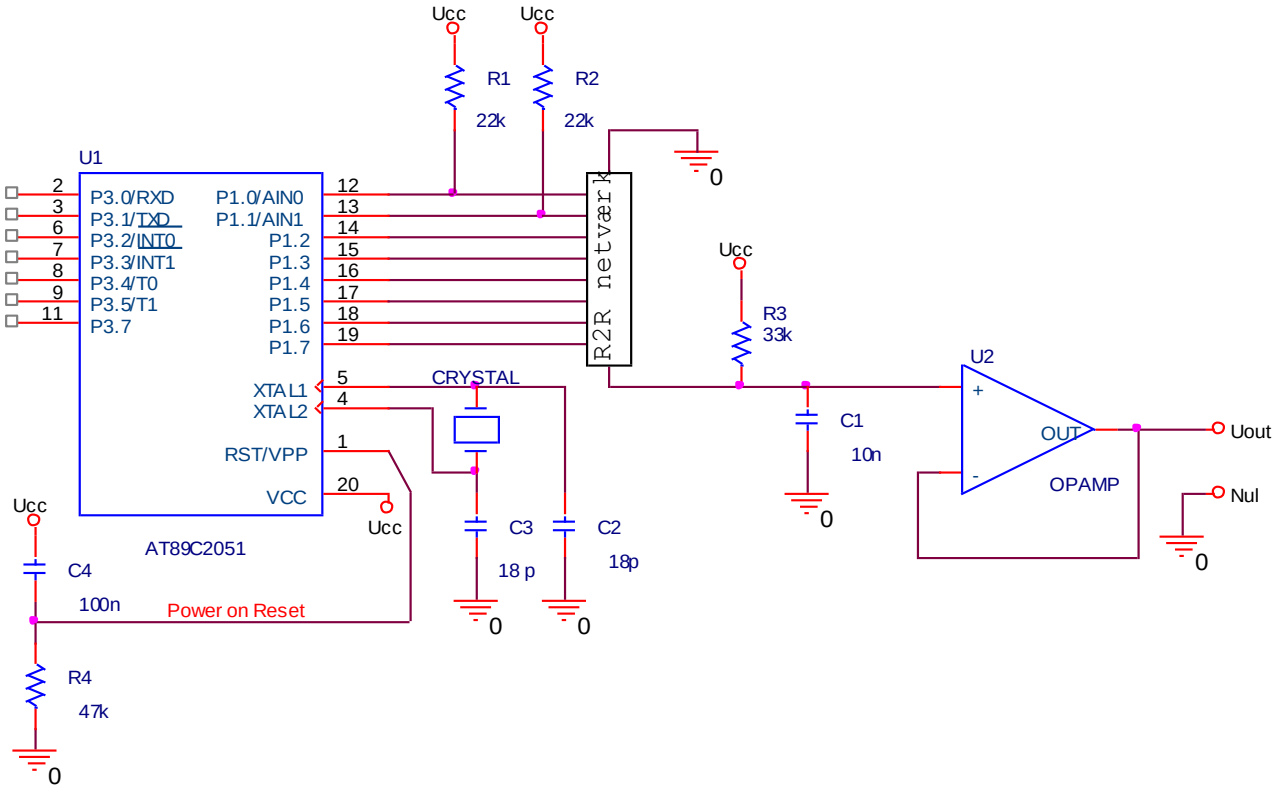
Signalgenerator

På port P1 monteres et R2R-netværk.

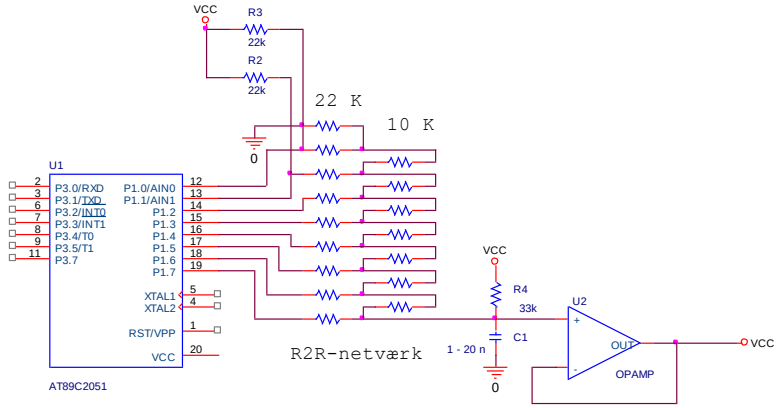
Send værdier ud på porten P1, og mål med scoopet, det frembragte signal.

Tæl fx binært op !!

Der skal 22 K pull up modstande på P1.0 og P1.1



R2R-netværket kan også opbygges af diskrete modstande som vist i det følgende eksempel:



Orgel:

Der konstrueres et lille program til orgel-kittet.

Find ud af, hvilket ben, der er output.

Hvad skal der ske på outputtet ??

Lav flowchart og lav lille test i en sub-rutine. Afprøv !!

Udvid flowchartet. Læs så input fra tastaturet, og send så et signal afhængig heraf ud til lydgiveren.

Stopur på 4 x 7segment-kit

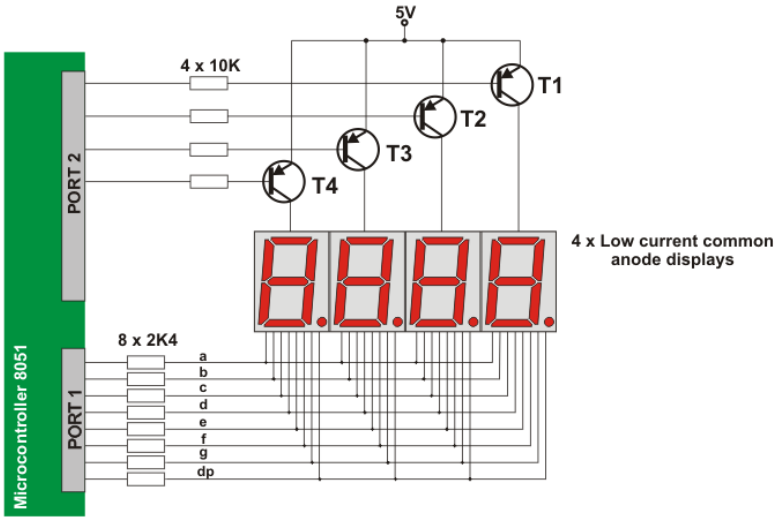
I denne opgave skal der arbejdes med **timer-interrupts.**

Der skal laves et stopur med 7-segment-kittet. Det skal kunne tage tid på 1/100del sekund.

De 4 7-segmenter styres vha. multiplexing. Dvs. at der først sendes data ud til segmentet længst til højre. Herefter tændes denne i en periode, og slukkes igen. Herefter sendes data ud til nr. 2, som så tændes osv.

Her er vist et eksempel på opbygning af et multiplex-system:

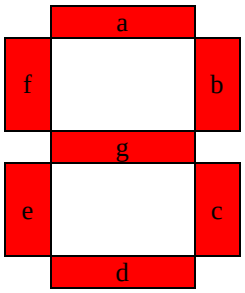
I dette diagram er der brugt port 1 og 2. I "vores uC " – AT89C4051 skal vi bruge port 1 til at bestemme tal-mønstret, og port 3 til at vælge hvilket 7-segment, der skal være tændt.



Følgende skitse viser Pin-belægning på stopurs-trænekittet:

Her til højre vises benævnelserne på 7-segmenterne. Lille "a" er segmentet øverst, osv.

I skemaet ses, hvilke bit i port 1, der er koblet til hvilket segment. Hvis pågældende portbit bliver lav, - og digit-bittet i port 3 også er lav, tændes segmentet.

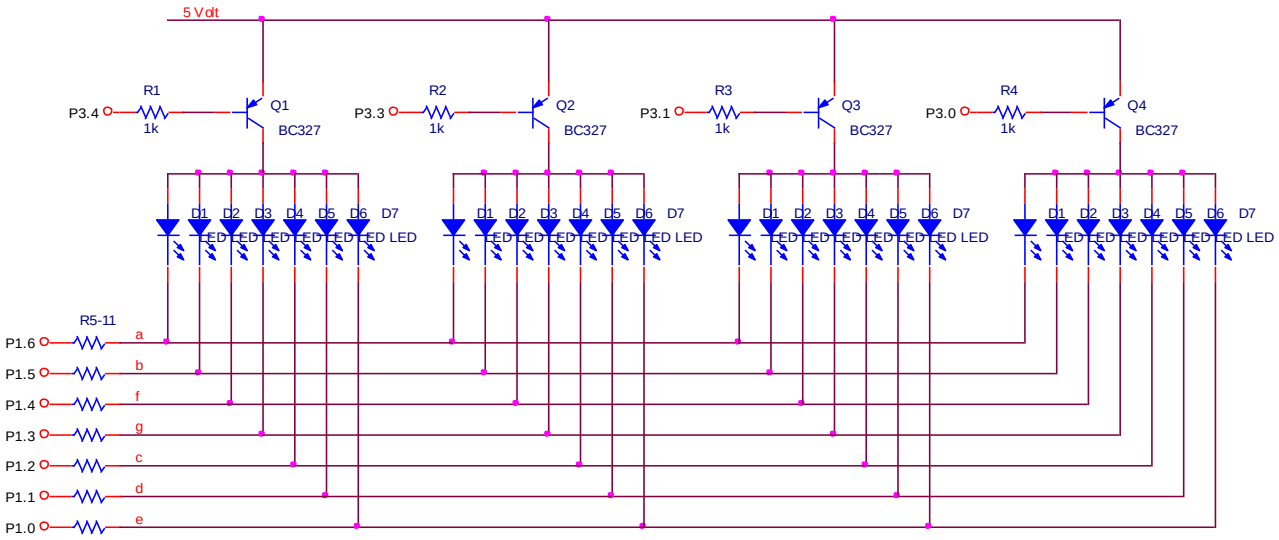


P1.7	nc.
P1.6	a
P1.5	b
P1.4	f
P1.3	g
P1.2	c
P1.1	d
P1.0	e

Hvilke af 7-segmenterne, der skal være aktive, styres af port 3 bit p3.0 til bit p3.3. Bemærk, at 7-segmentet er aktive på et lavt styrebit, som også ses på følgende diagram:

P3.7	
P3.6	
P3.5	
P3.4	digit 4
P3.3	digit 3
P3.2	
P3.1	digit 2
P3.0	Digit 1 længst til højre

Flg. skitse viser princippet i styringen af lysdioderne i 7-segmenterne.



De tal, der skal skrives i displayet, skal ligge i nogle RAM-adresser i controlleren. Ramadresserne kaldes også registre.

I registrene opbevares den aktuelle tid, dvs. værdier fra 0 til 9 (decimal).

Tallene kan ikke bare direkte sendes ud til 7-segmenterne. De skal først omkodes til et mønster, der for os viser det rigtige tal. Omkodningen sker vha. tabelopslag.

De indbyggede timere

Tælleren i uC'en får krystallets frekvens delt med 12

Krystallets frekvens kan ledes ind i en 16-bit tæller.

Når tælleren laver et overflow, kan der udløses et interrupt, og tiden skal tælles op.

Det skal ske hver hundrededel sekund.

