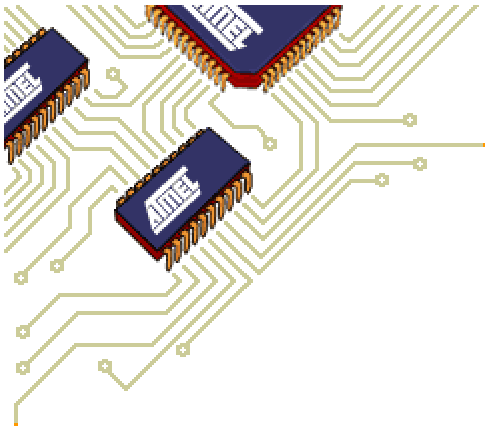
Microprocessoren

Projekt



Udarbejdet af Kasper jensen EUC-Syd – 4ES 10-3-2001

Indholdsfortegnelse :

Mikroprocessor general :

Mikroprocessors funktion :

I det daglige ligger man ikke mærke til mikroprocessore, men de sidder over alt næsten, hvis man ved de er der, Biler, mikroovne, clockradioer o.s.v.
Det der er smart ved en processor er den er utrolig brugbar, ved man selv bestemmer hvad den skal kunne, de eneste begrænsninger er hvor stor en mikroprocessor man anvender, de fås fra lille 8 pins til over 68 pins, med nogle med eksterne ram og rom, og andre med interne, men det vil ik' blive beskrevet her, denne rapport vil herfra kun handle om processoren fra Atmel, AT89C4051.

AT89C4051 :

At89C4051 er en processor der anvender 8051 arkitektur, den har en 8 bits port og en 7 bit , ved navn port1 og port 3, hvor der på port 3 findes 2 interupt, 2 16 bit timere og RX og TX til seriel kommunikation. Kredsen findes i en 20 pins hus, med 4 Kb intern flashram, 128 bytes ram.

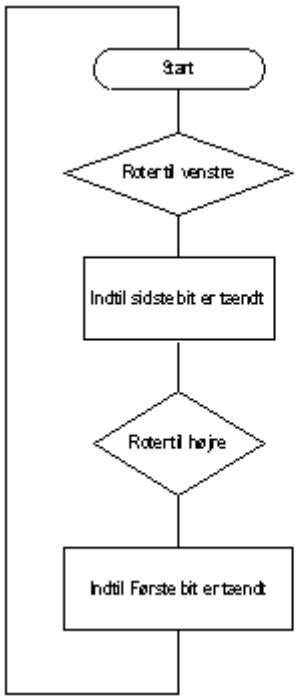
Programmering :

Programmeringen kan foregå i forskellige programmeringssprog, det der kommer tæt på cpu'en egene instruktioner er assambler, men man kan også programmere i højniveaus sprog, bare der findes en 8051 arkitektur compiller.

Assambler programering :

Nightraider :

Programmering af et program som kan udskrive værdier på en port på cpu'en, som vil starte på P1.0 derefter aktivere en port i rækkefølge til P1.7 og derefter til P1.0.

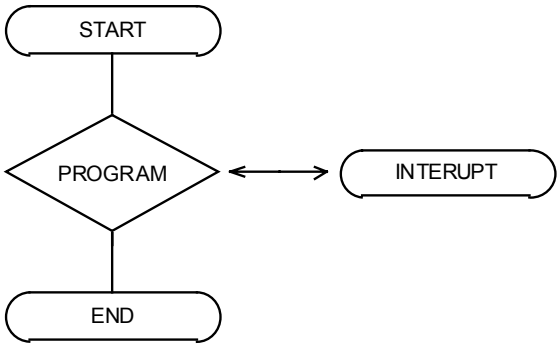


Programmet er lavet til lysdioder på P1, som er aktiv på logisk "0"

```
ORG 0000H                   Programets start adresse
        mov a,#0FEH         Flytter den inverteret 1 værdi i ACC
rotl:   mov p1,a            Flytter ACC ud på P1
        call delay          Kalder DELAY
        RL a                Rotere ACC mod venstre
        CJNE a,#7FH,rotl    Hopper til rotl hvis Acc <> fra 7FH
rotr:   mov p1,a            Flytter ACC ud på P1
        call delay          Kalder DELAY
        RR a                Rotere ACC mod højre
        cjne a,#0FEH, rotr  Hopper til rotr hvis Acc <> fra FEH
        jmp rotl            Hopper til rotl

DELAY:  MOV R1,#2DH         En venteløkke der tæller ned I 3 registre
LOOP3:  MOV R2,#2DH
LOOP2:  MOV R3,#2DH
LOOP1:  DJNZ R3,LOOP1       DJNZ = Decrease Jump Not Zero
        DJNZ R2,LOOP2
        DJNZ R1,LOOP3
        Ret                 Returnere fra en Call funktion
END                         Slutter program
```

