MORON'S GUIDE TO THE BUTTERFLY JOYSTICK & LCD v1.2

by RetroDan@GMail.com

TABLE OF CONTENTS:

INTRODUCTION

The joystick is a neat little switch that can be used for input on the Butterfly Demo Boards. The Liquid Crystal Display (LCD) is used as output, to display up to six characters. We are going to use these two devices to create a joystick tester in assembly language that will display which position of the joystick is active at any time.

Then we make a few changes and re-use most of the same code to create a program that will scroll long messages across the small LCD screen.

THE BUTTERFLY JOYSTICK

The Joystick is a combination of five switches in one; one for each of four directions and a centre switch which is activated by pressing down in the middle position. A quick look at the schematics for the Butterfly board shows that the joystick is connected to input pins of both Port B and Port E. Note that the left & right switches are connected to Port E.

```
MIDDLE SWITCH ____-____ PinB,4
    UP SWITCH ____-____ PinB,6
  DOWN SWITCH ____-____ PinB,7
  LEFT SWITCH ____-____ PinE,2
 RIGHT SWITCH ____-____ PinE,3
```

The joystick switches are pulled up by the pull-up resisters and are read as ones when not in use and are shorted to ground and read as zero when pressed. When untouched and open, the input pins float up to the three volts supplied to the Butterfly through an internal resistor tied to Vcc:

```
          +3VDC                          +3VDC
            |                              |
            Z                              Z
            Z                              Z
 OPEN       Z                  CLOSED      Z
SWITCH: _   |                  SWITCH:     |
    .---  ---+---- PINx        .------------+---- PINx
    |                          |
    V                          V
  GROUND                    GROUND
```

Since the joystick is an input device we use the PINx command to read them and not the PORTn form. To catch all the possibilities we might have code that resembles the following:

```
SBIS    PINB,4       ;JOYSTICK PRESS
 RJMP   JOYMID
SBIS    PINB,6       ;JOYSTICK UP
 RJMP   JOYUP
SBIS    PINB,7       ;JOYSTICK DOWN
 RJMP   JOYDOWN
SBIS    PINE,2       ;JOYSTICK LEFT
 RJMP   JOYLEFT
SBIS    PINE,3       ;JOYSTICK RIGHT
 RJMP   JOYRIGHT
```

Note that while three inputs are to Port B pins, the Left and Right switches are connected to Port E pins. The Skip if Bit is Set command (SBIS) test the indicated pin and if it is still set (indicating not pressed) it will skip the RJMP command that follows it. When the associated pin line is pressed (reads zero) the program jumps to the correct routine.

THE LCD SCREEN

The LCD is created by long crystals mounted behind polarized glass. In their normal state they are aligned with the polarized glass and appear transparent so the grey back of the LCD can be seen. When a voltage is applied, the crystals bend enough that light cannot be transmitted through the polarized glass, the associated segment then appears black to the viewer.

```
                          RELAXED              VOLTAGE APPLIED
POLARIZED GLASS: - - - - - - - - - -      - - - - - - - - - - - -
LIQUID CRYSTALS:  | | | | | | | |    ==>  / / / / / / / / / /
     BACKGROUND: -----------------      ---------------------
```

The LCD characters are made from fourteen segments (a to n). To create a character we need to activate the segments that make up the character. For example To create the letter I we might activate segments j and n; and if you look for the letter I in the table below, you see that there is a one in the position for n and j, and the letter C uses segments d,e,f,a:

```
;       mpnd legc jfhb k  a <------> LCD SEGMENTS
 .DW 0b_0011_1001_1001_0001 ;B        -----a-----
 .DW 0b_0001_0100_0100_0001 ;C       | \   |   / |
 .DW 0b_0011_0001_1001_0001 ;D       f  h  j  k  b
 .DW 0b_0001_1110_0100_0001 ;E       |   \ | /   |
 .DW 0b_0000_1110_0100_0001 ;F        --g-- --l--
 .DW 0b_0001_1101_0100_0001 ;G       |   / | \   |
 .DW 0b_0000_1111_0101_0000 ;H       e  p  n  m  c
 .DW 0b_0010_0000_1000_0000 ;I       | /   |   \ |
 .DW 0b_0001_0101_0001_0000 ;J        -----d-----
```

The above is part of a look-up table we use to convert values and ASCII characters to LCD Segments. Later you can modify it to create your own character set.