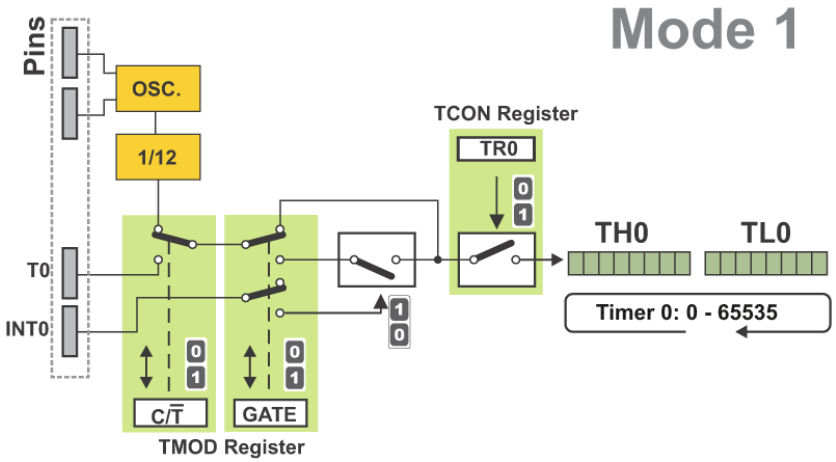
Hvis startværdien er rigtig, går der 1/100 Sekund inden der laves overflow, og der sendes en mente.

Tallet skal være FFFFh - 10.000d

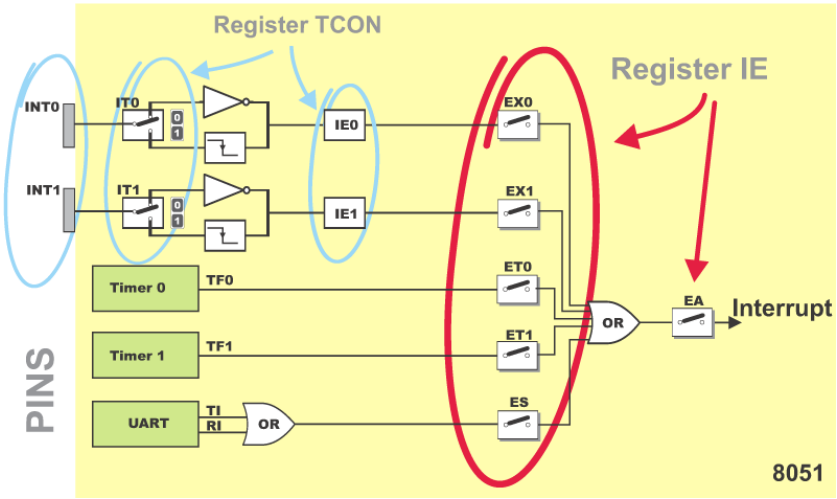
Blokstruktur af timerne:

uC'en har 2 indbyggede tællere, timer 0, og timer 1.

Deres opbygning og funktion vises her:

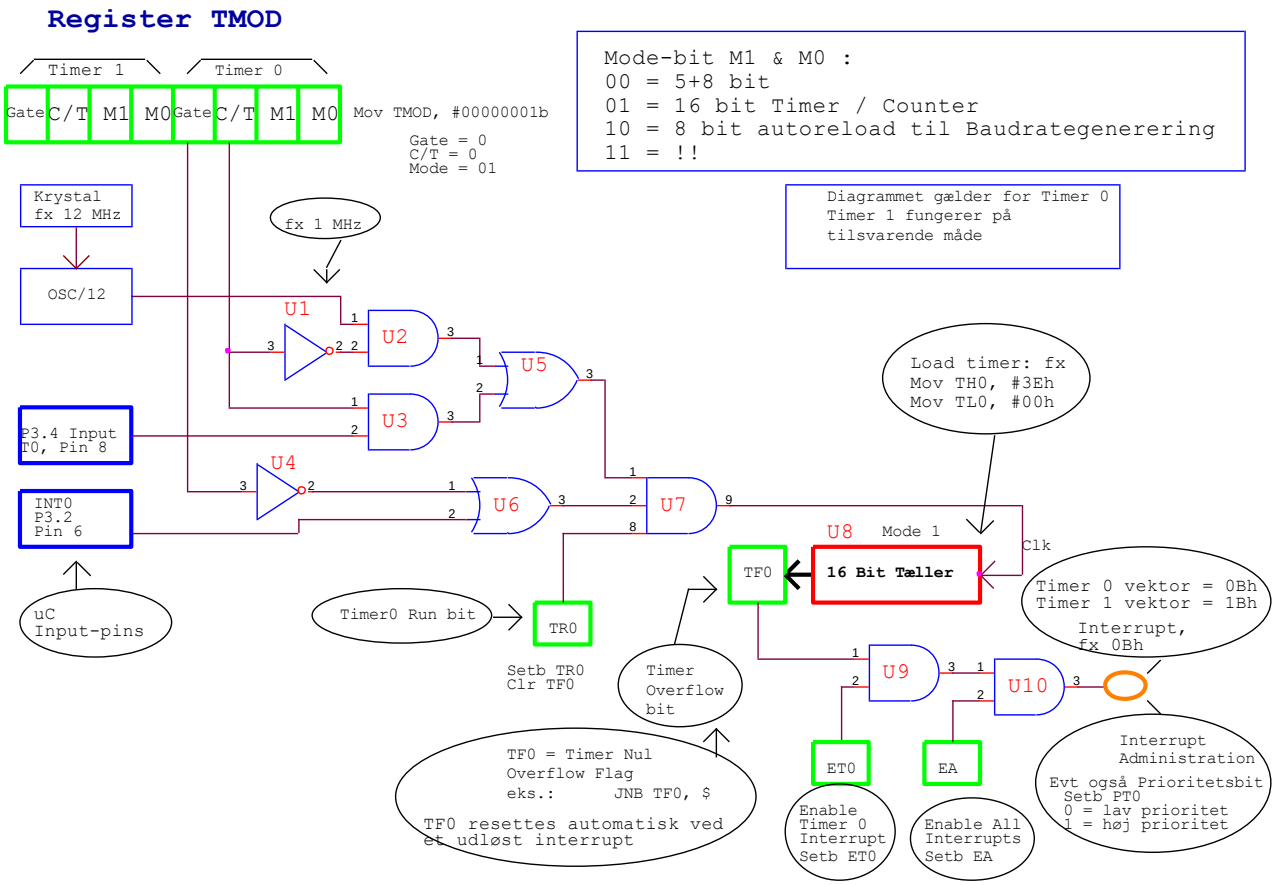


Interrupt-strukturen kan visualiseres sådan:



Kilde: http://www.mikroe.com/chapters/view/65/chapter-2-8051-microcontroller-architecture/

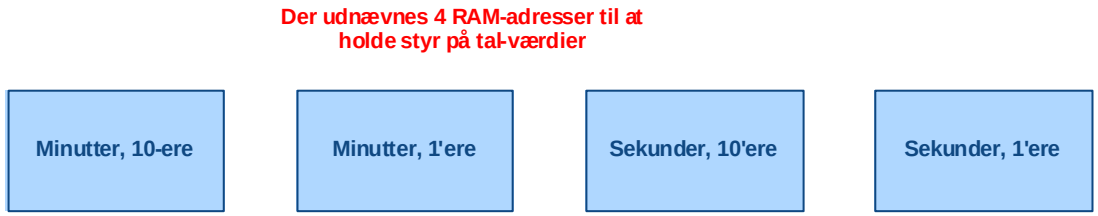
Og følgende den måde jeg har tegnet systemet:



Tilbage til opgaven:

Vi vil bruge en af de 2 indbyggede tællere, tæller 0, og indstille den til at tælle krystallets clockfrekvens. Det er altid 1/12 del af clockfrekvensen, der kan tælles !!

Tiden holdes der rede på i 4 RAM-adresser



Tælleren kan indstilles som en 16 bit tæller. Tælleren er altid en ”op-tæller”. Den kan loades med en startværdi, således at der går 1/100 del sek. før der sker et overløb.

Overløb vil sige at tælleren når til FFFF + 1, så der overføres en mente. Denne mente eller overløb kan bringes til at udløse et interrupt.

Et interrupt afbryder det igangværende program, udfører en subrutine, og returnerer igen til det igangværende program. Noget svarende til at et telefonopkald vil udløse et interrupt midt i madlavning !!

Interrupt-subrutinen skal nu incremente, dvs. tælle nogle registre 1 op.

Registrene kan passende navngives fx hundel, tidel, sek og tisek. Er der tale om fx et stopur, skal subrutinen også ud over at tælle 1 op holde øje med, om hundel er blevet større end 9. Ligeledes tidel osv. Endvidere skal tælleren genloades med startværdien !!

Altså efter start, er værdierne i de 4 registre lig den tid, der er gået, med en nøjagtighed på en hundredel sekunder.

Men værdierne i RAM-adresserne skal jo også vises på displayet. Det kan hovedprogrammet passende klare.

Processoren kan kun vise et tal ad gangen. Der er ikke ben nok !! Derfor skal der multiplexes. Dvs. der vises et tal, fx det højreste 7-segment, et stykke tid, derefter det næste 7-segment, osv.

Efter det sidste segment er vist, startes forfra.

Hvis det blot går hurtigt, dvs. mere end 25 omgange pr sekund, ser øjet ikke, at displayet blinker.

Det mønster, der skal sættes ud til et 7-segment, svarer jo ikke direkte til værdierne i RAM-adresserne. Er der fx værdien 5 decimal i sekund-registret, skal de segmenter i 7-segmentet tændes, der fremstiller et digitalt 5-tal. Der er altså behov for en tabel, der kan ”omdanne” en værdi til et mønster.

Tabellen laves vha. sandhedsskema.

Tabellen skal laves sådan, at der i tabellens element 0 er gemt et mønster, der på 7-segmentet vil vise et 0-tal. I tabellens element 1 findes et mønster for et 1-tal osv.

Programopbygning:

```
/*
Header

Programmet er til ---

Lavet af:

Dato:

*/

; Ramdefinitioner

        Hundel  equ     10h     ; register Hundel ligger på adresse 10h i RAM
        Tidel   equ     11h     ;
```

```
            Sek         equ         12h
            Tisek       equ         13h

; Pindefinitioner

            dighundel   equ         p3.0
            digtidel    equ         p3.1
            digsek      equ         p3.3
            digtisek    equ         p3.4

            org 0h
            jmp start               ; H


            org 0bh                 ; her kommer programmet, hvis timer 0 udløser et interrupt
            jmp timer0int           ; hop til interruptprogram

            org 30h
start:                              ; her starter programmet

            mov sp, #30h            ; stackpointeren er default 07h, men vi bruger jo ram over 07h

            mov dptr, #tabel        ;

; Først nulstilles register-værdier

            mov hundel, #00h
            mov tidel, #00h
            mov sek, #00h
            mov tisek, #00h


; opsætning af tæller:


; opsætning af interrupt



; her starter multiplex-løkken


;-----------------------------------
; Interrupt-rutine

Timer0int:
                        ; timeren skal have genloadet værdier, og den skal tælle registre op.



            Reti                    ; retur fra timer interruptprogrammet
;---------------------------------------


Tabel:
;               xabfgcde            ; sådan er bittene forbundet til segmenterne
            db 00001000b            ; Segmenterne tænder på et 0
            db
            db  osv


-------------------------------------------------
```

Foruden ovennævnte forbindelser til microcontrolleren er der ført følgende pins ud til kontakter:

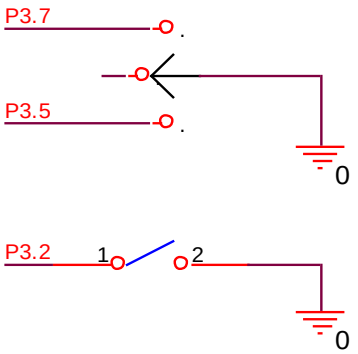
Undersøg "Timer-indmaden" og find ud af, hvordan, der kan laves start / stop med P3.2.