Styring af interupts :

Interupt betyder på engelsk afbrydelse, når du aktivere en interupt på cpu'en gør den det som står i interuptrutinen, og vender tilbage til sit program hvor den kom fra og fortsætter. Der er 8 forskellige interupts i 89c4051, hvor de 2 af dem er hardware interupts.




```
ORG 0000H
JMP START                   ; Hopper til Hovedprogram

ORG 0003H
JMP Interupt                ; EXT. INT0 Aktiveret

ORG 0100H
START:  SETB EA             ; Aktivere alle interupts
        SETB EX0            ; Aktivere interupt 0, aktiv lav

{Her skal programmet skrives}

Interupt: CLR EA            ; Sikrer interuptet ik' bliver aktiv nu
        CJNE R0,#0H,Hoj     ; De næste 4 linier bestemmer
        MOV r0,#1H          ; om programmet kører eller er stoppet
        JMP Retur           ; og aktivere det modsatte
Hoj:    MOV r0,#0H
Retur:  SETB EA             ; Aktivere alle interupts
        RETI                ; Return fra interupt rutine

Vent:   JMP Vent            ; Stop løkke
END
```

Interupten hoppe til en defineret adresse, i ex. Interupt 0 = 0003H, så selve hovedprogrammet skal starte på en senere adresse for ikke at genere interupt rutinen, selve interuptrutinen må ikke blive for lang, da der kun er afsat et hvis data område dertil.

LCD-Display :

Lcd-displayet består af 32 felter i 2 linier, som kan vise 5 * 8 pixels, som kan representere alle ASCII tegnene, men man kan også selv definere andre tegn/symboler. På displayet er der en ASCII decoder, som skal initaliseres for at diplayet virker, det kan køres i 2 modes, en 8 bit bus mode, eller en 4 bit hvor den øverste nible bliver sendt først, og derefter den nederste del.
R/W bestemmer om der skal læses eller skrives fra displayet, vi bruger kun skrive delen, så benet kan ligges på logisk ”0”.
RS, [register select] bestemmer om der skal skrives/læses data fra displayet, eller om det er en funktion på lcd-displayet, det kan ex. Bruges til at bestemme cursor typen, clear display eller flytte displayet til højre eller venstre.

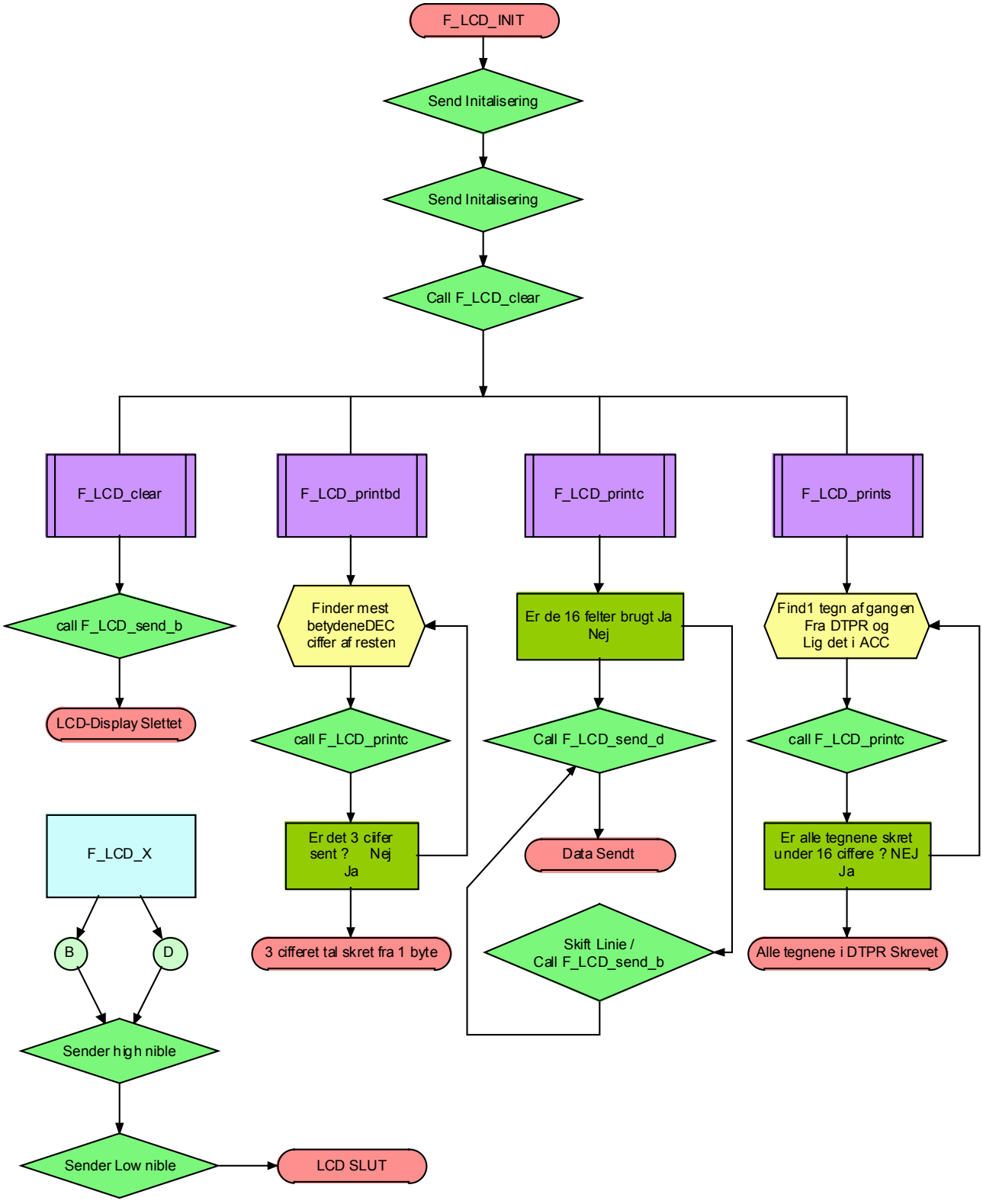
Instruction	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Clocks
NOP	0	0	0	0	0	0	0	0	0	0	No Operation	0
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears display & sets address counter to zero.	165
Cursor Home	0	0	0	0	0	0	0	0	1	0	Sets address counter to zero, returns shifted display to original position. DDRAM contents remains unchanged.	3
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction, and specifies automatic shift.	3
Display Control	0	0	0	0	0	0	1	D	C	B	Turns display (D), cursor on/off (C) or cursor blinking(B).	3
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	0	0	Moves cursor and shift display. DDRAM contents remains unchanged.	3
Function Set	0	0	0	0	1	DL	N	M	G	0	Sets interface data width(DL), number of display lines (N,M) and voltage generator control (G).	3
Set CGRAM Addr	0	0	0	1	Character Generator RAM						Sets CGRAM Address	3
Set DDRAM Addr	0	0	1	Display Data RAM Address							Sets DDRAM Address	3
Busy Flag & Addr	0	1	BF	Address Counter							Reads Busy Flag & Address Counter	0
Read Data	1	0	Read Data								Reads data from CGRAM or DDRAM	3
Write Data	1	1	Write Data								Writes data from CGRAM or DDRAM	3