The LCD segments are memory mapped to twenty memory locations LCDDR0 to LCDDR19 as shown below in a small subroutine that clears all the segments. The Y-Pointer is set to the first memory location LCDDR0, a zero is written to that location and the pointer is increased by one, and then next is cleared until we reach LCDDR19, at which point we stop:

```
;---------------------------;
; CLEAR ALL SEGMENTS ON LCD ;
;---------------------------;
LCD_CLR: LDI YL,LOW(LCDDR0)
         CLR YH
CLRLUPE: ST  Y+,ZERO
         CPI YL,LCDDR18+1
          BRNE CLRLUPE
           RET
```

The LCD Module is quite complex, but as long as we initialize and configure it correctly, all we need to do is convert numbers and ASCII characters to LCD segments, write them to the appropriate memory locations and they will display on the LCD. The LCD Module takes care of things such as duty cycle, frame rates, etc.

THE JOYSTICK TESTER PROGRAM

First we tell the assembler to include the definitions for the ATmega169 MCU on the Butterfly:

```
.INCLUDE "M169DEF.INC"    ;BUTTERFLY DEFS
```

We then define the registers that we will be using. Note that the six registers from R2 to R7 are used as a character buffer for our LCD routine. To display up to six character we load them into these six registers and call our LCD routine, which will do the conversion from numerical or ASCII characters to LCD Segments. CHR6BUF is a pointer to this buffer:

```
      .SET CHR6BUF = 2       ;6 CHAR BUFFER IS [R2,R3,R4,R5,R6,R7]
      .DEF ZERO    = R8
      .DEF T1      = R11
      .DEF T2      = R12
      .DEF A       = R16     ;R16:R31 CAN BE LOADED IMMEDIATE (LDI)
      .DEF AH      = R17
      .DEF B       = R18
      .DEF C       = R19
      .DEF D       = R20
      .DEF I       = R21
      .DEF J       = R22
      .DEF K       = R23
      .DEF N       = R24
```

We start our program at the bottom-of-memory. Set a register called ZERO to zero. Then we set-up a stack at the top-of-memory:

```
      .ORG $0000
             RJMP ON_RESET
      ON_RESET:
             CLR ZERO
             LDI A,HIGH(RAMEND) ;SETUP THE STACK POINTER
             OUT SPH,A          ;AT TOP OF MEMORY AND
             LDI A,LOW(RAMEND)  ;GROW DOWNWARDS
             OUT SPL,A
```

We set Port A and Port E for input. Then we initialize the LCD and make sure it is cleared:

```
             SER   A            ;INIT PORTS B&E FOR INPUT
             OUT   PORTB,A
             OUT   PORTE,A
             RCALL LCD_INIT     ;INITIALIZE LCD
             RCALL LCD_CLR      ;CLEAR LCD SEGMENTS
```

THE MAIN LOOP

The main part of the program polls the joystick switches. If one is pressed it becomes a zero and the appropriate routine is called:

```
MAIN:
LOOP:  SBIS    PINB,4       ;JOYSTICK PRESS
        RJMP   JOYMID
       SBIS    PINB,6       ;JOYSTICK UP
        RJMP   JOYUP
       SBIS    PINB,7       ;JOYSTICK DOWN
        RJMP   JOYDOWN
       SBIS    PINE,2       ;JOYSTICK LEFT
        RJMP   JOYLEFT
       SBIS    PINE,3       ;JOYSTICK RIGHT
        RJMP   JOYRIGHT
```

If no joystick switches are depressed, then the Z-Pointer is set to the message "PRESS" and then jumps to a routine that will display that message on the LCD:

```
NOJOY:  LDI ZL,LOW(MESWAIT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESWAIT*2)
         RJMP SHOWMESS
```

If a joystick switch then the program jumps to one of the following labels, which sets the Z-Pointer to an appropriate message.

```
JOYMID: LDI ZL,LOW(MESMID*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESMID*2)
         RJMP BPMESS

JOYUP:  LDI ZL,LOW(MESUP*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESUP*2)
         RJMP BPMESS

JOYDOWN:LDI ZL,LOW(MESDOWN*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESDOWN*2)
         RJMP BPMESS

JOYLEFT:LDI ZL,LOW(MESLEFT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESLEFT*2)
         RJMP BPMESS

JOYRIGHT:
        LDI ZL,LOW(MESRIGHT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESRIGHT*2)
```