Udvid programmet med mulighed for at vise LAP-time.

Strategi:

Kopier tids-raméne til nye Ram-adresser, der så vises på displayet !!

Obs: Kontakten længst til venstre på kittet er RESET !! den får programmet til at starte forfra !!



Strategi: Fx:

Der defineres 4 nye RAM-adrssser til Tids-kopier.

Der defineres et flag, som sættes hvis P3.5 er lav. Og den cleares, hvis P3.7 er lav

Lab_Flag EQU 0 ; ligger fysisk på Ram-adr 20.0

;----

Sidst i Interruptrutinen, testes P3.5.

```
JB P3.5, Timer0int_10
JB Lab_Flag, Timer0int_10      ; Der laves et flag, der fortæller, at der er lavet en
                               ; Kopi.  Kan også bruges i hovedrutinen, til at hoppe
                               ; til en ny Løkke, der viser de nye RAM-adresser
Call Kopier
Setb Lab_Flag
```

Timer0int_10:

;-----

I hoved-mux-løkken testes for Lab_Flag, og hvis sat, hop til en ny Mux-løkke.

I den nye Mux-løkke testes for P3.7, og hvis lav, så Clr Lab_Flag, og hop til den anden Mux-Løkke.

LCD-Karakter-modul

Download evt. et kompendium fra min hjemmeside om LCD-karakter-moduler.

Det er lidt kompliceret at lave et program til et LCD-modul fra Scratch. Men der kan hentes hjælp fra en "kodegenerator", fundet på en tysk hjemmeside.

Generatoren genererer forskellige subrutiner, der kan kopieres ind i ens kildetekst. Subrutinerne kaldes så og bruges af det program, man selv skriver.

Flowchartet på en af de efterfølgende sider viser de subrutiner, der genereres af "kodegeneratoren" til LCD-kode..

Generatoren findes på min hjemmeside. Elektronik > Atmel 89C2051 > Tyvstjålet kodegenerator.

OBS ! Generatoren er skrevet til et display med 2 gange 16 karakterer.

Jeg har også lagt noget kode ud på min side. Det er også baseret på kodegeneratoren! Det kan hentes og bruges. Der er her blandt andet subrutiner til at positionere cursoren i displayet.

Studer evt. kodegeneratoren. Generer en kode, og kopier den til kildeteksten i Keil... Studer koden og sammenhold den evt. med efterfølgende flowchart.

Øverst til venstre på en af de følgende siders flowchart's ses hovedprogrammet, der kalder en initieringsrutine. Denne er nødvendig for at få liv i LCD-en. Herefter indstilles Datapointeren på en tabeladresse, og der kaldes en subrutine, der printer en "string" af ASCII-karakterer. Rutinen undersøger karakteren i tabellen. Når der læses et 00h, stoppes.

De øvrige rutiner, der er vist er:

Printc	; Sender indholdet i Acc til LCD-en, der skriver mønsteret i displayet der hvor cursoren står. Cursoren går 1 frem og er klar til næste karakter[1]
LCD_Return_Home	;Cursoren til øverste linie, 1. plads.
LCD_Clear	; Sletter alt i cursoren, og anbringer cursoren i første linie, 1. pos.

[1] Det der egentlig sker, når der sendes en værdi til LCD-en er følgende:

Til hver plads, eller karakter i LCD-en er der tilknyttet en RAM-adresse, også i LCD-en. Den værdi, der sendes fra Microcontroleren, gemmes i RAM-en på pågældende plads. Værdien, dvs. tallet, i RAM-en peger nu på en ROM-adresse i LCD-en, og i LCD-ens ROM er der så et mønster, der giver et bogstav i displayet på pågældende plads.

Det er indrettet således, at sendes en ASCII-værdi for fx et Y, vil netop et Y skrives i displayet. Men der er i ROM'en fx også japanske tegn. Se databladet for detaljer. I ROM-en, fra adresse 0 til 7 er der i stedet RAM, hvori der fra Kontrolleren kan downloades fx specielle danske tegn. Dette kan kodegeneratoren også håndtere.

LCD_Send_b ; Send ”Befehl” = Ordre til LCD-en. I Acc er anbragt den kommando, der skal sendes.

LCD_Prints ; Sender en string af hex-værdier, angivet i tabel, afsluttet med 0h

For en 16x2 display er RAM-en i displayet indrettes således, at øverste linies RAM-adresse starter i adresse 00h. 2. linie starter i RAM-adresse 14h

For et 20x4 display er det lidt anderledes:

1. linie: RAM-adresse 00
2. linie: RAM-adresse 40h
3. linie: RAM-adresse 14h
4. linie: RAM-adresse 54h

På nettet har jeg lagt subrutiner til at positionere Cursoren i displayet !

En subrutine til at positionere cursoren kunne se således ud:

; Værdi i reg A (1 til 20d) angiver positionen på linien. Kald til Lin2 med 3d i reg Acc får så cursoren til 3. plads i linie 2.

```
; ------------------------------------------------------------------------------------
; Subrutine til at positionere cursoren.
; Fx Mov a, #03h, Call Lin3 positionerer cursoren i linie 3 pos. 3.
; Indhold i Acc angiver pladsen på linien.

Lin1:   add a, #00h
        Jmp Linn
Lin2:   add a, #40h
        Jmp Linn
Lin3:   Add a, #14h
        Jmp Linn
Lin4:   Add a, #54h
Linn:   add a, #80h     ; kode for positionering af Cursor.
        Call LCD_Send_b
        Ret
; ------------------------------------------------------------------------------------
```

Lav nu et program, der afhængig af nogle bit på port P3 skriver forskellige beskeder i displayet. ! Eller efter en pause skriver nye beskeder.

Flere display - manipuleringstrick

```
Mov a, #1 I/D S       ; 3 bit !!
Call LCD_send_b

Mov a, #1DCBb    ; 4 bit !!
Call LCD_send_b

Mov a, #1 S/C R/L * *  ; 5 bit !!
Call LCD_send_b
```

I/D = 1	Cursor incrementes efter hver Write
I/D = 0	Cursor decrementes efter hver Write
S = 1	Display shift ON

D = 0	Display Off
D = 1	Display On
C= 0	Cursor Off
C = 1	Cursor On
B = 0	Cursor blinker ikke
B = 1	Cursor Blinker

Værdierne, enten 0 eller 1 på pladserne D, C og B, findes i skemaet til højre !!

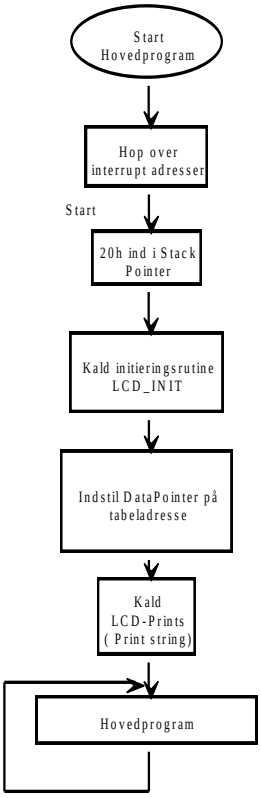
S/C = 1	Shift display, hold cursor fast
S/C = 0	Move Cursor
R/L = 1	Shift Right
R/L = 0	Shift Left

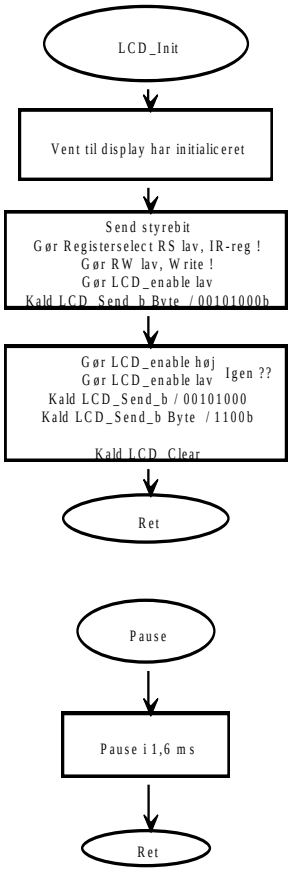
Bemærk, det er hele displayet, der "roteres"
Eks.:

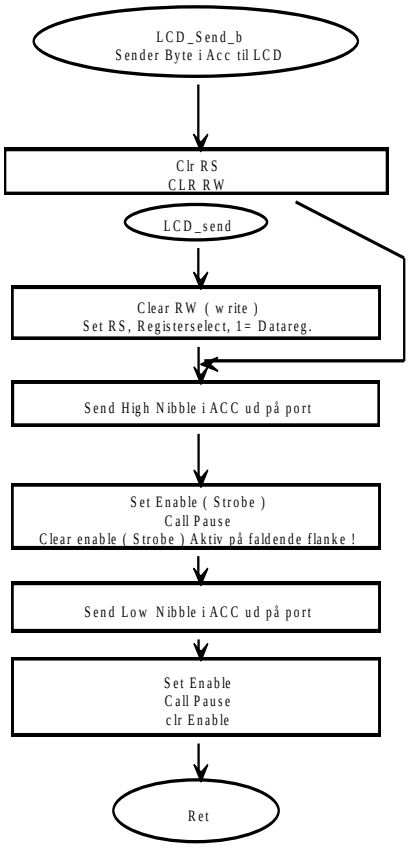
```
Mov a, #111b          ; Increment Cursor, Shift On
Call LCD_send_b       ; Send Befehl / Ordre

Mov a, #11001b        ; Display shift, Hold Cursor fast.
Call LCD_send_b       ; Send Befehl / Ordre
```

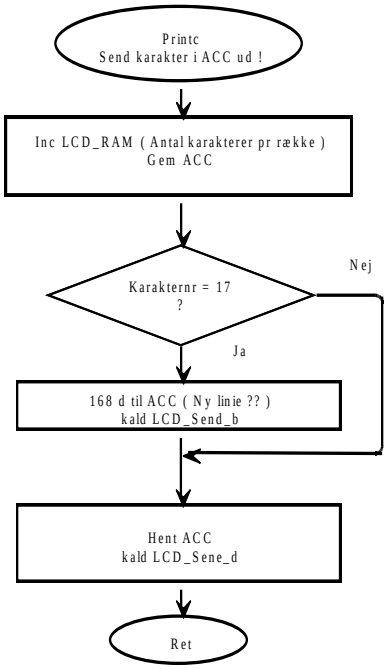
Følgende tegning viser et flowchart over LCD-subrutinerne:

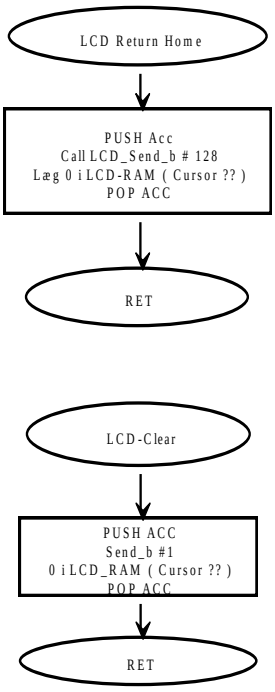
Flowchart over LCD-displayet

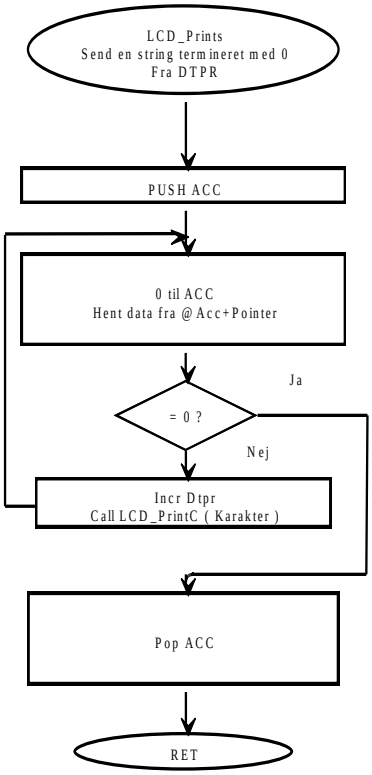












Opgave

Lav først et program, der ”bare” skriver dit navn i displayet. Lav fx 4 linjer tekst, og efter et par sekunder skrives en ny tekst.

b)
Dernæst opbygges et program, der får LCD-displayet til at vise en tekst, og fx i linje 3 en tæller, der tæller antal tastetryk på en knap. Lav 2 eller 3 cifre.

c)
Lav LCD-stopur !!! Brug relevant kode fra det tidligere eksperiment med LED-stopuret til at opbygge LCD-stopuret.