Først skal displayet initialiseres ved at udsende funktions byte.
Derefter kan man skrive til displayet, og bruge de specielle funktioner, som clear display o.s.v, men RS skal være ”0” for at bruge de specielle funktioner.
Enabelbenet også kaldet strobe skal gå højt i det øjeblik niblerne er stabile på dataporten.
Kontrastbenet skal ligge lavt på nogle typer display, for man kan se noget på displayet, men andre typer kan det forekomme det skal ligge på andre niveauer.

Lcd_4b.inc

For at få forståelse af lcd displayets kommandoer vil jeg prøve så godt som jeg kan, at forklare include filen til lcd displayet.

Flowchart over Inc filen



```
F_LCD_init          : Sætter displayet op i den rigtige mode
F_LCD_clear         : Sletter displayet
F_LCD_printbd       : Skriver en byte som tal (ACC, decimal)
F_LCD_printc        : Skriver en byte som karakter (ACC)
F_LCD_prints        : Skriver en streng (DPTR, Nul-afsluttet)
F_LCD_home          : Sætter cursoren i position 1.1

F_LCD_init:         ; LCD initialiseres
    push ACC        ; Gemmer det nuværende i ACC til brug efter rutine
                    ;  vent indtil spændingsforsyningen har stabiliseret sig
    mov a,#10       ; Ligger ventetiden i ACC
    acall F_wait_m  ; Kalder vente rutinen
    clr RS          ; Sætter LCD til speciel funktion
    clr RW          ; Sætter LCD til at kunne skrive til det
    clr ENABLE      ; Sørger for enable er lav før data sendes
    mov a,#00101000b ; Skriver funktion set til ACC
    acall F_LCD_send_b ; Kalder sende rutinen for instruktioner
    setb ENABLE     ; Kan ik' forklare disse 2 linier →
    clr ENABLE      ; De burde kunne slettes, tror jeg.
    mov a,#00101000b ; sender det same igen, for at være →
    acall F_LCD_send_b ; Kalder sende rutinen for instruktioner
    mov a,#1100b    ; Tænder LCD, Cursor off, ingen blink på markering
    acall F_LCD_send_b ; Sender ovenfor initalisering
    acall F_LCD_clear ; Kører rutinen Clr LCD
    pop ACC         ; Ligger det der lå i ACC før LCD_init i ACC igen
    ret             ; Return fra subrutine

F_LCD_home:
    push ACC        ; Gemmer det nuværende i ACC til brug efter rutine
    mov a,#2H       ; skriver funktionen Cursor home til lcd
    acall F_LCD_send_b ; Kalder sende rutinen for instruktioner
    mov LCD_RAM,#0  ; Reset LCD_Ram, Fortæller hvilken pos. Cursoren er i
    pop ACC         ; Ligger det der lå i ACC før LCD_init i ACC igen
    ret             ; Return fra subrutine

F_LCD_clear:
    push ACC        ; Gemmer det nuværende i ACC til brug efter rutine
    mov a,#1H       ; skriver funktionen slet display, og reset Adr.counter
    acall F_LCD_send_b ; Kalder sende rutinen for instruktioner
    mov LCD_RAM,#0  ; Reset LCD_Ram, Fortæller hvilken pos. Cursoren er i
    pop ACC         ; Ligger det der lå i ACC før LCD_init i ACC igen
    ret             ; Return fra subrutine
```

```
F_LCD_printc:              ; skriv en karakter fra acc (benytter LCD_RAM)
                           ; er mellemrummet i RAM nået?
    push ACC               ; Gemmer det nuværende i ACC til brug efter rutine
    inc LCD_RAM            ; Tæller op på LCD_Ram, for at se om 16 tegn er nået
    mov a,LCD_RAM          ; Flytter det I ACC
    clr c                  ; Sletter en måske sat carry
    subb a,#16D            ; trækker 16 fra Acc, for at se om det 16 tegn er skrevet
