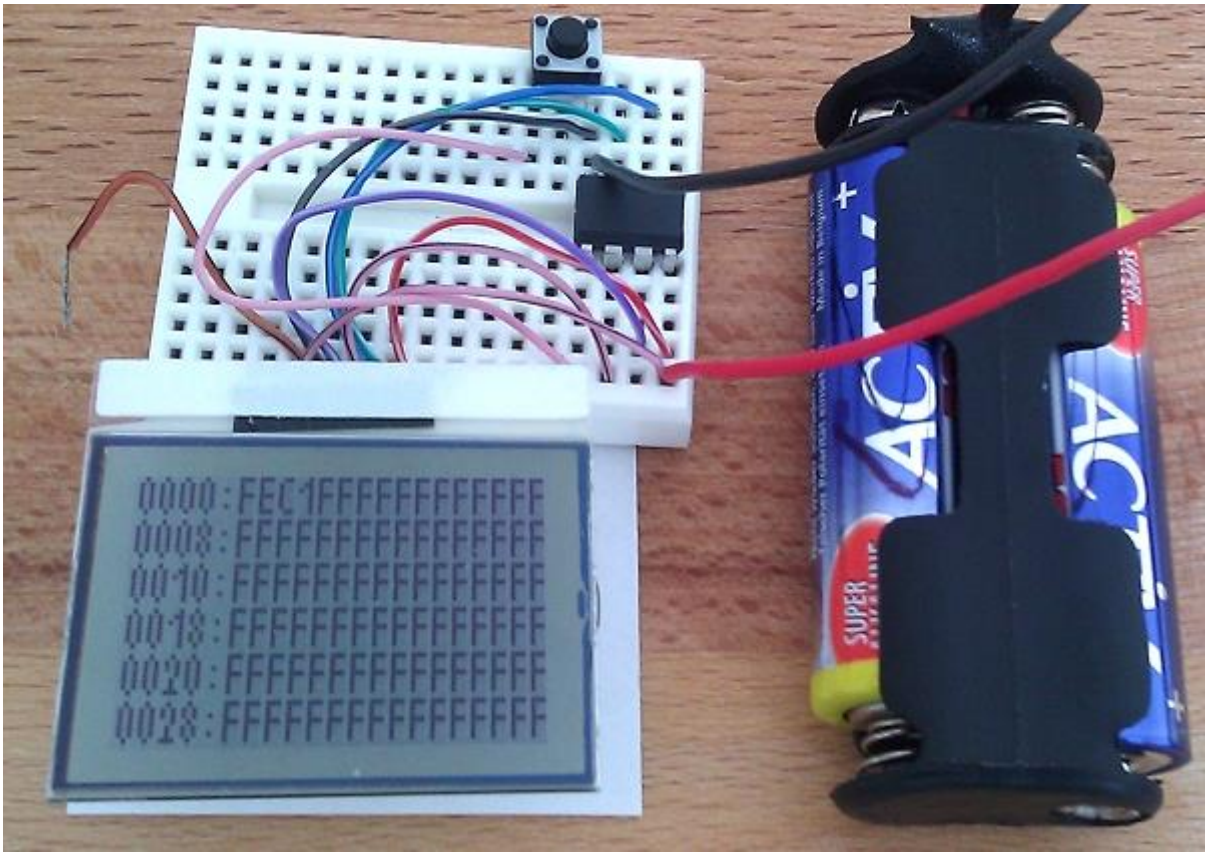
Hex-Editor

von Thomas Baum

Elektronik-Labor Projekte AVR T13-Contest



Inspiriert von der "TPS" und der händigen Programmierung aus dem "Assembler-Kurs" habe ich die Tage mal meinen Hexdump-Code zu einem rudimentären aber nutzbaren Hex-Editor aufgebohrt. Damit lässt sich der kleine AVR in Verbindung mit einem SPI-Nokia-Display ohne sonstige Hardware programmieren. Nicht einfach aber dafür mit vollem Zugriff auf die Hardware.

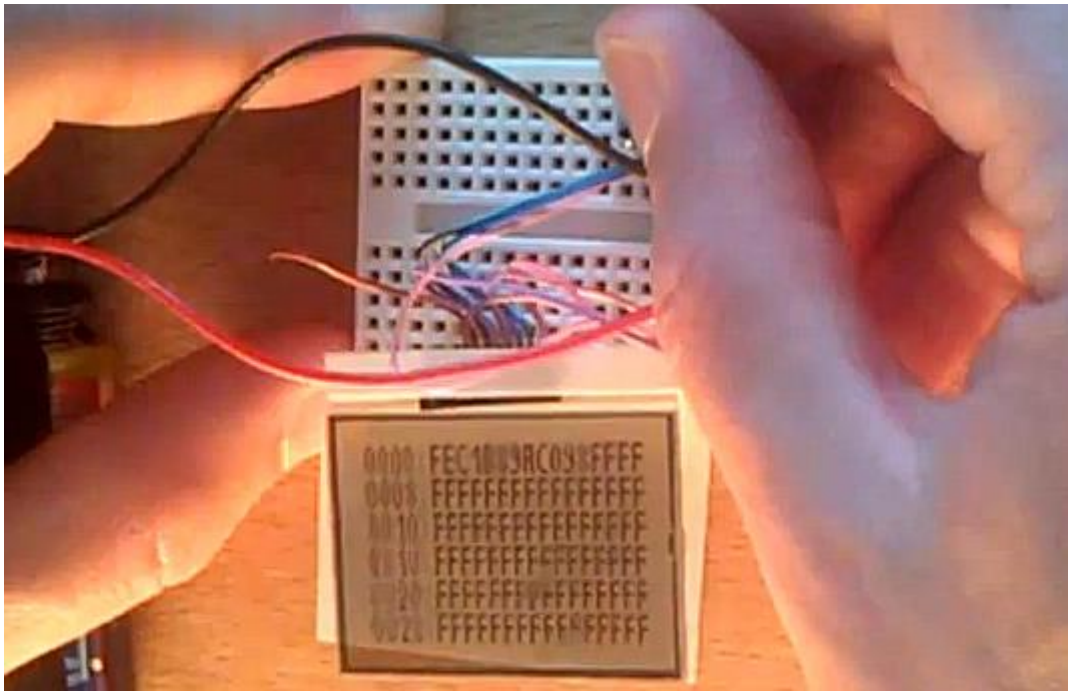
Verwendete Hardware:
Attiny13a
Nokia 5510 Display