Udvid evt. med Labtime

d)
Den interne timer bruges nu til at opbygge et ur. Tiden skal fx vises i øverste linje.
Der skal tilsluttes et antal knapper til at indstille uret.
I linje 3 vises Mode.

18:24:13		
Indstil timer		
Mode	Op	Ned

Under displayet er placeret 3 knapper. Deres betydning vises i nederste linje i displayet.

Hvis Mode ”indstil timer” er valgt, vil tryk på ”op” forøge timetallet, Ned vil formindske.

Der kan fx laves følgende modes:

Normal
Indstil timer
Indstil minutter
Nulstil sekunder

Start med flowchart, for at få planlagt, hvad der skal programmeres!!

SERVOMOTOR.

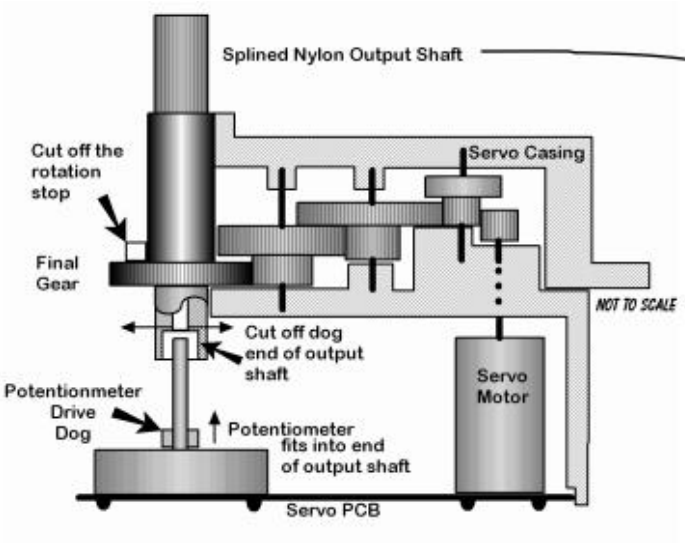
Denne øvelse går ud på at lave et program, der kan få servo-kittet til at udføre bevægelser med de to påmonterede servomotorer.

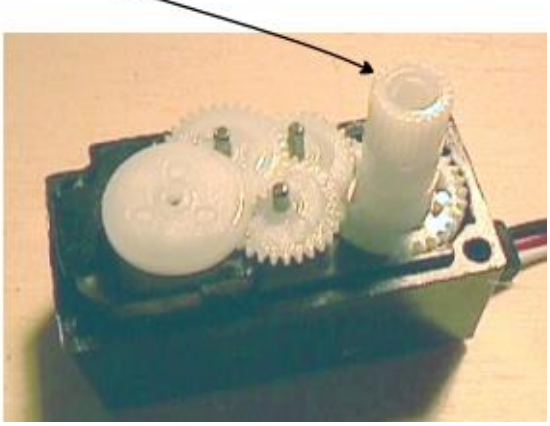
Servomotorer kan normalt dreje ca. 180 grader fra side til side, men der findes også motorer, der kan rotere kontinuerligt.

Benforbindelser:

BLACK Ground
WHITE Control pin
RED +4.8V power supply (+5V works well)

Indmaden i servomotoren ser således ud!



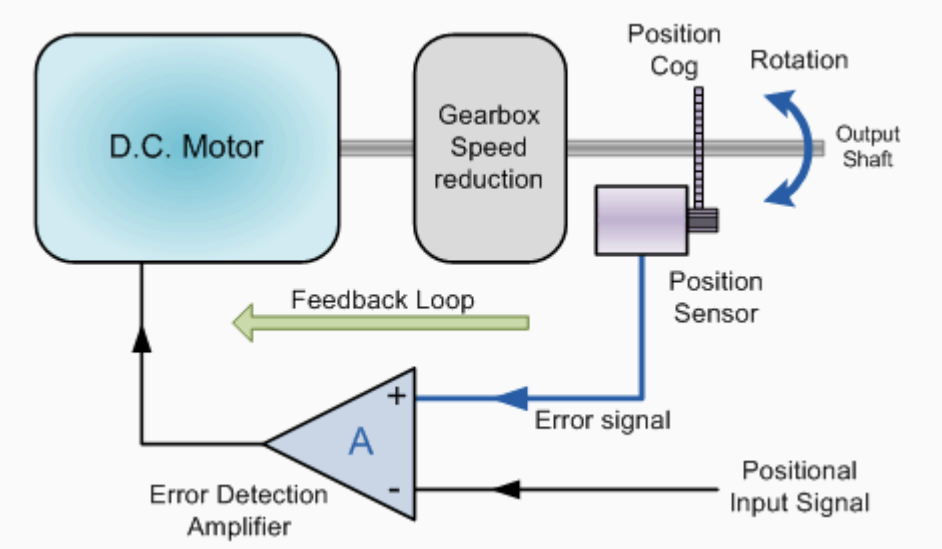


Blokdiagram:

Her er vist et blokdiagram over indmaden i en servomotor.

På akslen ud af motoren er der monteret et potentiometer. Denne giver en spænding fra 0 til 5 Volt, afhængig motorens position.

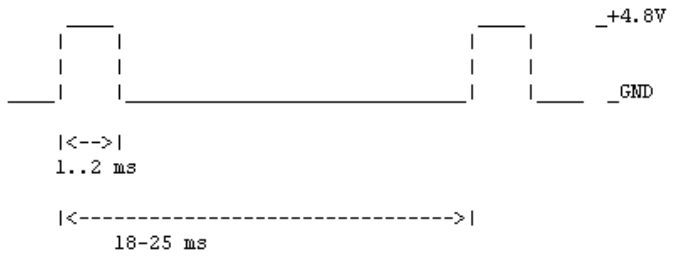
Herved kan elektronikken vide, hvor motoren står.



Kilde: http://www.electronics-tutorials.ws/io/io_7.html

Elektronikken i motoren styres af en Controller, der sender en række pulser til motoren.

Pulserne omformes i motorens elektronik til et positionssignal, og motoren kører til en stilling, så forskellen mellem den indbyggede potentiometers feedback-spænding, og den ønskede position er nul.

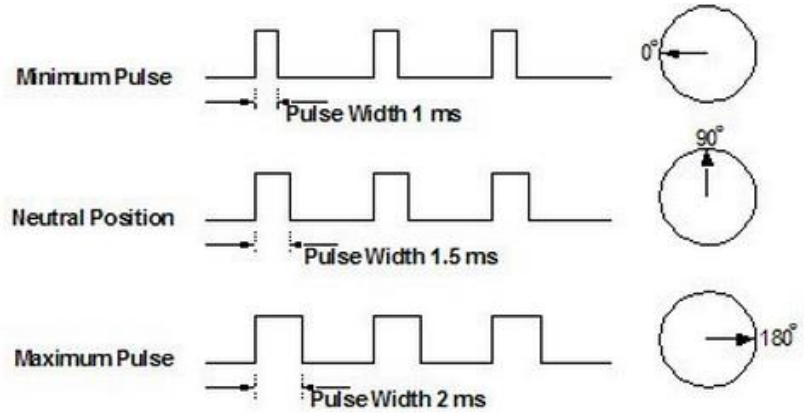


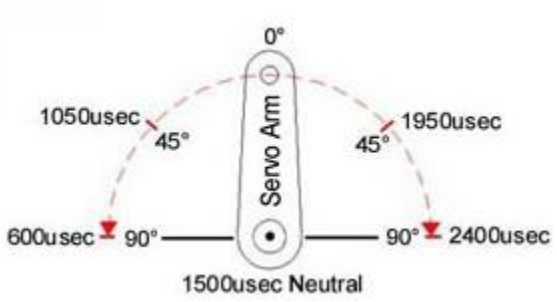
Vha bredden på pulserne på styreledningen kan man bestemme motorens position.

Pulserne skal gentages hver ca. 20 mS.

Motorens position styres af pulsbredden på signalledningen..

Motoren drejer fra den ene yderstilling til den anden hvis pulsernes bredde ændres fra 1mS til 2mS.