    jnz F_LCD_printc_weiter ; hvis den ik' er nul hopper den videre til Printc_weiter
    mov a,#128D+40D        ; Funktion til at skifte linie, burde være 40H note1
    acall F_LCD_send_b     ; Kalder sende rutinen for instruktioner
F_LCD_printc_weiter:       ; Skriver tegnet hvis linien ik' er fyldt
    pop ACC                ; Kalder Tegnet som skal skrives tilbage i ACC
    acall F_LCD_send_d     ; Kalder sende rutinen for Data
    ret                    ; Return fra subrutine
```

Note1 : 128D bruges til at selectere DDRAM med, 40D går lige ud over ram-området for 1 linie på lcd displayet, som går fra 00H til 27H, mens 40D er 28H, og ramområdet for linie 2 er 40H til 67H, jeg synes det giver lidt at tænke over, men det kan være fejl i databladet, og jeg kan kun se det kan være decimal i programmet.

```
F_LCD_printbd:
                           ; skriv en byte som tal
    push ACC               ; Gemmer det nuværende i ACC til brug efter rutine
    push 0                 ; Gemmer det nuværende i 0 til brug efter rutine ??
    push B                 ; Gemmer det nuværende i B til brug efter rutine
    mov B,#100             ; Dividere Tallet med 100 for at få mest betydene →
    div AB                 ; ciffer ud af en byte
    add a,#30h             ; og addere det med 30 for at få ASCII tallene
    acall F_LCD_printc     ; Kalder Printc rutinen for at skrive karekteren I lcd
    mov a,B                ; Flytter "resten" af forige dividering til ;ACC
    mov B,#10              ; Dividere "resten" med 10 for at få mest betydene ciffer →
    div AB                 ; af ACC og ligger "resten" I B
    add a,#30h             ; addere det med 30 for at få ASCII tallet
    acall F_LCD_printc     ; Kalder Printc rutinen for at skrive karekteren I lcd
    mov a,B                ; Flytter "resten" af forige dividering til ;ACC
    add a,#30h             ; addere det med 30 for at få ASCII tallene
    acall F_LCD_printc     ; Kalder Printc rutinen for at skrive karekteren I lcd
    pop B                  ; Kalder Tegnet som skal skrives tilbage i B
    pop 0                  ; Kalder Tegnet som skal skrives tilbage i 0 ??
    pop ACC                ; Kalder Tegnet som skal skrives tilbage i ACC
    ret                    ; Return fra subrutine
```

```
F_LCD_prints:
; skriv en tekststreng, som afsluttes med 0
      push ACC              ; Gemmer det nuværende i ACC til brug efter rutine
      push 0                ; Gemmer det nuværende i 0 til brug efter rutine
      mov R0,#0             ; Sætter R0 til 0 for at reset R0
F_LCD_prints_anf:
      mov a,R0              ; Ligger R0 i ACC
      movc a,@A+DPTR        ; Ligger det tegn som tallet fra ACC
                            ; angiver af tegnnummer fra DPTR i ACC, note 2
      jz F_LCD_prints_weiter ; Hvis ”0” Slut af DPTR så hop til LCD_prints_weiter
      inc R0                ; Tæller op på R0
      acall F_LCD_printc    ; Kalder skriv en karakter rutine
      mov a,R0              ; Flytter antallet af tegn til ACC
      subb a,#16            ; Trækker 16 fra, for at det kan være på en linie
      jnz F_LCD_prints_anf ; Hopper ikke videre hvis der ik’ er flere tegn at skrive på

F_LCD_prints_weiter:      ; Stop label
      pop 0                 ; Kalder Tegnet som skal skrives tilbage i 0
      pop ACC               ; Kalder Tegnet som skal skrives tilbage i ACC
      ret                   ; Return fra subrutine
```