Bedienung:
Editormodus starten: Port 0 beim Start auf Masse ziehen
Eingabe: kurz-weiter; lang-bestätigen
Organisation: Seitenwahl->Zeilenwahl->Wortwahl->Worteingabe->Speichern
Abbruch: Stromversorgung trennen

Aufbau und Funktion:
Der Programmcode des Editors befindet sich am Ende des Speichers. Das hat den Vorteil, dass man mit dem eigenen Programm auch die Interruptvektortabelle definieren kann. Der Befehlszähler beginnt jedoch immer bei 0. Deshalb ist der erste Befehl ein Sprung in den Programmcode des Editors. Zur Vereinfachung zeigt dieser auf die letzte Adresse im Speicher,

die man sich beim Überschreiben des Resetinterrupts besser merken kann, will man nochmal beim Neustart in den Editormodus. :)
Selbst geschriebene Programme können nach dem ersten Wort angefügt werden, oder man kümmert sich selbst in seinem Code um den Aufruf des Editors. Die einzige Beschränkung gegenüber einem blanken Attiny13 ist:
- nur noch 25% freier Programmspeicher verfügbar.

Wichtig: In den Fusebits muss die Selbstprogrammierung aktiviert werden. (Fuse 6AEF statt 6AFF)
Auf eine Änderung des Reset Pins wurde bewusst verzichtet. Damit lässt sich der Controller auch noch per ISP programmieren.