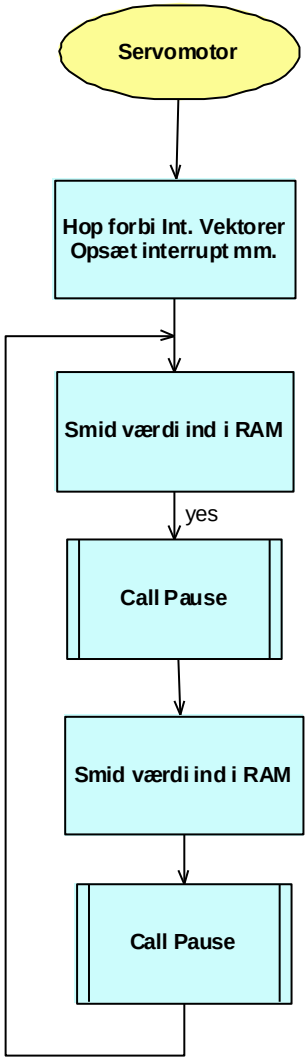


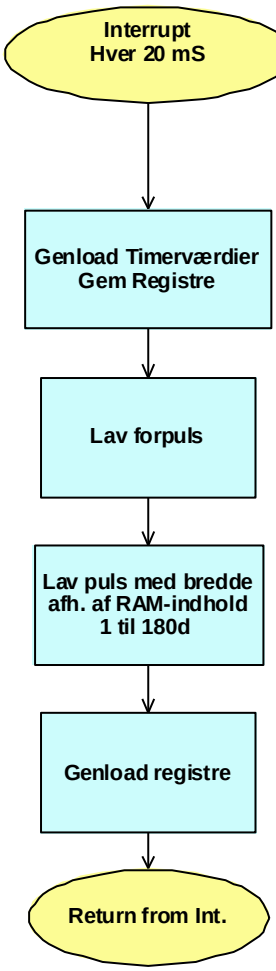
Kilde: http://www.servocity.com/html/hs-422_super_sport_.html

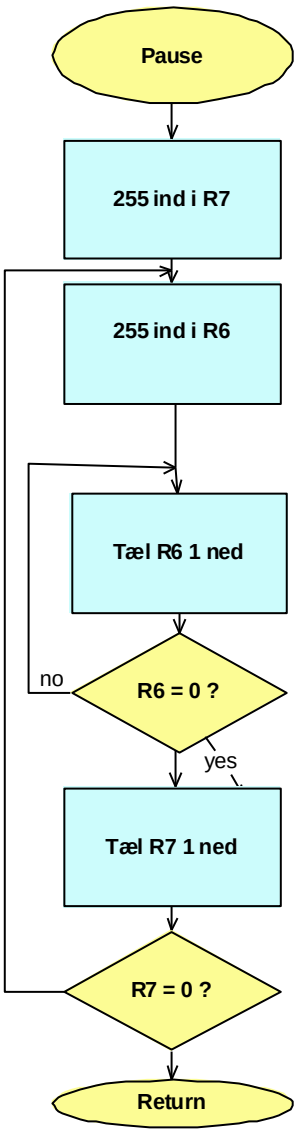
Eksempel på en header til Keil:

```
/*  program-eksempel til at styre en RC-Servomotor

Programmet genererer en puls fra 0,5 ms til 2 ms ca. @ 12 MHz.

Hver gang der er gået ca. 20 ms. skal servomotoren have en puls.

Hvis pulsen er 1 ms bred, er motoren i den ene yderstilling,
er den 1,5 ms, er motoren i midten, og hvis den er 2 ms, er motoren i den anden
yderstilling.

Udmåling af de 20 ms sker i en timer, der udløser et interrupt hver 20. ms.

I interruptrutinen genereres en puls, der er sammensat af først en default-puls, hvis
varighed er ca. 1/2 ms, og dernæst en tid, der er afhængig af indholdet i en RAM-
adresse.

Dvs. at hovedprogrammet kan styre motorens stilling blot ved at lægge en værdi ind i
denne RAM-adresse.

Ved test har der imidlertid vist sig en ret stor tolerance på pulsbredden. For nogel
servomotorer skal pulsen variere fra ca. 0,5 ms til ca. 2 ms.- for at Servoen udfører
en fuld bevægelse fra yderstilling til yderstilling.

*/
```

Flowchart:







Programeksempel:

```
; Konstanter:
        ;

; Pin_def:

    Servo_pin equ P1.0    ; forbindelsesben til servo.

;       Ramdef:

        pulstid equ 0Ah  ; ram for valgt pulstid

; Urdef:  ( Værdier til timeren )

        Timer0LB equ 0DFh
        Timer0HB equ 0B1h          ; virker,  1/20 del sek.
```

```
;*******************************************************************************

Org 0h

    jmp start

;       ORG 03h                 ; Interrupt from extern interrupt 0
;       jmp EX0int

        ORG 0Bh                 ; Interrupt from Counter 0 overflow
        Jmp Timer0int           ; Hop til timerrutine / Jump to subroutine

;       ORG 13h                 ; Interrupt from Extern interrupt 1
;       jmp EX1int

;       ORG 1Bh                 ;Interrupt from timer 1
;       jmp Timer               ;Hop til timerrutine / Jump to subroutine

;       org 023h         ; seriel interrupt vektor, udløst  af RI eller TI
;       jmp serint              ; Created by RI or TI


Org 30h

Start:
        mov sp, #30h

; Opsæt timer 0 til interrupt-generering:

        Mov TMOD, #01h          ;Timer 0, som 16 bit timer
        Mov TL0,#Timer0LB       ;load timer Low Byte
        Mov TH0,#Timer0HB       ;Load timer High Byte
        Setb ET0         ; Enable Timer 0 interrupt
        clr TF0                 ; læg int flag ned
        SETB EA                 ; Global interrupt enable/disable
        setb TR0

        Mov pulstid, #1d ; def startværdi i Pulstid RAM

;------------------------------

; Dette eksempel øger og formindsker trinvis værdien i en ramadresse
; svarende til at servoen bevæger sig langsom fra den ene yderstilling til den anden

Startloop:                      ; forøg værdi

        call Pause
        inc pulstid
        mov a, pulstid          ; tjek for værdi = 179d
        cjne a, #180d, Startloop ;Gentag

Startloop_1:                    ; formindsk værdi

        call Pause
        dec pulstid
        mov a, pulstid
        cjne a, #1d, Startloop_1 ; tjek for værdi = 1d

    jmp Startloop               ; loop

;-----------------------------

; Timer 0 interrupt rutinen:

Timer0int:        ; Udløses 20 gange pr sekund.
```

```
        Mov TH0,#Timer0HB        ; Genload Timer High Byte ; her genstartes,
        Mov TL0,#timer0LB        ; Genload Timer Low Byte  ; pausetiden bliver mindre
                                 ; med længere puls !
        CLR TF0                  ; Reset timer overflow
        PUSH ACC                 ; Gem ACC indhold
        PUSH PSW                 ; Gem programstatusregiste

; nu skal der køres en puls.

                setb servo_pin   ; puls start

; først en for-puls-varighed på ca 0,5 ms.

                Mov R5, #0FFh  ; forpuls  0,5 ms
                DJNZ R5, $

; så en puls, hvis bredde er afh. af værdi i RAM-adresse

                Mov R5, pulstid  ; Ram med pulsbredde, værdi fra 1 til 180d

Timer0int1:
                Mov R4, #3               ; tæl ned
                djnz R4, $
                djnz R5, Timer0int1
;-------
                clr servo_pin    ; puls slut

Timer0Int_9:

        POP PSW
        POP ACC
        Reti
;---------------------------

Pause:
        Mov R7, #255
        Pause1:
                Mov R6, #255
                djnz R6, $
        Djnz R7, Pause1
Ret

;---------------------------

End
```

Timer indstilling.

Microcontroleren har to indbyggede timere.

Timer 1 bruges til baudrate-generering til den serielle port.

Som det ses på næste skitse kan timeren sættes op på flere måder. ”4 modes”, hvor jeg har prøvet mode 1 og 2.

Det er registrene TH0 og TL0, der er de registre, der tælles op af timer 0. De er hver 8 bit, og kan gives en værdi, vha ”Mov TH0, #xxh”, Mov TL0, #xxh. Ligeledes kan værdierne fra registrene Moves til Acc-registeret.

Timeren kan vha af bit i TMOD-registeret indstilles til fx at tælle hver 12. puls fra krystallet, evt. startet og stoppet af en inputpin.

Den kan tælle pulser på en pin.

Timeren som ur.:
Sættes en fornuftig værdi ind i register TH0 og TL0, og pulser fra krystallet tælles,kan det udregnes, hvor lang tid, der går før der kommer overløb. Dvs. tælleren runder 16 1-taller (FF FFh). Dette kan fx ske hver hundrededel sekund. Når der sker overløb, sættes et bit, et flag, TF0. Denne bit kan man holde øje med i en lille løkke. ” JNB TF0, $ ” Eller man kan lade TF0-bittet udløse et interrupt.

Et interrupt er en midlertidig afbrydelse af det igangværende program. Er interrupt fra Timer 0 enablet, vil programmet hoppe til adresse 0Bh i ROM’en. Denne adresse kaldes Timer 0 interrupt-vektoren. Fra adresse 0Bh kan der udføres et kald eller JMP til en interruptrutine. Interruptrutinen afsluttes med ”RETI”.

Ved et 12 MHz krystal er de værdier, der skal loades ind i TH0 og TL0 efter hver overløb = ????.

SERIEL TRANSMISSION AF DATA:

Der benyttes et kit med TxD forbundet til en klemme.

Programmet skal skrives til at sende pakker a´ 8 bit til en ekstern enhed.

Her modtages de læste bit, og kittets driver reagerer som det er programmeret til.

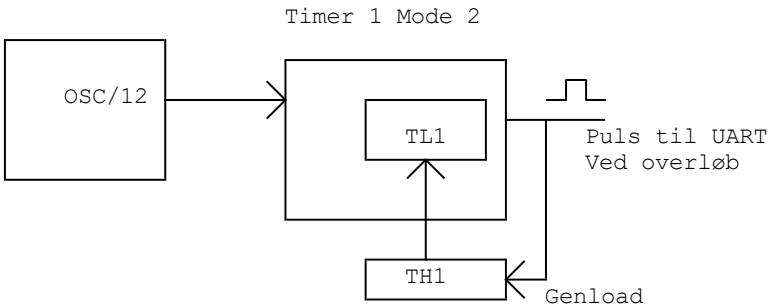
Vejledning:

Timer 1 bruges til baudrate-generering. Timeren indstilles til Mode 2. Her arbejder timeren som 8 bit op-tæller med automatisk genload.

Der flyttes en værdi ind i timerens register: TH1, og timeren indstilles til at tælle clockpulserne op fra denne værdi til 0FFh.

Ved hvert overløb sender timeren så en Clockpuls over til den serielle del. Ved fx 1200 baud er der 1200 pulser pr sek.

På skitseform ser det således ud !!



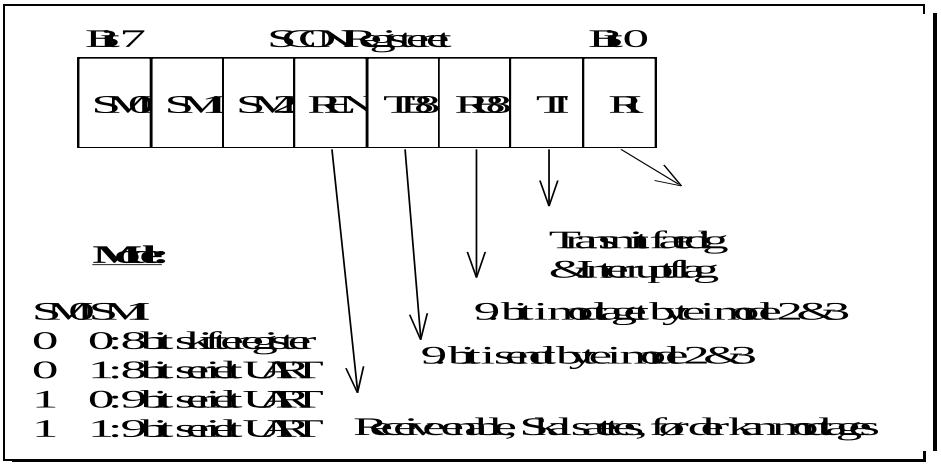
Programeksempel:

```
Mov TMOD, #20h      ; Indstil timer 1 til mode 2, og direkte pulser fra osc.
Mov TH1, #0E8h      ; 1200 Baud, = Bit pr sek.
Setb TR1            ; Sæt bit høj, dvs. sæt timer igang.
```

Herefter indstilles mode for den serielle del:

Opsætningen af den serielle port sker i SCON registeret.