Note 2, Det tal som ACC repræsentere, bestemmer hvilket tegnnummer fra DPTR der bliver lagt over i ACC. EX :

```
DTPR : ”FARNELL”
A = 3D
Movc a,@A+DPTR
A = ”R”
```

En tilføjelse jeg har lavet til min INC fil, så der kan vælges øverste eller nederste linie.

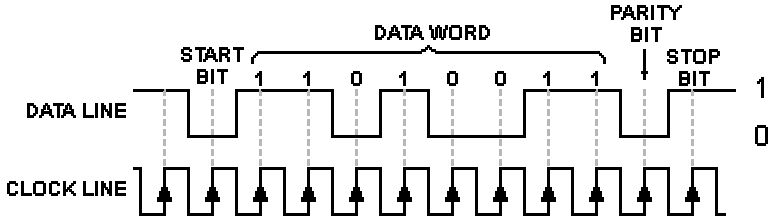
```
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
F_LCD_LOWLINE:
      push ACC
      mov a,#128D+64D
      Acall F_LCD_send_b
      POP acc
      RET
;-------------------------------------------------------
F_LCD_UPLINE:
      push ACC
      mov a,#128D+0D
      Acall F_LCD_send_b
      POP acc
      RET
;-------------------------------------------------------
```

```
F_LCD_send_b:       ; sender instruktioner fra ACC som data
      clr RS        ; Sætter RS benet lav, for function set
      clr RW        ; Sætter RW benet lav
      ajmp F_LCD_send ; Hopper til F_LCD_send
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
F_LCD_send_d:       ; sender data fra ACC til P1, først high-nibble,
                    ; derefter low-nibble
      clr RW        ; Sætter RW benet lav, for data
      setb RS       ; Sætter RS benet lav
F_LCD_send:
      push ACC      ; Pusher ACC 2 gange for at den skal bruges midt →
                    ; I rutinen og for at slutte rutinen med værdien
      push B        ; Pusher B for at være sicker på ik' at slette noget
      push ACC      ; Anden gang Push ACC
      anl a,#0F0h   ; Sørger for de 4 nederste bit er 0
      mov b,a       ; Ligger ACC over i B
      mov a,P1      ; Læser port 1 ind I ACC
      anl a,#0Fh    ; Sørger for de nederste 4 bit bliver I ACC, og 0 på resten
      orl a,b       ; Note 3
      mov P1,a      ; Flytter nu det hele ud på port 1
      setb ENABLE   ; Sender strobe pulsen ud
      mov a,#2      ; bestemer tiden på wait
      acall F_wait_m ; kalder en vent function på 2 ms
      clr ENABLE    ; Ligger stroben lav
      pop ACC       ; Henter indholdet af ACC med org. data i.
      swap a        ; Bytter om på øverste og nederste nible
      anl a,#0F0h   ; Det samme som ovenfor, bare med den nederste
      mov b,a       ; Nible af dataene som skal udskrives
      mov a,P1      ;
      anl a,#0Fh    ;
      orl a,b       ;
      mov P1,a      ;
      setb ENABLE   ;
      mov a,#2      ;
      acall F_wait_m ;
      clr ENABLE    ; Samme indtil her

      pop B         ; Kalder Tegnet som skal skrives tilbage i B
      pop ACC       ; Kalder Tegnet som skal skrives tilbage i ACC
      ret           ; Return fra subrutine
```

Note 3, Tager den øverste nible fra de data som skal sendes til lcd displayet og "or" sammen med de 4 nederste bit fra port 1, så den nederste nible på port 1 ikke bliver ændret.

RS232 Interface

Rs232 er en asynkron seriel data standard, som bruges meget til data transmission omkring computere, og tilhørende hardware, modem o.s.v. Dataene bliver sendt serielt gennem et RX og TX linie med, hvor under –3 V til ca –10V –15V er logisk "1", og fra 3V til ca 10V-15V er logisk "0", det bliver sendt ud i et bestemt mønster for de kan blive læst uden en Clock, der ved asynkron, det foregår ved der kommer en startbit, som starter en tæller der skal være sat ens op i hver ende, såkaldt baudrate, så er de 2 data ender synkroniseret og tiden vil derefter bestemme hvornår de 8 næste databit er valide på porten, en efter en, dernæst kan der komme prioritetsbit til tjek af dataene, men de bliver ikke brugt her, og til slut kommer der en slutbit, som kan forekomme 2 gange ved nogle data kommunikationstyper.