Youtube-Video: **http://youtu.be/_kgd4lg4PgM**

Download: **avr-hex-editor.zip**

```
;Onboard AVR-Hex-Editor (attiny13) Thomas Baum
(th.baum@online.de)
;Display Nokia 5510
;                ____
;             -|    |-VCC (3V)
;     LCD_SDIN-|    |-LCD_SCE
;     LCD_SCLK-|    |-LCD_DC
;     GND      -|__  |-INPUT(Button nach Masse)
;
;kurz-weiter; lang-bestÃ¤tigen
;Seitenwahl->Zeilenwahl->Wortwahl->Worteingabe->Speichern

.device ATtiny13A            ;Fuse 6AEF
```

```
.equ    INPUT        = 0        ;Pinbelegung
.equ    LCD_DC        = 1
.equ    LCD_SCE        = 2
.equ    LCD_SDIN    = 3
.equ    LCD_SCLK    = 4

.org 0x0000
flash_start:
        rjmp    flash_end        ;Sprung zum Editor

user_prog:                ;Userspace
;---------------------------------------
;---------------Userspace---------------
;---------------------------------------
;---
;---------------------------------------
.org 0x0080                ;Startadresse Editorcode
    rjmp    user_prog        ;Lauf in Editor verhindern
init:
    ldi    r16, RAMEND        ;Stackinitalisierung
    out    spl, r16
    sbi    PORTB, INPUT        ;Pullup definieren
    cbi    DDRB, INPUT        ;Definition der Ports
    sbi    DDRB, LCD_DC
    sbi    DDRB, LCD_SCE
    sbi    DDRB, LCD_SDIN
    sbi    DDRB, LCD_SCLK

start_prog:                ;Startmenu
    rcall    delay
    in    r16, PINB        ;Pin0 abfragen
    andi    r16, 0x01
    cpi    r16, 0x00        ;ist Pin0 0?
    brne    start_user_prog        ;wenn nicht starte
Userprogramm
    rcall    button            ;Taster absteigende Flake
abfangen
    rjmp    reset
start_user_prog:
    rjmp    user_prog        ;Adressierung nur Ã¼ber rjmp
mÃ¶glich
reset:                    ;Variablen zurÃ¼cksetzen
    clr    r31
    clr    r30
    clr    r10            ;Seitenvariable
    clr    r02            ;Zeilenvariable
    ldi    r16, 20            ;Offset Zeilennummerierung
    mov    r03, r16        ;word_pos
    clr    r04            ;word_add
    clr    r06            ;Eingabe
    clr    r07            ;Eingabe
```

```
main:
    rcall    init_display        ;Displayinitialisierung
    ldi    r16, 0x00         ;Ausgabeposition
    rcall    set_x_position        ;x auf 0
    ldi    r16, 0x00
    rcall    set_y_position        ;y auf 0
    rcall    draw_init
    rcall    out_page         ;Ganze Seite ausgeben
    rcall    draw_finish
    rcall    button            ;Eingabe abfragen
    cpi    r16, 0x02         ;langer Tastendruck?
    breq    line             ;gehe Zeile auswÃ¤hlen
    inc    r10
    mov    r16, r10
    cpi    r16, 22            ;Ãœberlauf?
    brne    main             ;Nein, dann nÃ¤chste Seite
    ldi    r31, 0x00         ;Ãœberlaufkorrektur
    ldi    r30, 0x00
    ldi    r16, 0x00
    mov    r10, r16
    rjmp    main

line:                    ;Zeile auswÃ¤hlen
    rcall    init_display
    ldi    r16, 0x00
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0xff         ;Zeilenmarkierung laden
    rcall    spi_out             ;Zeilenmarkierung zeichnen
    rcall    draw_finish
    rcall    button            ;Zeile auswÃ¤hlen?
    cpi    r16, 0x02         ;ja, dann Wort wÃ¤hlen
    breq    word
    inc    r02
    rcall    init_display         ;alte Markierung lÃ¶schen
    ldi    r16, 0x00
    rcall    set_x_position
    mov    r16, r02
    dec    r16
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0x00         ;leere Markierung
    rcall    spi_out             ;Zeilenmarkierung zeichnen
    mov    r16, r02
    cpi    r16, 6             ;ZeilenÃ¼berlauf
    brne    line
    ldi    r16, 0x00         ;Werte korrigieren
    mov    r02, r16
    rjmp    line
```

```
word:                   ;Wort auswÃ¤hlen
    rcall    init_display
    mov    r16, r03
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0xff        ;Wortmarkierung laden
    rcall    spi_out            ;ausgeben
    rcall    draw_finish
    rcall    button           ;Eingabe abfragen
    cpi    r16, 0x02
    breq    edit
    mov    r16, r03
    ldi    r17, 16
    add    r16, r17
    mov    r03, r16
    inc    r04
    rcall    init_display        ;Alte Markierung lÃ¶schen
    mov    r16, r03
    ldi    r17, 16
    sub    r16, r17
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0x00
    rcall    spi_out
    rcall    draw_finish
    mov    r16, r03
    cpi    r16, 84
    brne    word
    ldi    r16, 20            ;Ãœberlaufkorrektur
    mov    r03, r16
    ldi    r16, 0x00
    mov    r04, r16
    rjmp    word

edit:                   ;Worteinagbe
    ldi    r16, 0x00
    mov    r07, r16        ;Ergebnis zurÃ¼cksetzen
    mov    r06, r16

    rcall    get_nibble        ;1. Zeichen einlesen
    andi    r20, 0x0f
    swap    r20
    mov    r07, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;2. Zeichen einlesen
    andi    r20, 0x0f
```

```
    or    r07, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;3. Zeichen einlesen
    andi    r20, 0x0f
    swap    r20
    mov    r06, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;4. Zeichen einlesen
    andi    r20, 0x0f
    or    r06, r20
    ldi    r16, 0x04
    add    r03, r16

add_page:                ;Seitenadresse ermitteln
    clr    r24
    clr    r25
    mov    r16, r10
add_add_p:
    cpi    r16, 0x00
    breq    add_line
    adiw    r25:r24, 48        ;48 Byte pro Seite
    dec    r16
    rjmp    add_add_p

add_line:                ;Zeilenadresse ermitteln