Enten ved at flytte en byte til SCON, eller ved at sætte bittene enkeltvis.




```
Setb SM1          ; Vælger Mode 1, 8 bit data, startbit og stopbit.
Setb REN          ; Recieve Enable enables. Hvis der skal
                  ; modtages data !!
```

Kodeeksempel til at Sende data serielt:

```
Clr TI            ; Clear Transmit færdig Interrupt flag
Mov SBUF, A       ; Læg værdi fra A over i Senderbuffer
JNB TI, $         ; Vent evt her indtil alle 8 bit er sendt.
```

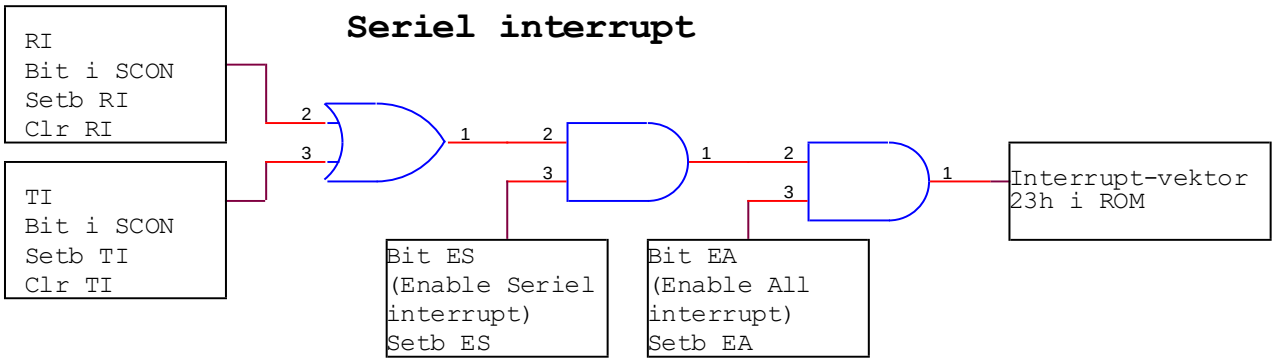
Kodeeksempel til at Modtage data:

```
Clr RI
JNB RI, $         ; Vent til RI-flaget er sat. Sker når en hel byte
                  ; er modtaget.
Clr RI            ; Slet flaget
Mov A, SBUF       ; Flyt værdi over I Acc.
                  ; Ps. Der er i realiteten 2 SBUF- registre
                  ; En til at sende, en til at modtage.
```

Interrupt fra seriel port:

Har et program ikke tid til at vente på et modtaget en byte eller at en byte er sendt, kan man lade den serielle port udløse et interrupt.

Dette kan ske enten på RI-flaget (Recieve Interrupt, Modtaget byte) eller på TI-flaget, (Transmit Interrupt, Byte færdigsendt).



Kodeeksempel:

; Den serielle interrupt-vektor ligger i adresse 23h i ROM-en.

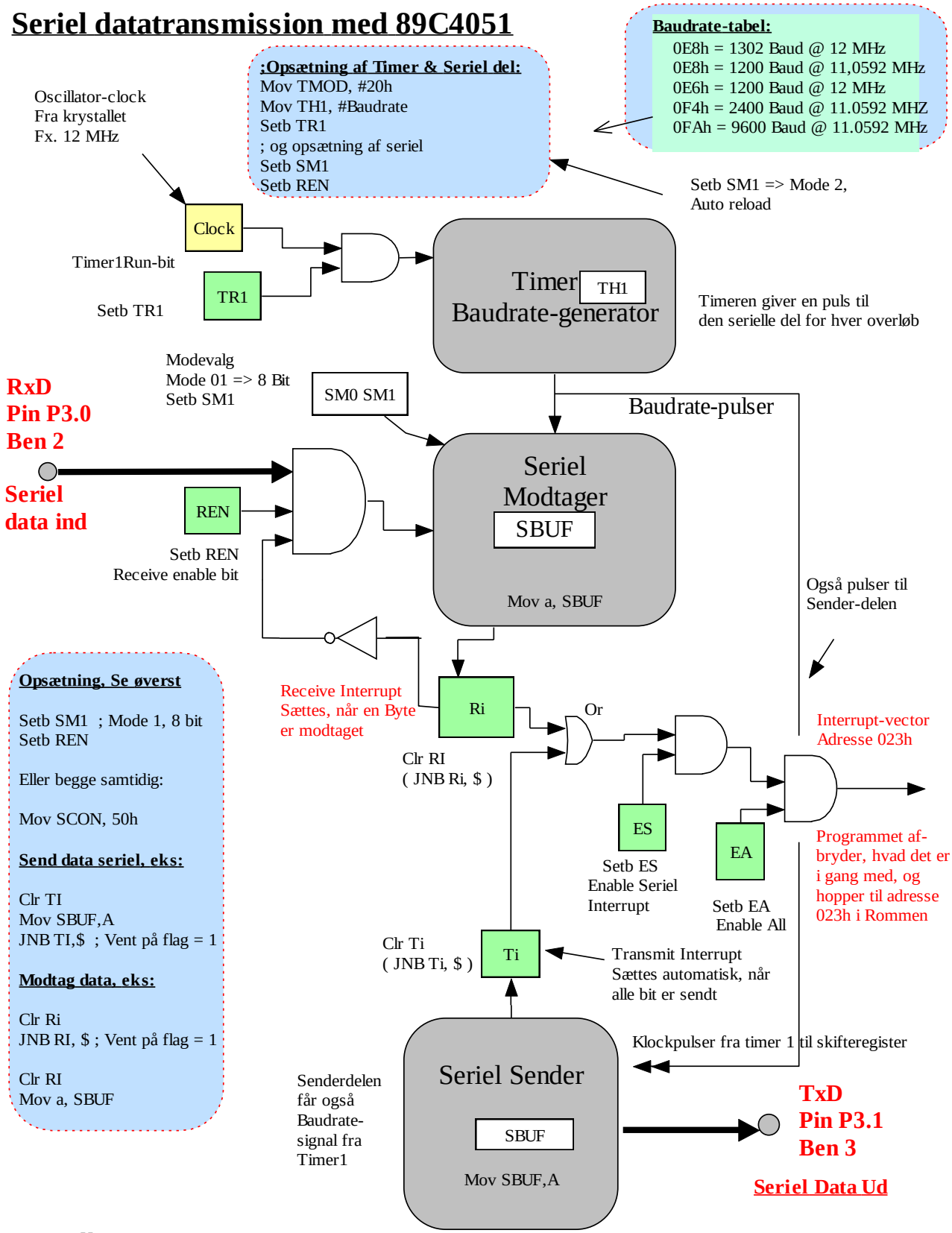
```
        Org 23h         ; Interuptvektor
        Jmp Seriel_int  ; Hop til subrutine. ( Afslut altid med RETI )
```

Bit-flagene Ri og Ti bliver ikke resat ved et Interrupt. Derfor er det muligt for interrupt-rutinen at teste på de to flag, hvilken af dem, der udløste interruptet !!

De skal så cleares i Interrupt-rutinen, for at der igen kan udløses et interrupt.

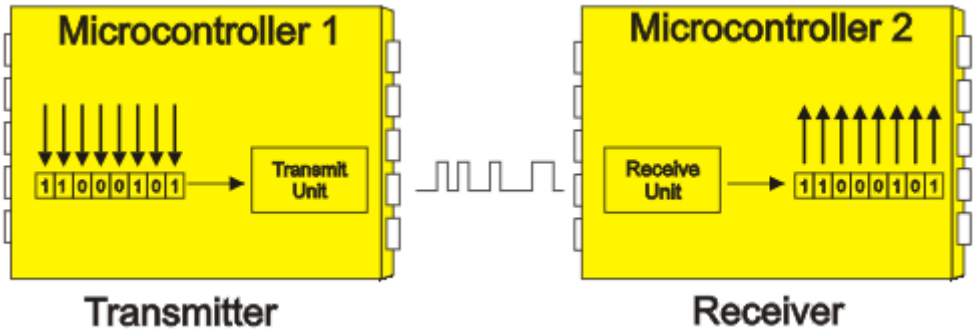
Hele den serielle del ses næste side:

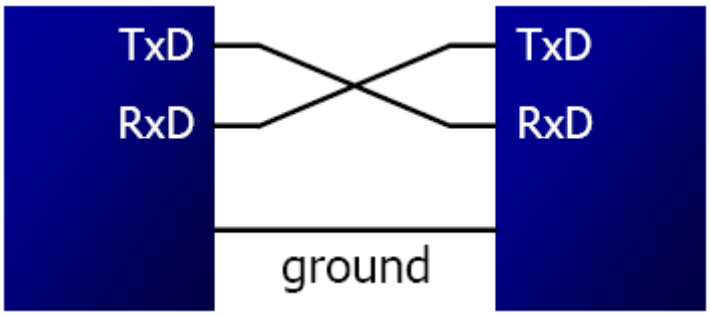
Seriel datatransmission med 89C4051



/ Valle
Redigeret
7/4-2010

Seriel transmission i sin simpleste form sker direkte ud af en UART og ind i en anden UART





TxD forbindes til modtageren RxD.

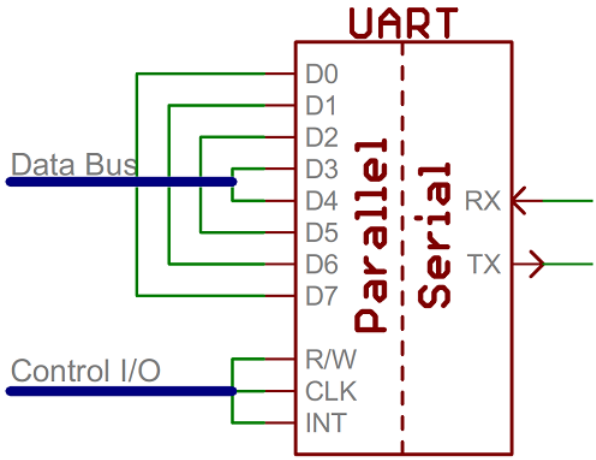
Beskrivelse:

Universal Asynkron Reciever Transmitter

Alt kører på 5 Volt.

Der skal overføres en dataledning og stel, fælles stel !! Vigtigt !!

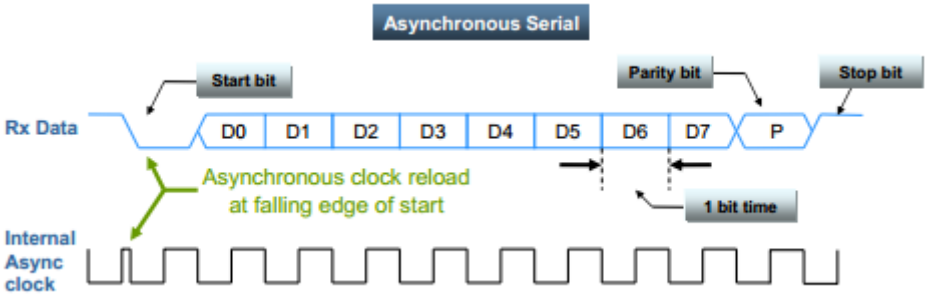
Data læses parallelt ind i et register, - og skiftes derefter ud serielt.