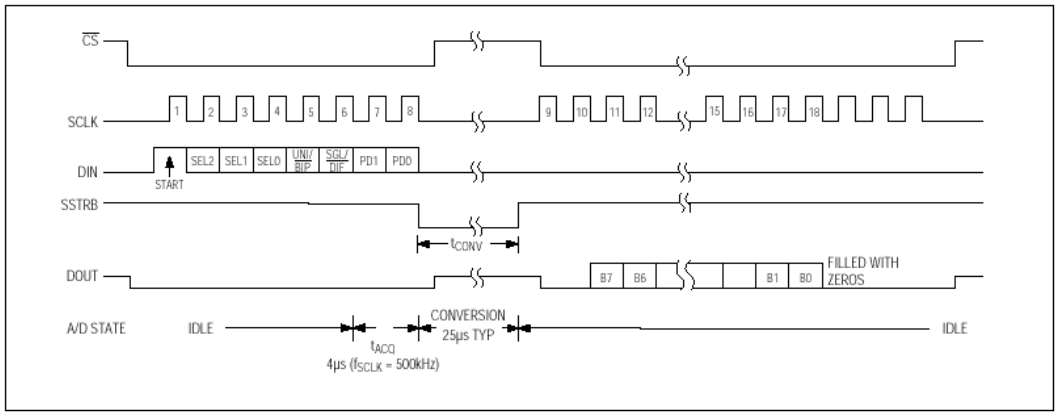
RS232 opsætning af Atmel :

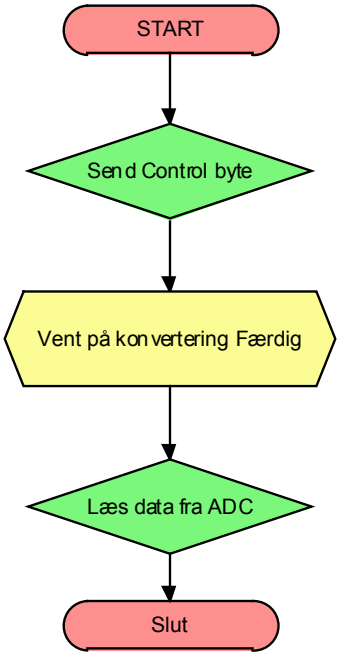
```
ORG 8000H
CALL    INIT    ; Kalder Initilisering
IGEN:
MOV A,#'E'      ; Data der skal sendes flyttes til acc
CALL SEND       ; Kalder send rutinen
MOV A,#'U'      ; Data der skal sendes flyttes til acc
CALL SEND       ; Kalder send rutinen
MOV A,#'C'      ; Data der skal sendes flyttes til acc
CALL SEND       ; Kalder send rutinen
MOV A,#' '      ; Data der skal sendes flyttes til acc
CALL SEND       ; Kalder send rutinen
JMP IGEN        ; Kører programmet igen

INIT:
MOV TMOD,#20h   ; Sæt Timer 1 for auto reload (mode 2)
MOV TCON,#41h   ; Start Timer 1 og sæt flanketrig interupt
MOV TH1,#0F3h   ; Sæt Timer 1 til ca. 2400 baud (Xtal=12MHz)
MOV SCON,#50h   ; Sæt Mode 1 og 8 bit data
RET             ; Springer tilbage til hovedprogram

SEND:
CLR TI          ; Skal slettes software mæssigt
MOV SBUF,A      ; smider byten ud på Sbuf
TXLOOP: JNB TI,TXLOOP ; Venter til starten af stopbit
RET             ; Ud af subrutine

END
```

AD Konverter :

Analog til digital konverter, laver en spænding om til et tilsvarende digitalværdi ud fra dens referencespænding. Hver bit bliver tildelt en værdi, som bliver fastlagt af hvor stor spændings område den skal kunne måle, den mest betydende bit vil have værdien som referencen har, de andre bit kommer med LSB mindre. For hver gang den analoge spænding stiger med en LSB, vil den binære code blive 1 mere.