This part of the main routine displays the characters pointed to by the Z-Pointer then calls a delay routine before it loops-back to start over. If a switch is closed then it enters this part of the code from the BPMESS label which also calls a routine to emit a sound from the built-in speaker.

```
BPMESS: RCALL WHIT
SHOWMESS:
        RCALL SHOWBUF         ;SHOW MESSAGE
        RCALL DELAY           ;WAIT
DONE:    RJMP LOOP
```

THE MESSAGES DEFINED

Here we define the six messages for the LCD Display:

```
MESWAIT:  .DB "PRESS "
MESMID:   .DB "CENTRE"
MESUP:    .DB "  UP  "
MESDOWN:  .DB " DOWN "
MESLEFT:  .DB " LEFT "
MESRIGHT: .DB "RIGHT "
```

TRANSFERING DATA TO OUR BUFFER

The SHOWBUF routine copies the characters pointed to by the Z-Pointer into the six character buffer in registers R2 to R7, then calls the routine DISPN that will display them on the LCD Screen:

```
SHOWBUF:LPM  A,Z+
        MOV  R7,A
        LPM  A,Z+
        MOV  R6,A
        LPM  A,Z+
        MOV  R5,A
        LPM  A,Z+
        MOV  R4,A
        LPM  A,Z+
        MOV  R3,A
        LPM  A,Z+
        MOV  R2,A
        PUSH A
        RCALL DISPN
        POP A
         RET
```

THE DISPLAY ROUTINE

The DISPN routine does the conversion from an ASCII character (or a number value) stored in the six character buffer at R2-R7 to LCD segments then stuffs the results into the appropriate LCD Display Registers LCDDR0-LCDDR19.

First it points the X-Pointer to the six character registers R2-R7:

```
DISPN:   LDI  XL,LOW(CHR6BUF)  ;POINTS BUFFER-6
         LDI  XH,HIGH(CHR6BUF)
```

Our character buffer and LCD Display is six characters long, so we set a counter to six. The LCD registers stuff two characters into one byte, so we are going to need a bit mask $F0 to strip away one four bit nybble for us later.

```
LCD_DSP: LDI  N,6         ;SIX CHARS
         LDI  B,$F0       ;BITMASK
```

Next we read in a character from our buffer and check if it is a space, and if so we set it to a blank space (no segments activated):

```
DSPNXT:  LD   A,X+        ;FETCH THE CHAR TO DISP
         CPI  A,' '       ;SPACE?
          BRNE NOSPC      ;SPACE XLATION
         LDI  A,SPACE-LCD_TABLE
```

We check if it is a small letter of ASCII and if so convert it to upper-case:

```
NOSPC:
         CPI  A,'a'       ;CHARACTER XLATION
          BRLO NOSMLET    ;SMALL LETTERS?
         SUBI A,$20       ;FOLD#1 a=>A
```

If it is an upper-case ASCII letter we subtract $37 to make the letter "A" the tenth character in our look-up table (A=10 in Hex). This will make the rest of the upper-case letters line up properly for our translation table:

```
NOSMLET: CPI  A,'A'       ;CAP LETTERS
          BRLO NOBGLET    ;
         SUBI A,$37       ;FOLD#2 A=>10
```

ASCII numbers have $30 subtracted from them so that the character zero is made to equal zero. This aligns the numbers in our look-up table from 0-9.

```
NOBGLET: CPI  A,'0'       ;ASCII NUMBERS
          BRLO NOANUM     ;
         SUBI A,$30       ;FOLD#3 "0"=>0
```

Once we have the ASCII converted to an entry in our look-up table, we multiply it by two and use it as an off-set into our segment look-up table. We multiply it by two because each entry in the table is is a "word" wide, made of two bytes:

```
NOANUM:  LSL  A           ;POINT Z INTO TABLE
         LDI  ZL,LOW(LCD_TABLE*2)
         LDI  ZH,HIGH(LCD_TABLE*2)
         ADD  ZL,A        ;OFFSET INTO
         ADC  ZH,ZERO     ;CHARACTER TABLE
```