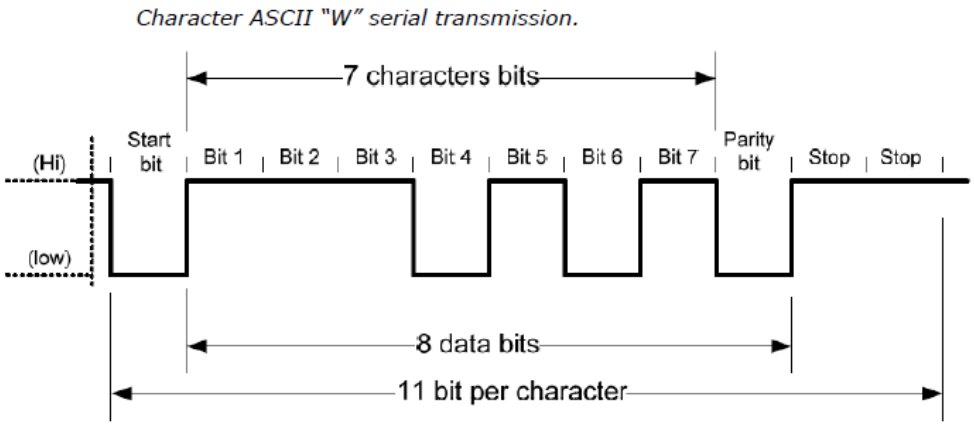
Der er valgt nogle Standard Baudrates: 150, 300, 600, 1200, 2400, 4800, 9600 19200, ….

Signalet starter på +5 Volt.

På et tidspunkt starter et signal på ledningen.



Det skal modtageren bare reagere på.

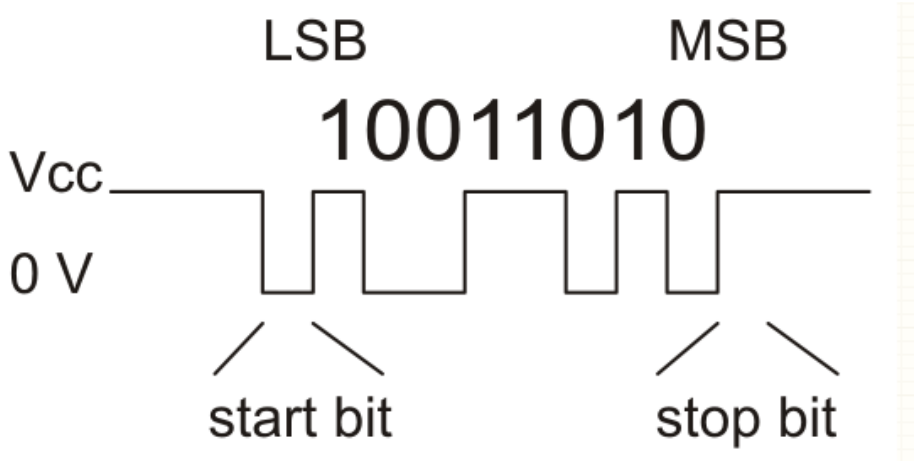


Hi er lig 5 Volt.

Startbit er beregnet til at få modtageren til at "vågne op"

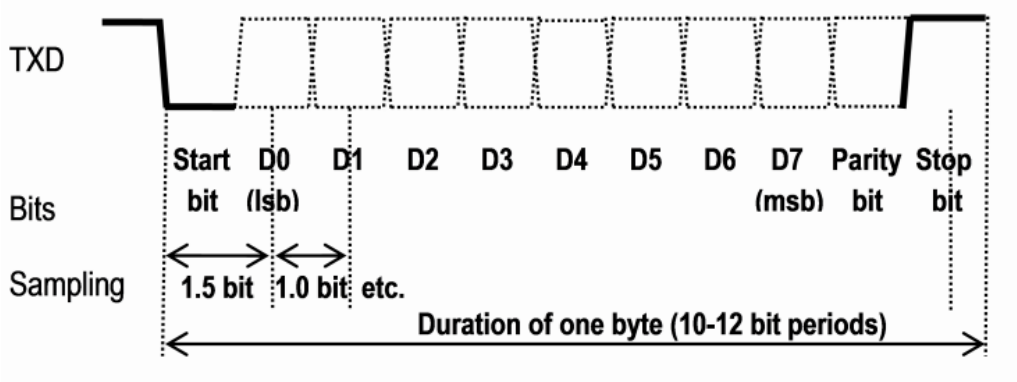
Evt. kan vælges at sende en paritetsbit også for at modtageren kan foretage en form for test af datatransmissionen.

Signal på TX ud af en UART er Normally High



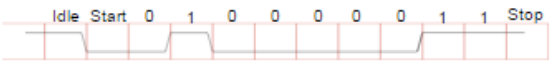
Modtageren skal sample

dvs. Skifte signal ind i et skifteregister

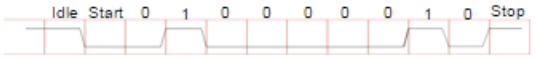


Man skal kende Baud-rate, altså antal bit pr sekund.

De to følgende grafer viser signaler ved brug af paritetsbit.

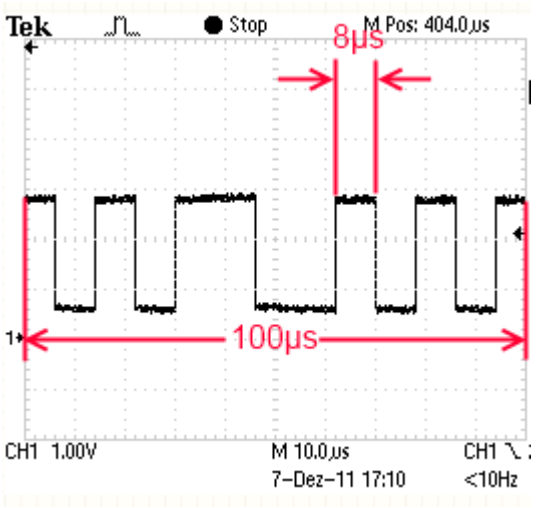


Sending 'A' through UART with odd parity.

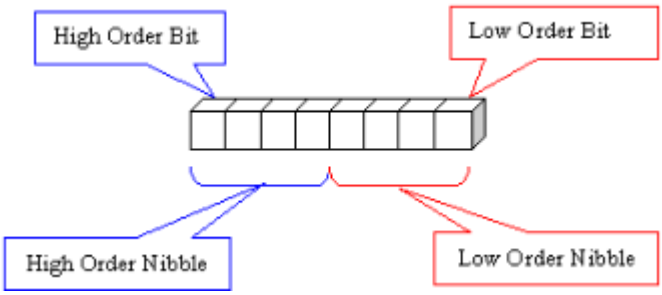


Sending 'A' through UART with even parity.

Husk at forbinde 0 Volt også, dvs. husk fælles stel !!



Et scoopbillede af et serielt signal !!



<u>Testinstrument, - se under instrumenter:</u>

Jeg har et testinstrument, der kan bruges, hvis man skal teste et serielt signal.

7 character alphanumerisk display: (Hej Mor)

Skriv et program, der sender en række karakterer til displayet.

Fx dit navn. Og efter en periode en anden tekst !!

Det Alpha Numeriske display viser karakterer ud fra ASCII koden. Dvs. at hvis der sendes et ”41”h vil displayet vise et stort A.

Protokollen:

Den måde data skal sendes på, skal aftales, så modtageren forstår det sendte. Det beskrives i en såkaldt protokol, svarende til en beskrivelse i en protokol af, hvordan en diplomat skal opføre sig ved et besøg i et land.

Driverne i displayet er skrevet sådan, at der skal sendes 2 byte til hver position I displayet. Den første byte er adressen, og den anden er data, dvs. hvad der skal skrives. Der skal sendes med en Baudrate på 1200 Baud, (-0E6h-).

Displayets segmenter er programmeret til at reagere på adresserne 81h, 82h … 87h – fra venstre.

Hvis bit 8 i en byte er sat, vil displayet opfatte byten som en adresse.

Desværre er ikke alle ASCII karakterer implementeret endnu.

De specielle danske karakterer har jo ikke en værdi i ASCII-tabellen. De har fået tildelt ASCII-værdierne:

æ = 01h
ø = 02h
å = 03h

Æ = 04h
Ø = 05h
Å = 06h

Selv-definerede Karakterer:

Det er også muligt, at skrive selvdefinerede karakterer i displayet.

Hvis der sendes en adressebyte, med Bit 4 sat høj, (fx. 10010xxxb), afventer displayet at der kommer yderligere 5 bytes.

Den første byte er mønstret i venstre søjle, 2. byte er næste søjle osv.

Protokollen for de 5 data-bytes er: 0xxxxxxxb, hvor bit 6 er den øverste LED, og bit 0 er den nederste LED i søjlen. Bit 7 skal være 0, ellers tror de andre karakterer, at det er en adresse.

Protokol tilføjelse:

Det burde også at være implementeret i displayets drivere, at hvis der sendes en pakke med A i Highnibble, fx 1010 xxxx, vil de næste 7 byte direkte blive skrevet i de 7 segmenter.

Dette er dog ikke testet endnu !!

På næste side vises ASCII tabellen. ASCII er forkortelsen for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.

Det er en kode skabt til at standardisere data der blev sendt til fx printere tilbage i tiden. Det var jo ikke så smart, at alle printere skulle have de samme data for at skrive en tekst.

ASCII-tabellen:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Udvidelse af programmet:

Lav program, der henter tekst i en tabel ved tabelopslag.

Udvid med tjek af danske karakterer fra tabellen, og konverter dem så æ = 01h, ø = 02h osv. !!

Lav flowchart !!

Løbelys-kit.

I løbelyskittet sidder der på bagsiden en uC til hver Dot matrix display.

Den højreste uC fodres med serielle data, i form af en byte, der beskriver hvilke lysdioder, i den aller højreste søjle.

Når næste byte sendes til displayet, sørger uC'ene for, at den forrige byte rykker 1 tand mød venstre.

Selve matrixet i de enkelte enheder er multiplexet af en controller.

Der sidder yderligere et print på bagsiden – med en Controller, der fodrer displayene serielt.

I programmet er der et par tekststrings, indprogrammeret i tabeller.

Disse tekster sendes serielt til styre-uC-erne.

Teksterne i tabellerne indlæses til RAM adresse 30h og opad. De afsluttes med en 00h ~ end of file.

Inden data sendes vider, bliver de omkodet til et mønster for pågældende bogstav, og yderligere omkodet, så de forskellige bit bliver flyttet rundt, til den rigtige placering i hardwaren. Det er nemlig ikke sådan på printet, at bit P1.7 styrer øverste række, P1.6 næstøverste osv.

Programmet i styre-uC-en ønskes nu ændret så der kan modtages serielle data fra en ekstern enhed.

Den eksterne enhed kan så opfattes som en programmeringsenhed, så der fx kan flyttes aktuelle beskeder over i displayet, og så disse bliver vist i stedet for de fast indprogramerede.