Den AD konverter vi har brugt Max 1111, har 4 analoge indgange som kan bruges med reference til stel eller i differentiel mode, den bruger seriel data ud og ind, de serielle data ind bruges til at konfigurer konverteren med, mens de analoge værdier sendes gennem data ud i serie form, som er styret af en clock, når man har chipselected kredsen, skal der komme 8 positiv gående clockpulser, mens initaliserings byten udskrives på Data in på konverteren, dernæst skal chipselecten gå høj igen mens konverteren måler og regner det binære tal, indtil stroben går høj igen, og dataene er klar i konverteren, de 2 første clockpulser kommer der 2 dummy bit, og derefter 8 databit, og det hele går til start mode.





```
; Variable sættes lig med portbit
ENABLE          EQU P3.4
RS              EQU P1.2
RW              EQU P1.3
AD_D_OUT        EQU P1.1
AD_SCLK         EQU P1.0
AD_STTRB        EQU P3.3
AD_D_IN         EQU P1.2
ACC7            EQU 0xE7    ; Contrplbyte til ADC

LCD_RAM EQU 29h             ;  En RAM-plads defineres, for at gøre den 0?

org 00h                     ; Startadresse for koden:
ajmp start

$INCLUDE (time.inc)         ; Her følger koden for 4-bit styring
$INCLUDE (lcd_4c.inc)
start:
mov sp,#30h                 ; Stackpointer sættes til adresse 30h
;-------------------------------------------------------------------------
;   Hovedprogram
;-------------------------------------------------------------------------
MAIN:
CLR P3.2                    ; Clr Stroben

LOOP:
CALL STARTCON               ; Kalder initalisering af ADC
CALL VENT                   ; Venter til stroben går høj igen(data ready]
CALL VISRES                 ; Henter data og udskriver på LCD
JMP LOOP                    ; Genstarter programmet

STARTCON:
MOV A,#AD_CTRL_BYTE         ; Ligger control byten i ACC
MOV R0,#08                  ; Ligger antal af bit der skal sendes i R0
CALL CS                     ; Kalder CS gennem 3-8 lines Decoder
DOUT:
JNB ACC7, DATAZ             ; Hvis Bit 7 i ACC er ”0” så hop til dataZ
SETB AD_D_OUT               ; Sætter data indgangen på ADC høj
 JMP CLK1                   ; Kalder clock pulsen
 DATAZ:  CLR AD_D_OUT       ; Sætter data indgangen på ADC lav
 CLK1:   SETB AD_SCLK       ; Sætter Clocken
 CLR AD_SCLK                ; Clear Clocken
 RL A                       ; Rotere A mod venstre for at få mindre B.bit
 DJNZ R0, DOUT              ; Tjekker om den sidste bit i acc er sendt
 CALL NCS                   ; Deselectere kredsen
 CLR AD_D_OUT               ; Sikre indganen på ADC er lav
 RET                        ; Return fra Sub rutine
```

```
VENT:
clr AD_STTRB                    ; Sikrer stroben er ”0”
JNB AD_STTRB, VENT              ; Venter til Stroben går høj igen
RET                             ; Return fra Sub rutine

VISRES:
MOV R0,#02                      ; Bestemmer antal clp før data kommer
Call CS                         ; Selectere kredsen gennem 74LS138
CLK2:
SETB AD_SCLK                    ; Starter en clock puls
CLR AD_SCLK                     ; Stopper en clock puls
DJNZ R0, CLK2                   ; Bliver ved til dummy bitene er væk
; *** NOW READ THE RESULTS ***
MOV A,#00                       ; Sætter A til 00
MOV R0,#08                      ; Bestemmer antal bit der skal læsen fra ADC
CLK3:
RL A                            ; Sørger for der bliver skrevet fra MSB
SETB AD_SCLK                    ; Clock´er
JNB AD_D_IN, DINZ               ; Hvis udgangen af ADC er 0, hop til DINZ
ORL A,#01                       ; Ligger værdien 1 ind på LSB, som er hygget
DINZ:   CLR AD_SCLK             ; Ligger clocken lav
DJNZ R0, CLK3                   ; Tæller ned på R0 og hvis ik’ nul så hop clk3
call ncs                        ; Deselectere kredsen
setb p3.7                       ; Sørger for at lcd er CS
setb p3.5
acall F_LCD_init                ; Initalisere LCD Disp… Ellers virker det ik’
call F_LCD_printbd              ; skriver acc ud som tal byte
RET                             ; Return fra sub rutine

cs:                             ; Selectere ADC gennem decoderen
clr p3.7
clr p3.4
setb p3.5
RET
NCS:                            ; Deselectere ADC gennem decoderen
clr p3.7
clr p3.4
clr p3.5
RET
END                             Slutter Program
```

C Programering

Løbelys

```
#include

void wait(void)
{
;
}

void main(void)
{
unsigned int i;
unsigned char j;

while (1)
{
        for (j=0x01; j< 0x80; j<<=1)
        {
                P1 = ~j;
                for (i=0; i 0x01; j>>=1)
         {
                P1 = ~j;
                for (i=0; i ++)
                {
                        wait();
                }
        }
}
```

C programmering er et højt niveaus sprog som jeg synes skoder for vild, for jeg ikke kan finde ud af det, og jeg har heller ik' umildbart tænkt mig at sætte mig ind i det, da vi har haft en dags undervisning i det, og ik' har behov for det.