    mov    r16, r02
add_add:
    cpi    r16, 0x00
    breq    ok1
    adiw    r25:r24, 0x08        ;addiere pro Zeile 8
    dec    r16
    rjmp    add_add
ok1:
    mov    r16, r04
add_add1:
    cpi    r16, 0x00
    breq    ok2
    adiw    r25:r24, 0x02        ;addiere pro Wort 2
    dec    r16
    rjmp    add_add1
ok2:
    mov    r31, r25        ;lade Pageadresse
    mov    r30, r24
    andi    r30, 0b11100000        ;Wortadresse auf 0

    ldi    r21, 0x00
    mov    r22, r24        ;geÃ¤nderte Wortadresse ermitteln
    andi    r22, 0x1f
```

```
    lsr    r22            ;wort bit 1, 0 ist byteadresse
page_l:
    cpi    r21, 16            ;fÃ¼r alles 16 WÃ¶rter
    breq    delete

    cp    r21, r22
    brne    fill_old
    mov    r00, r07         ;geÃ¤nderte Werte laden
    mov    r01, r06
    adiw    r31:r30, 0x02        ;z manuell hochzÃ¤hlen
    rjmp    fill_add
fill_old:                ;lade alte Werte
    lpm    r00, Z+
    lpm    r01, Z+
fill_add:
    mov    r28, r30         ;rette Adresszeiger
    mov    r29, r31
    mov    r30, r21
    lsl    r30             ;Linksshift
    ldi    r16, 0b00000001
    out    spmcsr,    r16
    spm                ;WÃ¶rter in Puffer laden
    mov    r30, r28
    mov    r31, r29
    inc    r21
    rjmp    page_l

delete:                    ;Page lÃ¶schen und neu schreiben
    mov    r31, r25         ;Pageadresse laden
    mov    r30, r24
    andi    r31, 0b00000011         ;Page maskieren
    andi    r30, 0b11100000
    ldi    r16, 0b00000011         ;LÃ¶schbefehl laden
    out    spmcsr,    r16
    spm                ;lÃ¶schen
    ldi    r16, 0b00000101         ;Schreibbefehl laden
    out    spmcsr,    r16
    spm                ;schreiben
    rcall    button             ;User bereit fÃ¼r neue Anzeige?
    rjmp reset             ;Editor neu starten

get_nibble:                ;Zeichen einlesen
    clr    r20             ;Ausgabe 0 setzen
s_get_nibble:
    rcall    init_display
    mov    r16, r03
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    mov    r16, r20
    rcall    out_nibble         ;Zeichen ausgeben
```

```
    rcall    draw_finish
    rcall    button            ;Eingabe abfragen
    cpi    r16, 0x02         ;lange gedrÃ¼ckt?
    breq    f_get_nibble        ;Eingabe beenden
    inc    r20             ;EingabezÃ¤hler++
    ldi    r16, 0x0f
    and    r20, r16
    rjmp    s_get_nibble
f_get_nibble:
    ret

button:                     ;Taster abfragen
    in    r16, PINB         ;PortB einlesen
    andi    r16, 0x01         ;Port0 separieren
    cpi    r16, 0x00         ;wurde auf Masse gezogen?
    brne    button            ;nein dann weiter abfragen
    rcall    delay            ;ja dann kurz warten
    in    r16, PINB         ;erneut abfragen
    andi    r16, 0x01
    cpi    r16, 0x00         ;noch auf Masse?
    breq    ok             ;ja dann langes Event
    ldi    r16, 0x01         ;RÃ¼ckgabewert fÃ¼r "kurz"
    ret
ok:                     ;langes drÃ¼cken erkannt
    rcall    delay            ;delay
    in    r16, PINB         ;PortB einlesen
    andi    r16, 0x01
    cpi    r16, 0x00         ;noch auf Masse?
    breq    ok             ;ja dann warte
    ldi    r16, 0x02         ;RÃ¼ckgabewert fÃ¼r "lang"
    ret

out_page:                 ;Seite ausgeben (6 Zeilen)
    cpi    r16, 0x00         ;erste Seite?
    breq    load_p
load_add:
    adiw    r31:r30, 48         ;pro Seite 48 Byte weiter
    dec    r16
    brne    load_add
load_p:
    ldi    r19, 0x06
    mov    r26, r30         ;Anfangsadresse laden
    mov    r27, r31
p_loop:
    rcall    out_line
    adiw    r27:r26, 8         ;Adresse fÃ¼r die nÃ¤chsten 8 Byte
    dec    r19
    brne    p_loop
    ret

delay:                     ;Delay mit 2 Schleifen
    ldi    r16, 0x00
```

```
    ldi    r17, 0x00
d_wait:
    inc    r16
    cpi    r16, 0x00
    brne    d_wait
    inc    r17
    cpi    r17, 0xa0
    brne    d_wait
    ret

out_line:                   ;Zeile ausgeben
    mov    r16, r27         ;Adresse laden (H)
    rcall    out_byte       ;Adresse ausgeben
    mov    r16, r26         ;Adresse laden (L)
    rcall    out_byte       ;Adresse ausgeben
    ldi    r16, 0x00        ;":" bauen und ausgeben
    rcall spi_out
    ldi    r16, 0x00
    rcall spi_out
    ldi    r16, 0x24
    rcall spi_out
    ldi    r16, 0x00
    rcall spi_out
    ldi    r18, 0x08         ;ZÃ¤hler fÃ¼r 8 Byte
l_loop:
    lpm    r16, Z+            ;Byte laden
    rcall    out_byte         ;Byte ausgeben
    dec    r18
    brne    l_loop
    ret

out_byte:                  ;Byte ausgeben
    mov    r00, r16          ;Zeichen retten
    swap    r16              ;oberstes Nibble
    andi    r16, 15              ;Maskierung
    mov    r28, r30          ;rette Z-Register
    mov    r29, r31          ;rette Z-Register
    rcall    out_nibble         ;Ausgabe
    mov    r16, r00          ;unterstes Nibble
    andi    r16, 15              ;Maskierung
    rcall    out_nibble         ;Ausgabe
    mov    r30, r28          ;rette Z-Register
    mov    r31, r29          ;rette Z-Register
    ret

draw_init:                 ;DatenÃ¼bertragung vorbereiten
    sbi    PORTB, LCD_DC         ;LCD Kommandomodus setzen
    cbi    PORTB, LCD_SCE         ;LCD aktivieren
    ret

draw_finish:                ;LCD Schreibvorgang abschlieÃŸen
    sbi    PORTB, LCD_SCE         ;LCD deaktivieren
```