THE LCD DATA REGISTERS

The next part is tricky because the LCD Module expects two characters to be stuffed into one byte, but also the segments for these two characters are spread over four different registers which are stored five bytes apart: For example if the two characters are “C” and “I”:

```
;       mpnd legc jfhb k  a <------> LCD SEGMENTS
 .DW 0b_0011_1001_1001_0001 ;B        -----a-----
 .DW 0b_0001_0100_0100_0001 ;C       | \   |   / |
 .DW 0b_0011_0001_1001_0001 ;D       f  h  j  k  b
 .DW 0b_0001_1110_0100_0001 ;E       |   \ | /   |
 .DW 0b_0000_1110_0100_0001 ;F        --g-- --l--
 .DW 0b_0001_1101_0100_0001 ;G       |   / | \   |
 .DW 0b_0000_1111_0101_0000 ;H       e  p  n  m  c
 .DW 0b_0010_0000_1000_0000 ;I       | /   |   \ |
 .DW 0b_0001_0101_0001_0000 ;J        -----d-----
```

```
           high-nybble  low-nybble
LCDDRx:     k - - a      k - - a
LCDDRx+5:   j f h b      j f h b
LCDDRx+10:  l e g c      l e g c
LCDDRx+15:  m p n d      m p n d
```

We see that “C” requires segments d,e,f,a activated and “I” needs n & j so our LCD Data Registers would look like this:

```
  LCD       “C”        “I”
LCDDR0:  0 0 0 1    0 0 0 0
LCDDR4:  0 1 0 0    1 0 0 0
LCDDR9:  0 1 0 0    0 0 0 0
LCDDR14: 0 0 0 1    0 0 1 0
```

BIT MANIPULATION GYMNASTICS

Now that we have converted our ASCII character into LCD segments, the next part does bit manipulation gymnastics because the LCD Module expects our two characters to be stuffed into one byte, but also the segments for these two characters are spread over four different registers which are stored five bytes apart.

```
         LDI  YL,LOW(LCDDR1)-1 ;(=251)POINTS TO
         CLR  YH        ;LCD SEGMENTS
         MOV  A,N       ;USE COUNTER
         DEC  A
         LSR  A         ;AS OFFSET TO
         ADD  YL,A      ;SEGMENTS

         SET
         LDI  I,4
DISPLUP: CPI  YL,LOW(LCDDR8) ;PAST CHECK POINT?
          BRLO NOZINC   ;PAST 2ND READ?
           BRTC NOZINC  ;SHOULD WE INCZ?
         ADIW  ZH:ZL,1  ;INCZ AFTER 2ND READ
         CLT            ;STOP FURTHER INCZ
NOZINC:  LPM   A,Z      ;LOAD SEGMENT DATA
         SBRS  I,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN SEGS
         SBRC  N,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN DIGITS
POTRIP:  AND   A,B      ;MASK NEEDED INFO
         COM   B        ;INVERT MASK
         LD    C,Y      ;READ-IN SEGMENT
         AND   C,B      ;CLEAR A SPOT
         OR    A,C      ;SHOVE-IN NEW
         ST    Y,A      ;WRITE-BACK
         COM   B        ;RE-INVERT MASK
         ADIW  YH:YL,5  ;NEXT SEG
         DEC   I
          BRNE DISPLUP  ;DONE 4 SEGS?
SKPNUM:  COM  B         ;INVERT BIT-MASK
         DEC  N         ;DONE 6 DIGITS?
NOINC:     BRNE DSPNXT
           RET
```

INTIALIZING THE LCD MODULE

Before we can use the LCD Module we must set-up and initialize it. First we set the clock to external by setting the LCD Clock Select (LCDCS) to one, we select a duty cycle of ¼ by setting the LCDMUX1 & LCDMUX0 to one. We tell the Module to use 4 x 25 pins for output by setting the three bits LCDPM2:0 to one in the LCD Control Register "B" (LCDCRB):

```
            LCDCRB: [LCDCS,LCDB2,LCDMUX1,LCDMUX0,_,LCDPM2,LCDPM1,LCDPM0]

      LCD_INIT:
                LDI A,0b1011_0111  ;SET CLOCK, DUTY CYCLE AND # of PINS
                STS LCDCRB, A      ;ENABLE ALL SEGEMENTS
```