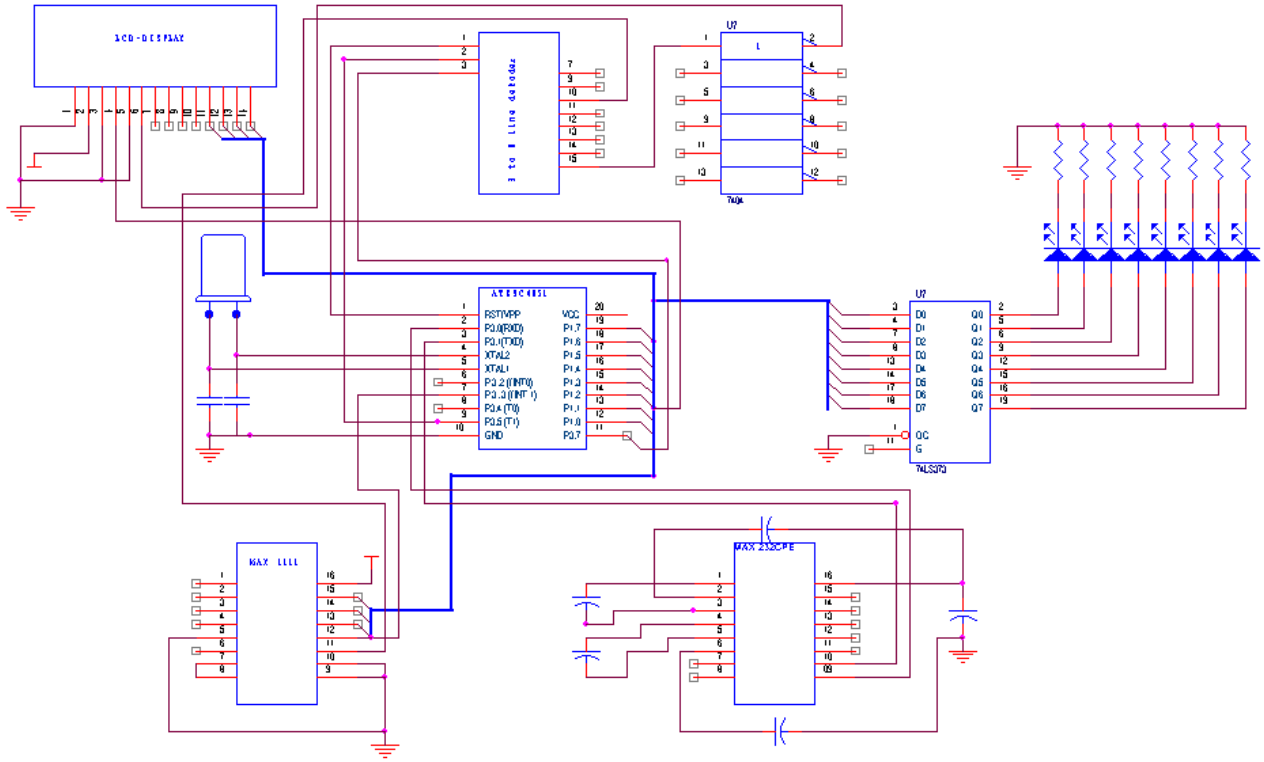
Mit-Projekt :

Indledning :

Fra starten skulle mit kredsløb kunne styre en Max038, som er en funktionsgenerator i en ic-kreds, den skulle kunne skifte funktion og område på på frekvensbåndet. Lcd displayet skulle så vise hvilket område den var i og hvilken mode den kørte.

Hardware :

Før at jeg kunne begynde at lave mit projekt ville jeg gerne have de basale ting til at virke, LCD display, lys dioder med latch, Rs232 interface of ADC, og det skulle gerne kunne køre på samme tid, så vi blev nød til at lave CS kredsløb men en 74LS138, men det gjorde det hele en del svære, og der gik meget tid til at få det til at virke. Det færdige diagram ser ca. således ud :



Det som er tegnet på diagrammet virker og kan køre sammen, alle tingene er testet. Det er lavet således man kan bruge boardet til evaluatin board bagefter og kan bygge videre på det.

Konklusion :

Projektet har været meget lærerigt, og udvidede vores viden for mikrontroller, programmering i assambler og op stille Hardware og se problemerne i det.
Tiden til projektet har været lige knapt, til hvad der bliver forventet.
Jeg synes det er rimelig for dårligt, vi skal anstrenge os så meget får at kan få lov til at bruge et digital scope, som egentlig er vigtigt for at kunne måle ordentligt på digitale kredsløb. Jeg synes der godt kunne være blevet lavet lidt mere ud af Inc filerne med de rutiner i som vi bruger, for bedre at kunne forstå hvad der sker, i stedet for vi får de færdige programmer som vi bare skal bruge, og ik' ved hvorfor de skal bruges og hvorfor forskellige ting sker. Problemet med mit projekt kan kun værre mit eget problem jeg ikke er blevet færdig med, men jeg synes det var mere vigtigt at forstå hvor lcd displayet skriver noget og hvordan man bruger det, ligesådan med ADC'en, som virker i min opstilling.