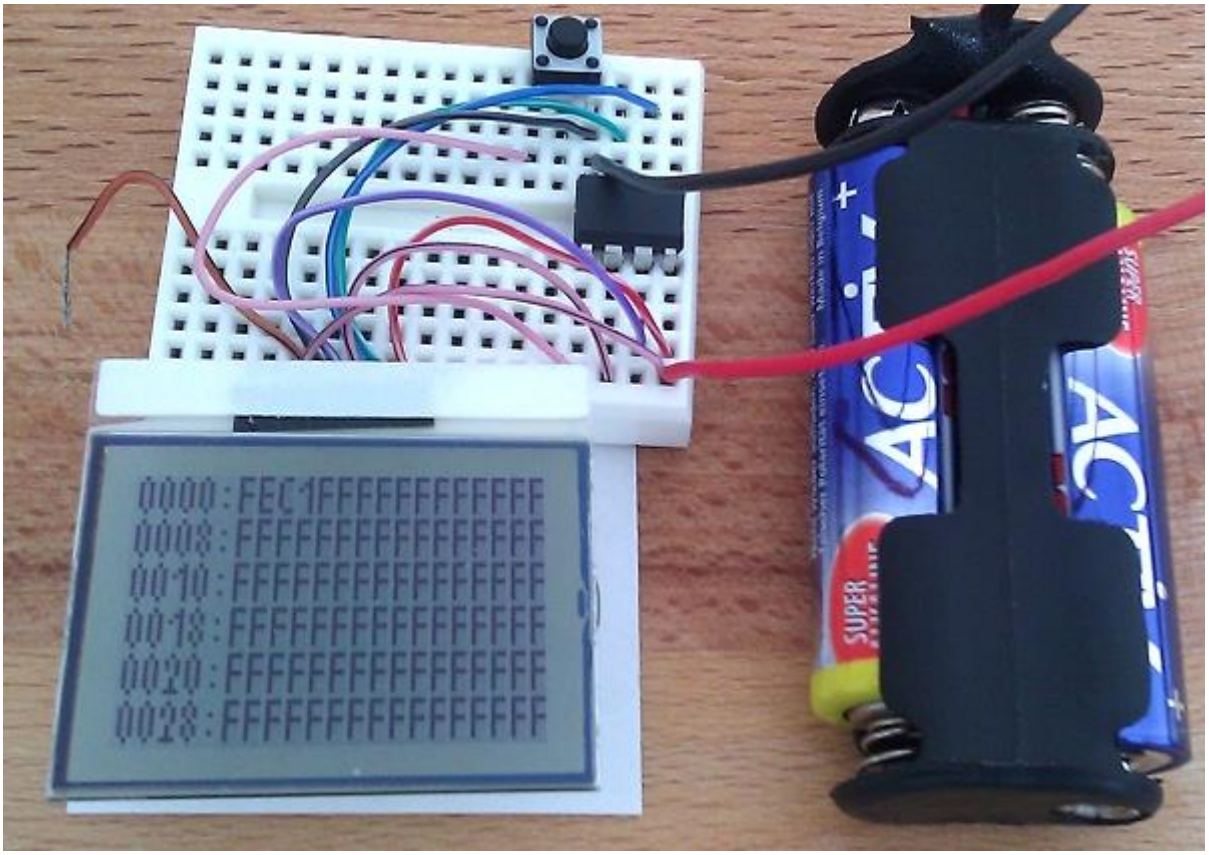
```
    ret

init_display:
    ldi    r16, 0x21        ;Displayinitialisierung
    rcall    lcd_write_cmd
    ldi    r16, 0xD0
    rcall    lcd_write_cmd
    ldi    r16, 0x04
    rcall    lcd_write_cmd
    ldi    r16, 0x13
    rcall    lcd_write_cmd
    ldi    r16, 0x20
    rcall    lcd_write_cmd
    ldi    r16, 0x0C
    rcall    lcd_write_cmd

    ldi    r16, 0x21        ;Kontrasteinstellung
    rcall    lcd_write_cmd
    ldi    r16, 0x80 | 0x3C    ;Kontrastwert laden
    rcall    lcd_write_cmd
    ldi    r16, 0x20
    rcall    lcd_write_cmd

    ret

set_x_position:                ;LCD x-Position setzen
    ori    r16, 0x80        ;Wertnormalisierung
    rcall    lcd_write_cmd        ;Daten schreiben
    ret

set_y_position:                ;LCD y-Position setzen
    ori    r16, 0x40        ;Wertnormalisierung
    rcall    lcd_write_cmd        ;Daten schreiben
    ret

spi_out:                ;Software SPI
    cbi    PORTB, LCD_SCLK        ;Clock auf 0 setzen
    ldi    r17, 0x08        ;SchleifenzÃ¤hler fÃ¼r 8 Bit
spi_out_byte:                ;Byte ausgeben
    sbrc    r16, 0x07        ;PrÃ¼fe ob Bit 7 nicht gesetzt
    sbi    PORTB, LCD_SDIN        ;Datenleitung auf 1 setzen
    sbrs    r16, 0x07        ;PrÃ¼fe ob Bit 7 gesetzt
    cbi    PORTB, LCD_SDIN        ;Datenleitung auf 0 setzen
    sbi    PORTB, LCD_SCLK        ;Clock auf 1 setzen
    cbi    PORTB, LCD_SCLK        ;Clock auf 0 setzen
    lsl    r16            ;Shift fÃ¼r nÃ¤chstes Bit
    dec    r17            ;SchleifenzÃ¤hler--
    brne    spi_out_byte        ;FÃ¼r alle Ã¼brigen Bits
    ret

lcd_write_cmd:                ;LCD Kommando schreiben
    cbi    PORTB, LCD_DC        ;LCD Kommandomodus setzen
```

```
    rcall    lcd_out            ;Daten schreiben
    ret

lcd_write_data:                 ;LCD Daten schreiben
    sbi    PORTB, LCD_DC        ;LCD Datenmodus setzen
    rcall    lcd_out            ;Daten schreiben
    ret

lcd_out:
    cbi    PORTB, LCD_SCE        ;LCD aktivieren
    rcall    spi_out             ;Daten ausgeben
    sbi    PORTB, LCD_SCE        ;LCD deaktivieren
    ret

out_nibble:                 ;Nibble ausgeben
    add    r16, r16
    add    r16, r16         ;Zeichenadressierung
    ldi    r30, LOW(data*2)    ;Offset laden
    ldi    r31, HIGH(data*2)
    add    r30, r16         ;Zeichenposition addieren
    ldi    r16, 0x00
    adc    r31, r16
    lpm    r16, Z+          ;Zeichen laden
    rcall spi_out           ;Zeichen ausgeben
    lpm    r16, Z+          ;...
    rcall spi_out
    lpm    r16, Z+
    rcall spi_out
    lpm    r16, Z+
    rcall spi_out
    ret

data:                     ;Zeichensatz
    .db    0x00, 0x3E, 0x41, 0x3E    ;0
    .db    0x00, 0x04, 0x02, 0x7F    ;1
    .db    0x00, 0x46, 0x71, 0x4E    ;2
    .db    0x00, 0x49, 0x49, 0x36    ;3
    .db    0x00, 0x0C, 0x0A, 0x7F    ;4
    .db    0x00, 0x4F, 0x49, 0x31    ;5
    .db    0x00, 0x3E, 0x49, 0x31    ;6
    .db    0x00, 0x01, 0x79, 0x0F    ;7
    .db    0x00, 0x36, 0x49, 0x36    ;8
    .db    0x00, 0x46, 0x49, 0x3E    ;9
    .db    0x00, 0x7F, 0x09, 0x7F    ;A
    .db    0x00, 0x7F, 0x49, 0x36    ;B
    .db    0x00, 0x3E, 0x41, 0x41    ;C
    .db    0x00, 0x7F, 0x41, 0x3E    ;D
    .db    0x00, 0x7F, 0x49, 0x49    ;E
    .db    0x00, 0x7F, 0x09, 0x09    ;F

flash_end:
    rjmp    init             ;jmp to editor
```