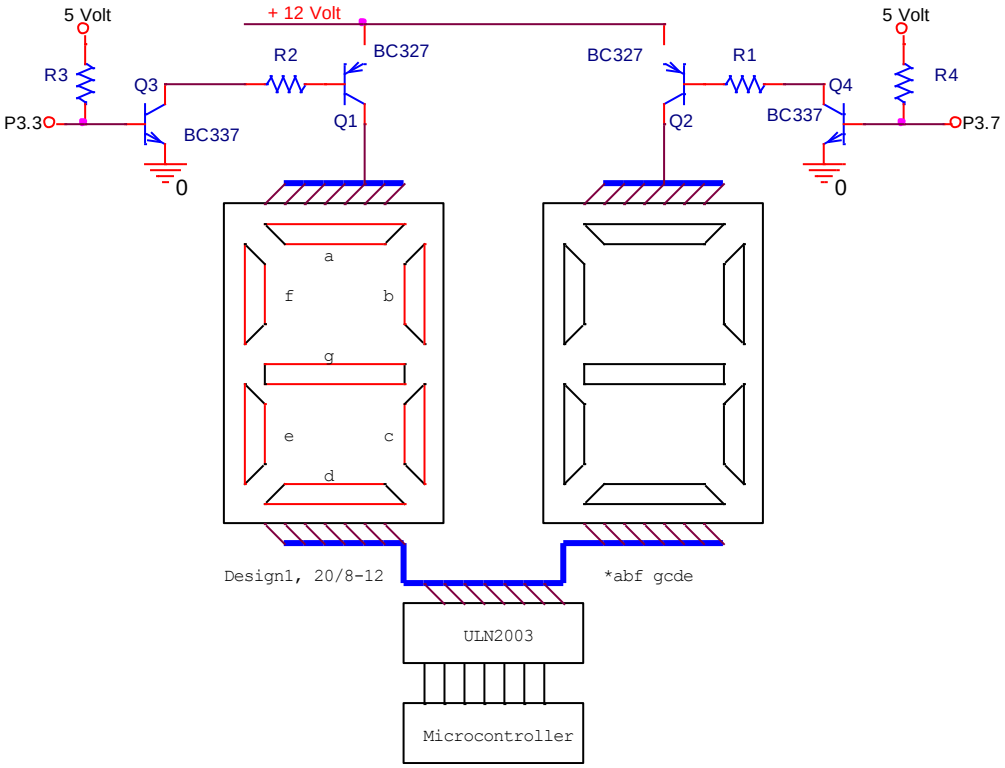
Pernille:

Pernille-kittet er et tal-display.

P3.0, P3.1, P3.2, P3.4 og P3.5 er ført ud til extern klemme.

Data skal sendes serielt!!

Diagrammet viser hvordan systemet er forbundet!



Protokol: Baudrate = 1200, dvs. Mov TH1, #0E6h

Data skal sendes serielt som pakket BCD. 10'ere i high nibble, og 1'ere i low nibble.

Thermo printer

Thermo printeren skal også have data sendt serielt.

Lav et program, der - fx når der trykkes på en knap - sender en tekst til termoprinteren.

Protokol: Printerens ID = 8Ah, dernæst sendes tekst som ASCII. Printeren gemmer den modtagne tekst, og skriver først en linje når den modtager en Carriage Return. Se ASCII tabellen.
Her kan vi opfatte pakken som "End of String". Værdien der skal sendes er 0Dh.

Baudrate = 1200.

A/D-KONVERTER TLC549

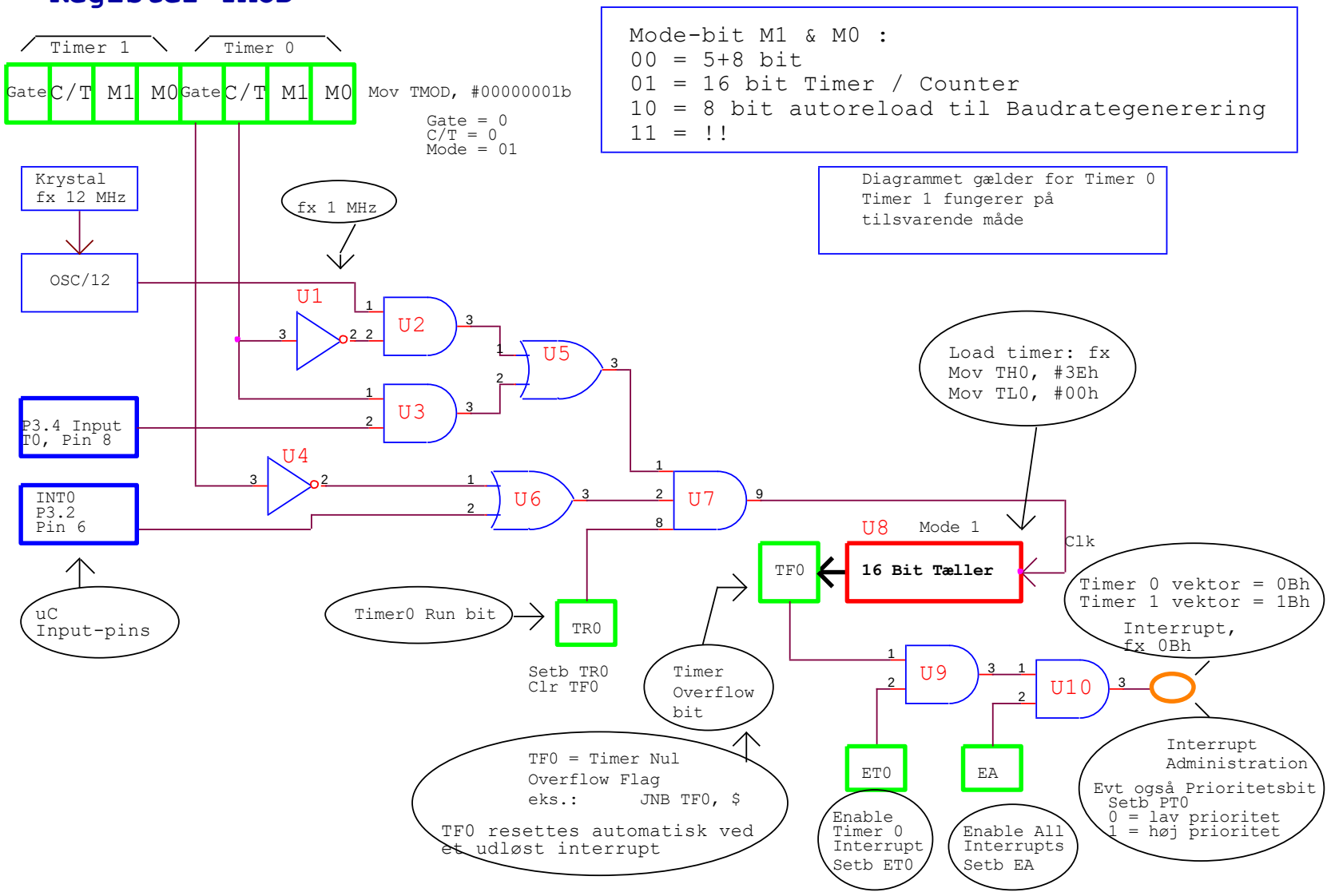
Lav et program, der vha. den serielle AD-converter TLC 549 konverterer spændingen fra et potmeter.

TLC har en opløsning på 8 bit, og programmet skal sende disse, ud på porten P1 til 8 lysdioder ! Husk at invertere dataene. (Alternativt kan et LCD-display bruges, men så skal der konverteres)

Følgende program kan bruges:

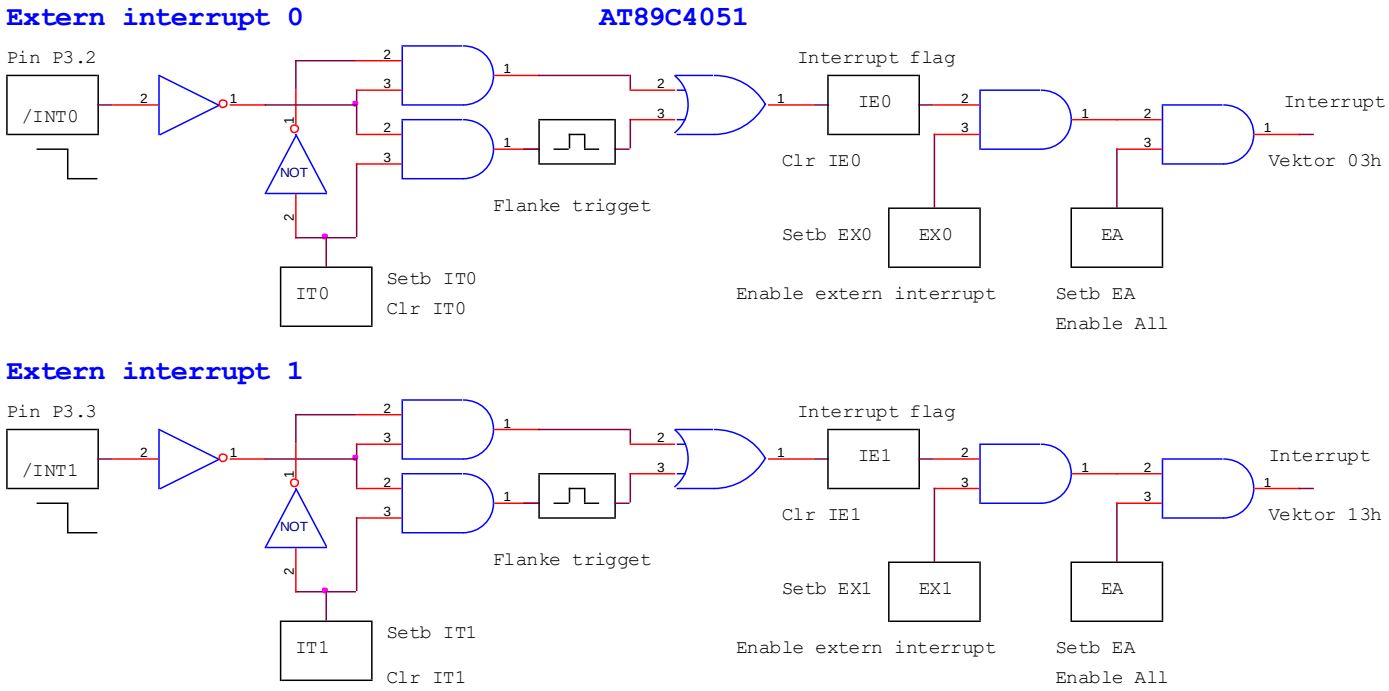
```
; Konverteringsrutine for TLC549 8 bit A/D-konverter.


; Pindefinitioner

        ADC_CLK     EQU P1.x ; Vælg Clockben
        ADC_DATA    EQU P1.y ; Vælg hvilken ben, der forbindes til ”Data”
        ADC_CS  EQU P1.z ; Vælg Chip Celect Pin


; Subrutine-programmet, der konverterer og returnerer én konverteret værdi I reg A

ADC1:   setb ADC_DATA
        Clr ADC_Clk
        Setb ADC_CS

ADC2:
        Clr A

; Start Access
        Clr ADC_CS
        Mov B, #8               ; Tæller, samtidig 2 us delay efter CS low

ADC3:
        Mov C, ADC_DATA         ; Data til Carry
        Setb ADC_Clk            ; Clock høj
        RLC A                   ; Rotate Carry til Acc
        Clr ADC_CLK             ; Clock lav for næste bit
        DJNZ B, ADC3            ; Gentag 8 gange

; Start næste konvertering

        Setb ADC_CS             ; Tager mindst 17 uSek !
        Ret
```

Register TMOD



Anvendes timeren til 1/100 sek. interrupt, skal der genloades med D8F7h @ 12 MHz

Oversigt over indstilling af ekstern interrupt !



Er der valgt flanketrigning, resettes IE0 / IE1 automatisk af interruptfunktionen

d. 23/4-05 / Valle

Hvis "Edge triggering" er enabled, bliver IE0 / IE1 bit resat ved hop til interrupt routine !