To set the update/frame rate to 32Hz we set the clock divider to 8 by setting the by setting the LCDCD2:0 to one in the LCD Frame Rate Register (LCDFRR). Anything slower than 26Hz and the screen will flicker:

```
            LCDFRR: [_,LCDPS2,LCDPS1,LCDPS0,_,LCDCD2,LCDCD1,LCDCD0]

                LDI A,0b0000_0111  ;SET FRAME RATE TO 32Hz
                STS LCDFRR, A
```

For high contrast we select a voltage of 3.3 Volts by setting the LCDCC3:1 to one in the LCD Contrast Control Register (LCDCCR). To save power you could use a lower setting but the characters will be less black:

```
                LDI A,0b0000_1110  ;(1<<LCDCC3) | (1<<LCDCC2) | (1<<LCDCC1)
                STS LCDCCR, A      ;SET THE CONTRAST
```

We enable the LCD Module and tell it to use a power-saving wave form by Setting the LCD Enable (LCDEN) and the LCDAB bits to one:

```
                LDI A,0b1100_0000  ;(1<<LCDEN)  | (1<<LCDAB)
                STS LCDCRA, A      ;ENABLE THE LCD
                 RET
```

THE LOOKUP TABLE

This is our look-up table that will convert our characters into LCD Segments. Each bit of the two-byte word corresponds to one of the LCD character segments:

```
LCD_TABLE:
;       mpnd legc jfhb k  a <------> LCD SEGMENTS
 .DW 0b_0001_0101_0101_0001 ;ZERO
 .DW 0b_0010_0000_1000_0000 ;1
 .DW 0b_0001_1110_0001_0001 ;2
 .DW 0b_0001_1011_0001_0001 ;3
 .DW 0b_0000_1011_0101_0000 ;4
 .DW 0b_0001_1011_0100_0001 ;5
 .DW 0b_0001_1111_0100_0001 ;6
 .DW 0b_0000_0001_0101_0001 ;7
 .DW 0b_0001_1111_0101_0001 ;8
 .DW 0b_0001_1011_0101_0001 ;9
 .DW 0b_0000_1111_0101_0001 ;A
 .DW 0b_0011_1001_1001_0001 ;B        -----a-----
 .DW 0b_0001_0100_0100_0001 ;C       | \   |   / |
 .DW 0b_0011_0001_1001_0001 ;D       f  h  j  k  b
 .DW 0b_0001_1110_0100_0001 ;E       |   \ | /   |
 .DW 0b_0000_1110_0100_0001 ;F        --g-- --l--
 .DW 0b_0001_1101_0100_0001 ;G       |   / | \   |
 .DW 0b_0000_1111_0101_0000 ;H       e  p  n  m  c
 .DW 0b_0010_0000_1000_0000 ;I       | /   |   \ |
 .DW 0b_0001_0101_0001_0000 ;J        -----d-----
 .DW 0b_1000_0110_0100_1000 ;K
 .DW 0b_0001_0100_0100_0000 ;L
 .DW 0b_0000_0101_0111_1000 ;M
 .DW 0b_1000_0101_0111_0000 ;N
 .DW 0b_0001_0101_0101_0001 ;O
 .DW 0b_0000_1110_0101_0001 ;P
 .DW 0b_1001_0101_0101_0001 ;Q
 .DW 0b_1000_1110_0101_0001 ;R
 .DW 0b_0001_1011_0100_0001 ;S
 .DW 0b_0010_0000_1000_0001 ;T
 .DW 0b_0001_0101_0101_0000 ;U
 .DW 0b_1000_0001_0011_0000 ;V
 .DW 0b_1100_0101_0101_0000 ;W
 .DW 0b_1100_0000_0010_1000 ;X
 .DW 0b_0010_0000_0010_1000 ;Y
 .DW 0b_0101_0000_0000_1001 ;Z
 .DW 0b_0001_0100_0100_0001 ;[
 .DW 0b_1000_0000_0010_0000 ;\
 .DW 0b_0001_0001_0001_0001 ;]
 .DW 0b_0000_0000_0110_0000 ;^
 .DW 0b_0001_0000_0000_0000 ;_
 .DW 0b_0000_0000_0000_1000 ;'
 .DW 0b_1110_1010_1010_1000 ;*
 .DW 0b_0010_1010_1000_0000 ;+
 SPACE:
 .DW 0B_0000_0000_0000_0000 ;(SPACE)
 .DW 0b_0000_1010_0000_0000 ;-
 .DW 0b_0100_0000_0000_0000 ;.
 .DW 0b_0100_0000_0000_1000 ;/
 .DW 0b_1000_0000_0000_1000 ;<
 .DW 0b_0001_1010_0000_0000 ;=
 .DW 0b_0100_0000_0010_0000 ;>
```