Hex-Editor

http://www.elektronik-labor.de/AVR/T13contest/HexEditor.html

Design by Thomas Baum
Elektronik-Labor Projekte AVR T13-Contest



Inspired by "TPS" and the “special hands-on programming” from the "assembler course", I've converted my hexdump code into a rudimentary but usable hex editor. This makes it possible to program the small AVR Tiny13 connected to a SPI Nokia display without any other hardware.
Not easy but with full access to the hardware.

Hardware used:
Breadboard
Attiny13a
Nokia 5510 Display
Switch
Battery

How to control it:
To start editor mode: Pull Port 0 to ground during startup
Input: short push forward; long push to confirm
Organization: page selection to -> to line selection->to word selection-> to word input-> save
Finish: Disconnect power

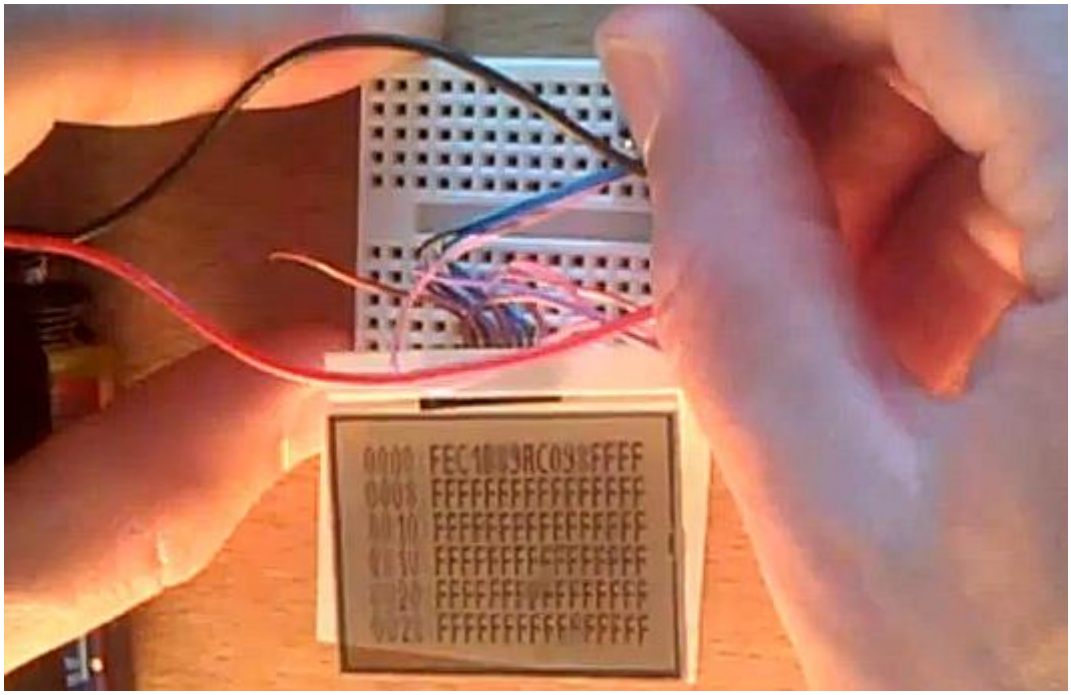
Layout and function:

The program code of the editor is located at the end of memory. This has the advantage that you can also define the interrupt vector table with your own program. However, the instruction counter always starts at address 0. Therefore, the first instruction there is a jump to the starting address of the program code of the editor.
For simplification, this points to the last address in memory, which can be better remembered when overwriting the reset interrupt, if you want to re-enter the editor mode. :)

Own programs can be added after this first word, or you might want to take care of calling the editor via your own code. The only limitation compared to a bare Attiny13 is:
- now only 25% of free program memory space is left – the rest taken by the editor code.

Wichtig: In den Fusebits muss die Selbstprogrammierung aktiviert werden. (Fuse 6AEF statt 6AFF)
Auf eine Änderung des Reset Pins wurde bewusst verzichtet. Damit lässt sich der Controller auch noch per ISP programmieren.

Important: Self-programming must be activated via the fusebit setting. (**Fuse code 6AEF** instead of 6AFF)
Regarding a change of the reset pin: this was deliberately omitted. As result his allows the controller to be programmed via ISP.