A SOUND EFFECT AND A PAUSE

The WHIT routine simply makes a small sound effect on the speaker. It repeatedly toggles the speaker pin and calls a pause routine between the toggles, the result is a sound on the speaker. As it does this the counter R0 is decremented so the inter-toggle pause gets smaller and smaller, so the frequency goes up. The result is a sound effect like “WHIT”:

```
WHIT:   CLR  R0
        SBI  DDRB,5         ;SET PORTB-BIT5 FOR OUTPUT
WHLUPE: SBI  PINB,5         ;SET PORTB-BIT5
        RCALL WPAUSE        ;WAIT
        DEC R0
         BRNE WHLUPE        ;LOOP AROUND
          RET
WPAUSE: PUSH R0             ;PAUSE TWEEN PULSES
WPLUPE: DEC R0              ;IE DETERMINS FREQ
         BRNE WPLUPE
        POP R0
         RET
```

The pause routine just goes in loops wasting time:

```
PAUSE:
DELAY: PUSH A
       LDI A,8
DLUPE: DEC R0
        BRNE DLUPE
         DEC R1
          BRNE DLUPE
           DEC A
            BRNE DLUPE
             POP A
              RET
```

FINAL JOYSTICK TESTER PROGRAM LISTING

We put all the pieces together an you get this program ready to run for the Butterfly:

```
;---------------------------------------------;
;              JOYSTICK_TESTER                ;
;                                             ;
; AUTHOR: DANIEL J, DOREY (RETRODAN@GMAIL.COM ;
; 19-OCT-09: CREATED    LAST UPDATE:04-SEP-10 ;
;---------------------------------------------;
.NOLIST
.INCLUDE "M169DEF.INC"    ;BUTTERFLY DEFS
.LIST

;---------------------------------:
; RENAME/DEFINE WORKING REGISTERS ;
;---------------------------------;

.SET CHR6BUF = 2      ;6 CHAR BUFFER IS [R2,R3,R4,R5,R6,R7]

.DEF ZERO    = R8
.DEF T1      = R11
.DEF T2      = R12

.DEF A       = R16    ;R16:R31 CAN BE LOADED IMMEDIATE (LDI)
.DEF AH      = R17
.DEF B       = R18
.DEF C       = R19
.DEF D       = R20
.DEF I       = R21
.DEF J       = R22
.DEF K       = R23
.DEF N       = R24

.ORG $0000
       RJMP ON_RESET

;-----------------;
; INITIALIZATIONS ;
;-----------------;
ON_RESET:
       CLR ZERO
       LDI A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT SPH,A          ;AT TOP OF MEMORY AND
       LDI A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT SPL,A
       SER  A             ;INIT PORTS B&E FOR INPUT
          OUT PORTB,A
          OUT PORTE,A
       RCALL LCD_INIT     ;INITIALIZE LCD
       RCALL LCD_CLR      ;CLEAR LCD SEGMENTS
```

```
;-----------;
; MAIN LOOP ;
;-----------;
MAIN:
LOOP:  SBIS    PINB,4       ;JOYSTICK PRESS
        RJMP   JOYMID
       SBIS    PINB,6       ;JOYSTICK UP
       RJMP   JOYUP
       SBIS    PINB,7       ;JOYSTICK DOWN
        RJMP   JOYDOWN
       SBIS    PINE,2       ;JOYSTICK LEFT
        RJMP   JOYLEFT
       SBIS    PINE,3       ;JOYSTICK RIGHT
        RJMP   JOYRIGHT

NOJOY:  LDI ZL,LOW(MESWAIT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESWAIT*2)
         RJMP SHOWMESS

JOYMID: LDI ZL,LOW(MESMID*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESMID*2)
         RJMP BPMESS

JOYUP:  LDI ZL,LOW(MESUP*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESUP*2)