Youtube-Video: **http://youtu.be/_kgd4lg4PgM**

Download: **avr-hex-editor.zip**

```
;Onboard AVR-Hex-Editor (attiny13) Thomas Baum
(th.baum@online.de)
;Display Nokia 5510
;                ____
;             1 |   | 8-VCC (3V)
;    LCD_SDIN-2 |   | 7-LCD_SCE
;    LCD_SCLK-3 |   | 6-LCD_DC
;    GND     -4 |__| 5-INPUT(Button to Ground)
;
; short push means continue; long push means confirm
: Page selection->Line sel.->Word sel.->Word Input->Save

.device ATtiny13A            ;Fuse 6AEF

.equ    INPUT        = 0        ;Pin connections
.equ    LCD_DC       = 1
.equ    LCD_SCE      = 2
.equ    LCD_SDIN     = 3
.equ    LCD_SCLK     = 4

.org 0x0000
flash_start:
        rjmp    flash_end        ;Jump to Editor

user_prog:                ;Userspace
;---------------------------------------
;---------------Userspace---------------
;---------------------------------------
;---
;---------------------------------------

.org 0x0080                  ;Start address Editor code
    rjmp    user_prog        ;Avoid jump to Editor
init:
    ldi    r16, RAMEND       ;Initialize stack
    out    spl, r16
    sbi    PORTB, INPUT      ;Define internal Pullups
    cbi    DDRB, INPUT       ;Define Port Pin Direction
    sbi    DDRB, LCD_DC
    sbi    DDRB, LCD_SCE
    sbi    DDRB, LCD_SDIN
    sbi    DDRB, LCD_SCLK

start_prog:                   ;Start menu
    rcall    delay
    in    r16, PINB           ;Pin0 status?
    andi    r16, 0x01
    cpi    r16, 0x00          ;is Pin0 0?
    brne    start_user_prog   ;if not, start User Program
    rcall    button           ;catch switch level going to LOW
    rjmp    reset
start_user_prog:
```

```
    rjmp    user_prog        ;Addressing only via rjump
reset:                      ;Reset Variables
    clr    r31
    clr    r30
    clr    r10              ;Page variable
    clr    r02              ;Line variable
    ldi    r16, 20              ;Offset for line numbers
    mov    r03, r16          ;word_pos
    clr    r04              ;word_add
    clr    r06              ;Input
    clr    r07              ;Input

main:
    rcall    init_display         ;Initialize Display
    ldi    r16, 0x00          ; Output position
    rcall    set_x_position         ;Set x to 0
    ldi    r16, 0x00
    rcall    set_y_position         ;Set y to 0
    rcall    draw_init
    rcall    out_page          ;Output full page
    rcall    draw_finish
    rcall    button             ;Question Input
    cpi    r16, 0x02          ;Long Push?
    breq    line              ;Then go to Line Selection
    inc    r10
    mov    r16, r10
    cpi    r16, 22             ;Overrun?
    brne    main              ;No, then next page
    ldi    r31, 0x00          ;Overrun correction
    ldi    r30, 0x00
    ldi    r16, 0x00
    mov    r10, r16
    rjmp    main

line:                      ;Select Line
    rcall    init_display
    ldi    r16, 0x00
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0xff          ;Load Line Marking
    rcall    spi_out             ;Draw Line Marking
    rcall    draw_finish
    rcall    button             ;Select row ?
    cpi    r16, 0x02          ;Yes, then select Word
    breq    word
    inc    r02
    rcall    init_display         ;Delete old Marking
    ldi    r16, 0x00
    rcall    set_x_position
    mov    r16, r02
```

```
    dec    r16
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0x00        ;Empty marking
    rcall    spi_out            ;Draw Line marking
    mov    r16, r02
    cpi    r16, 6             ;Row overrun
    brne    line
    ldi    r16, 0x00        ;Corct values
    mov    r02, r16
    rjmp    line

word:                    ;Select Word
    rcall    init_display
    mov    r16, r03
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0xff        ;Load Word marking
    rcall    spi_out            ;Output
    rcall    draw_finish
    rcall    button            ;Check Input
    cpi    r16, 0x02
    breq    edit
    mov    r16, r03
    ldi    r17, 16
    add    r16, r17
    mov    r03, r16
    inc    r04
    rcall    init_display        ;Delete old Marking
    mov    r16, r03
    ldi    r17, 16
    sub    r16, r17
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    ldi    r16, 0x00
    rcall    spi_out
    rcall    draw_finish
    mov    r16, r03
    cpi    r16, 84
    brne    word
    ldi    r16, 20            ;Overrun correction
    mov    r03, r16
    ldi    r16, 0x00
    mov    r04, r16
    rjmp    word

edit:                    ;Word Input
    ldi    r16, 0x00
```

```
    mov    r07, r16        ;Reset the result
    mov    r06, r16

    rcall    get_nibble        ;1. Input first character
    andi    r20, 0x0f
    swap    r20
    mov    r07, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;2. Input second character
    andi    r20, 0x0f
    or    r07, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;3. Input third character
    andi    r20, 0x0f
    swap    r20
    mov    r06, r20
    ldi    r16, 0x04
    add    r03, r16

    rcall    get_nibble        ;4. Input fourth character
    andi    r20, 0x0f
    or    r06, r20
    ldi    r16, 0x04
    add    r03, r16

add_page:                ;Calculate Page address
    clr    r24
    clr    r25
    mov    r16, r10
add_add_p:
    cpi    r16, 0x00
    breq    add_line
    adiw    r25:r24, 48        ;48 Bytes per page
    dec    r16
    rjmp    add_add_p

add_line:                ;Calculate Row address
    mov    r16, r02
add_add:
    cpi    r16, 0x00
    breq    ok1
    adiw    r25:r24, 0x08        ;Add 8 per row
    dec    r16
    rjmp    add_add
ok1:
    mov    r16, r04
add_add1:
    cpi    r16, 0x00
```

```
    breq    ok2
    adiw    r25:r24, 0x02        ;Add 2 per Word
    dec    r16
    rjmp    add_add1
ok2:
    mov    r31, r25        ;Load Page address
    mov    r30, r24
    andi    r30, 0b11100000        ;Set Word address to 0

    ldi    r21, 0x00
    mov     r22, r24        ;Calculate modified Word address
    andi    r22, 0x1f
    lsr    r22            ;word bit 1, 0 is byte address
page_l:
    cpi    r21, 16            ; for all 16 Words
    breq    delete

    cp    r21, r22
    brne    fill_old
    mov    r00, r07        ;Load changed values
    mov    r01, r06
    adiw    r31:r30, 0x02        ;increment z manually
    rjmp    fill_add
fill_old:                ;Load old values
    lpm    r00, Z+
    lpm    r01, Z+
fill_add:
    mov    r28, r30        ;Save address pointer
    mov    r29, r31
    mov    r30, r21
    lsl    r30            ;Shift left
    ldi    r16, 0b00000001
    out    spmcsr,    r16
    spm                ;Load Words into buffer
    mov    r30, r28
    mov    r31, r29
    inc    r21
    rjmp    page_l

delete:                    ;Delete Page and write new
    mov    r31, r25        ;Load Page address
    mov    r30, r24
    andi    r31, 0b00000011        ;Mark Page
    andi    r30, 0b11100000
    ldi    r16, 0b00000011        ;Load Delete Command
    out    spmcsr,    r16
    spm                ;Delete