         RJMP BPMESS

JOYDOWN:LDI ZL,LOW(MESDOWN*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESDOWN*2)
         RJMP BPMESS

JOYLEFT:LDI ZL,LOW(MESLEFT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESLEFT*2)
         RJMP BPMESS

JOYRIGHT:
        LDI ZL,LOW(MESRIGHT*2) ;SET A POINTER TO MESSAGE
        LDI ZH,HIGH(MESRIGHT*2)
BPMESS: RCALL WHIT
SHOWMESS:
        RCALL SHOWBUF         ;SHOW MESSAGE
        RCALL DELAY           ;WAIT

DONE:    RJMP LOOP

MESWAIT:  .DB "PRESS "
MESMID:   .DB "CENTRE"
MESUP:    .DB "  UP  "
MESDOWN:  .DB " DOWN "
MESLEFT:  .DB " LEFT "
MESRIGHT: .DB "RIGHT "
```

```
;=====================[ SUBROUTINES ]=========================

;-----------------------------------------------;
; NO RESTORE WHIT ROUTINE, USES THE R0 REGISTER ;
;-----------------------------------------------;
WHIT:   CLR  R0
        SBI  DDRB,5         ;SET PORTB-BIT5 FOR OUTPUT
WHLUPE: SBI  PINB,5         ;SET PORTB-BIT5
        RCALL WPAUSE        ;WAIT
        DEC R0
         BRNE WHLUPE        ;LOOP AROUND
          RET
WPAUSE: PUSH R0             ;PAUSE TWEEN PULSES
WPLUPE: DEC R0              ;IE DETERMINS FREQ
         BRNE WPLUPE
        POP R0
         RET

;-------------------------------;
; COPIES TEXT TO DISPLAY BUFFER ;
; MUST LOAD (Z) FIRST           ;
;-------------------------------;
SHOWBUF:LPM  A,Z+
        MOV  R7,A
        LPM  A,Z+
        MOV  R6,A
        LPM  A,Z+
        MOV  R5,A
        LPM  A,Z+
        MOV  R4,A
        LPM  A,Z+
        MOV  R3,A
        LPM  A,Z+
        MOV  R2,A
        PUSH A
        RCALL DISPN
        POP A
         RET

;-----------------------------------------------;
; DISPN - DISPLAY THE NUMBER IN R7:R2 REGISTERS ;
; NOTE CHR6BUF MUST BE POINTING 6 CHAR BUFFER   ;
; APR/06 VERSION II WITH ASCII XLATION          ;
;-----------------------------------------------;
DISPN:   LDI  XL,LOW(CHR6BUF)  ;POINTS BUFFER-6
         LDI  XH,HIGH(CHR6BUF)

;-------------------------;
; ENTER HERE IF XH:XL SET ;
;-------------------------;
LCD_DSP: LDI  N,6       ;SIX CHARS
         LDI  B,$F0     ;BITMASK

DSPNXT:  LD   A,X+      ;FETCH THE CHAR TO DISP
         CPI  A,' '     ;SPACE?
          BRNE NOSPC    ;SPACE XLATION
         LDI  A,SPACE-LCD_TABLE
NOSPC:
         CPI  A,'a'      ;CHARACTER XLATION
          BRLO NOSMLET  ;SMALL LETTERS?
         SUBI A,$20     ;FOLD#1 a=>A

NOSMLET: CPI  A,'A'-1   ;CAP LETTERS
          BRLO NOBGLET  ;
         SUBI A,$37     ;FOLD#2 A=>10

NOBGLET: CPI  A,'0'-1   ;ASCII NUMBERS
          BRLO NOANUM   ;
         SUBI A,$30     ;FOLD#3 "0"=>0
```

```
NOANUM:  LSL  A         ;POINT Z INTO TABLE
         LDI  ZL,LOW(LCD_TABLE*2)
         LDI  ZH,HIGH(LCD_TABLE*2)
         ADD  ZL,A      ;OFFSET INTO
         ADC  ZH,ZERO   ;CHARACTER TABLE

         LDI  YL,LOW(LCDDR1)-1 ;(=251)POINTS TO
         CLR  YH        ;LCD SEGMENTS
         MOV  A,N       ;USE COUNTER
         DEC  A
         LSR  A         ;AS OFFSET TO
         ADD  YL,A      ;SEGMENTS

         SET
         LDI  I,4