    ldi    r16, 0b00000101        ;Load Write Command
    out    spmcsr,    r16
    spm                ;Write
    rcall    button            ;Is User ready for new Display?
    rjmp reset            ;Restart Editor
```

```
get_nibble:                ;Input Character
    clr    r20            ;Set Output to 0
s_get_nibble:
    rcall    init_display
    mov    r16, r03
    rcall    set_x_position
    mov    r16, r02
    rcall    set_y_position
    rcall    draw_init
    mov    r16, r20
    rcall    out_nibble        ;Output Character
    rcall    draw_finish
    rcall    button            ;Check Input data
    cpi    r16, 0x02         ;Was it a long push?
    breq    f_get_nibble         ;Stop Input
    inc    r20             ;Input counter ++
    ldi    r16, 0x0f
    and    r20, r16
    rjmp    s_get_nibble
f_get_nibble:
    ret

button:                    ;Question switch status
    in    r16, PINB         ;Input PortB
    andi    r16, 0x01          ;Port0 separieren
    cpi    r16, 0x00         ;had been pulled to 0 ?
    brne    button             ;no, then continue Input
    rcall    delay             ;yes, then wait for Short
    in    r16, PINB         ;question again
    andi    r16, 0x01
    cpi    r16, 0x00         ;Still connected to Ground?
    breq    ok             ;Yes, then it is a long activity
    ldi    r16, 0x01         ;Value for “short”
    ret
ok:                    ;Long push recognized
    rcall    delay             ;delay
    in    r16, PINB         ;Read PortB
    andi    r16, 0x01
    cpi    r16, 0x00         ;still connected to Ground?
    breq    ok             ;Yes, then wait
    ldi    r16, 0x02         ;Value for “long”
    ret

out_page:                 ;Output Page ( 6 rows )
    cpi    r16, 0x00         ;first Page?
    breq    load_p
load_add:
    adiw    r31:r30, 48        ;Step 48 Bytes further per Page
    dec    r16
    brne    load_add
load_p:
```

```
    ldi    r19, 0x06
    mov    r26, r30        ;Load start address
    mov    r27, r31
p_loop:
    rcall    out_line
    adiw    r27:r26, 8        ;Address for the next 8 Bytes
    dec    r19
    brne    p_loop
    ret

delay:                    ;Dealy using 2 loops
    ldi    r16, 0x00
    ldi    r17, 0x00
d_wait:
    inc    r16
    cpi    r16, 0x00
    brne    d_wait
    inc    r17
    cpi    r17, 0xa0
    brne    d_wait
    ret

out_line:                  ;Output Row
    mov    r16, r27        ;Load Address (H)
    rcall    out_byte      ;Adresse ausgeben
    mov    r16, r26        ;Load Address (L)
    rcall    out_byte      ;Output Address
    ldi    r16, 0x00       ;Build ":" and outout it
    rcall spi_out
    ldi    r16, 0x00
    rcall spi_out
    ldi    r16, 0x24
    rcall spi_out
    ldi    r16, 0x00
    rcall spi_out
    ldi    r18, 0x08        ;Counter for 8 Bytes
l_loop:
    lpm    r16, Z+           ;Load Byte
    rcall    out_byte        ;Byte ausgeben
    dec    r18
    brne    l_loop
    ret

out_byte:                 ;Output Byte
    mov    r00, r16         ;Save Character
    swap    r16             ;Most left Nibble
    andi    r16, 15             ;Masking
    mov    r28, r30         ;save Z Register
    mov    r29, r31         ;save Z Register
    rcall    out_nibble         ;Output
    mov    r16, r00         ;Lowest Nibble
    andi    r16, 15             ;Masking
```

```
    rcall    out_nibble        ;Output
    mov    r30, r28        ;save Z Register
    mov    r31, r29        ;save Z Register
    ret

draw_init:                ;Prepare Data Transfer
    sbi    PORTB, LCD_DC        ;Set LCD Command Mode
    cbi    PORTB, LCD_SCE        ;Activate LCD
    ret

draw_finish:                ;LCD Schreibvorgang abschlieÃŸen
    sbi    PORTB, LCD_SCE        ;LCD deaktivieren
    ret

init_display:
    ldi    r16, 0x21        ;Initialize Display
    rcall    lcd_write_cmd
    ldi    r16, 0xD0
    rcall    lcd_write_cmd
    ldi    r16, 0x04
    rcall    lcd_write_cmd
    ldi    r16, 0x13
    rcall    lcd_write_cmd
    ldi    r16, 0x20
    rcall    lcd_write_cmd
    ldi    r16, 0x0C
    rcall    lcd_write_cmd

    ldi    r16, 0x21        ;Set Contrast
    rcall    lcd_write_cmd
    ldi    r16, 0x80 | 0x3C    ;Load Contrast value
    rcall    lcd_write_cmd
    ldi    r16, 0x20
    rcall    lcd_write_cmd

    ret

set_x_position:                ;Set LCD x Position
    ori    r16, 0x80        ;Normalize value
    rcall    lcd_write_cmd        ;Write data
    ret

set_y_position:                ;Set LCD y Position
    ori    r16, 0x40        ;Normaloze value
    rcall    lcd_write_cmd        ;Write data
    ret

spi_out:                ; SPI Software
    cbi    PORTB, LCD_SCLK        ;Set Clock to 0
    ldi    r17, 0x08        ;Set Loopcounter to 8 Bits
spi_out_byte:                ;Output Byte
    sbrc    r16, 0x07        ;Check if Bit 7 not set
```

```
    sbi    PORTB, LCD_SDIN        ;Set Data Line to 1
    sbrs    r16, 0x07        ;Check if Bit 7 is set
    cbi    PORTB, LCD_SDIN        ;Set Data Line to 0
    sbi    PORTB, LCD_SCLK        ;Set Clock to 1
    cbi    PORTB, LCD_SCLK        ;Set Clock to 0
    lsl    r16            ;Shift for the next Bit
    dec    r17            ;Decrement Loop Counter --
    brne    spi_out_byte        ;The same for all other Bits
    ret

lcd_write_cmd:                ;Write LCD Command
    cbi    PORTB, LCD_DC        ;Set LCD Command Mode
    rcall    lcd_out            ;Write Data
    ret

lcd_write_data:                ;Write LCD Data
    sbi    PORTB, LCD_DC        ;Set LCD Data Mode
    rcall    lcd_out            ;Write data
    ret

lcd_out:
    cbi    PORTB, LCD_SCE        ;Activate LCD
    rcall    spi_out            ;Output data
    sbi    PORTB, LCD_SCE        ;De-activate LCD
    ret

out_nibble:                ;Output Nibble
    add    r16, r16
    add    r16, r16        ;Character Addressing
    ldi    r30, LOW(data*2)    ;Load Offset
    ldi    r31, HIGH(data*2)
    add    r30, r16        ;Add Character Position
    ldi    r16, 0x00
    adc    r31, r16
    lpm    r16, Z+         ;Load Character
    rcall spi_out          ;Output Character
    lpm    r16, Z+         ;...
    rcall spi_out
    lpm    r16, Z+
    rcall spi_out
    lpm    r16, Z+
    rcall spi_out
    ret

data:                    ;Character set
    .db    0x00, 0x3E, 0x41, 0x3E    ;0
    .db    0x00, 0x04, 0x02, 0x7F    ;1
    .db    0x00, 0x46, 0x71, 0x4E    ;2
    .db    0x00, 0x49, 0x49, 0x36    ;3
    .db    0x00, 0x0C, 0x0A, 0x7F    ;4
    .db    0x00, 0x4F, 0x49, 0x31    ;5
    .db    0x00, 0x3E, 0x49, 0x31    ;6
```

```
    .db    0x00, 0x01, 0x79, 0x0F    ;7
    .db    0x00, 0x36, 0x49, 0x36    ;8
    .db    0x00, 0x46, 0x49, 0x3E    ;9
    .db    0x00, 0x7F, 0x09, 0x7F    ;A
    .db    0x00, 0x7F, 0x49, 0x36    ;B
    .db    0x00, 0x3E, 0x41, 0x41    ;C
    .db    0x00, 0x7F, 0x41, 0x3E    ;D
    .db    0x00, 0x7F, 0x49, 0x49    ;E
    .db    0x00, 0x7F, 0x09, 0x09    ;F

flash_end:
    rjmp    init            ;jmp to editor
```