DISPLUP: CPI  YL,LOW(LCDDR8) ;PAST CHECK POINT?
          BRLO NOZINC   ;PAST 2ND READ?
           BRTC NOZINC  ;SHOULD WE INCZ?
         ADIW  ZH:ZL,1  ;INCZ AFTER 2ND READ
         CLT            ;STOP FURTHER INCZ
NOZINC:  LPM   A,Z      ;LOAD SEGMENT DATA
         SBRS  I,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN SEGS
         SBRC  N,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN DIGITS
POTRIP:  AND   A,B      ;MASK NEEDED INFO
         COM   B        ;INVERT MASK
         LD    C,Y      ;READ-IN SEGMENT
         AND   C,B      ;CLEAR A SPOT
         OR    A,C      ;SHOVE-IN NEW
         ST    Y,A      ;WRITE-BACK
         COM   B        ;RE-INVERT MASK
         ADIW  YH:YL,5  ;NEXT SEG
         DEC   I
          BRNE DISPLUP  ;DONE 4 SEGS?
SKPNUM:  COM  B         ;INVERT BIT-MASK
         DEC  N         ;DONE 6 DIGITS?
NOINC:    BRNE DSPNXT
           RET

;---------------------------;
; CLEAR ALL SEGMENTS ON LCD ;
;---------------------------;
LCD_CLR: LDI YL,LOW(LCDDR0)
         CLR YH
CLRLUPE: ST  Y+,ZERO
         CPI YL,LCDDR18+1
          BRNE CLRLUPE
           RET

;-------------------------------;
; INITIALIZE LCD DISP REGISTERS ;
;-------------------------------;
LCD_INIT: PUSH A
          LDI A,0b1011_0111  ;SET CLOCK, DUTY CYCLE, # PINS
          STS LCDCRB, A      ;ENABLE ALL SEGEMENTS
          LDI A,0b0000_0111  ;SET FRAME RATE TO 32Hz
          STS LCDFRR, A
          LDI A,0b0000_1110  ;SET CONTRAST VOLTAGE TO 3.3 VOLTS
          STS LCDCCR, A
          LDI A,0b1100_0000  ;ENABLE LCD WITH POWER SAVE WAVEFORM
          STS LCDCRA, A
          POP A
           RET
```

```
PAUSE:
DELAY: PUSH A
       LDI A,8
DLUPE: DEC R0
        BRNE DLUPE
         DEC R1
          BRNE DLUPE
           DEC A
            BRNE DLUPE
             POP A
              RET

;------------------------------------------------------------;
;    RETRO DAN'S IMPROVED LCD CHARACTER TABLE V1.2          ;
;
;------------------------------------------------------------;
LCD_TABLE:
;       mpnd legc jfhb k  a <------> LCD SEGMENTS
 .DW 0b_0001_0101_0101_0001 ;ZERO
 .DW 0b_0010_0000_1000_0000 ;1
 .DW 0b_0001_1110_0001_0001 ;2
 .DW 0b_0001_1011_0001_0001 ;3
 .DW 0b_0000_1011_0101_0000 ;4
 .DW 0b_0001_1011_0100_0001 ;5
 .DW 0b_0001_1111_0100_0001 ;6
 .DW 0b_0000_0001_0101_0001 ;7
 .DW 0b_0001_1111_0101_0001 ;8
 .DW 0b_0001_1011_0101_0001 ;9
 .DW 0b_0000_1111_0101_0001 ;A
 .DW 0b_0011_1001_1001_0001 ;B        -----a-----
 .DW 0b_0001_0100_0100_0001 ;C       | \   |   / |
 .DW 0b_0011_0001_1001_0001 ;D       f  h  j  k  b
 .DW 0b_0001_1110_0100_0001 ;E       |   \ | /   |
 .DW 0b_0000_1110_0100_0001 ;F        --g-- --l--
 .DW 0b_0001_1101_0100_0001 ;G       |   / | \   |
 .DW 0b_0000_1111_0101_0000 ;H       e  p  n  m  c
 .DW 0b_0010_0000_1000_0000 ;I       | /   |   \ |
 .DW 0b_0001_0101_0001_0000 ;J        -----d-----
 .DW 0b_1000_0110_0100_1000 ;K
 .DW 0b_0001_0100_0100_0000 ;L
 .DW 0b_0000_0101_0111_1000 ;M
 .DW 0b_1000_0101_0111_0000 ;N
 .DW 0b_0001_0101_0101_0001 ;O
 .DW 0b_0000_1110_0101_0001 ;P
 .DW 0b_1001_0101_0101_0001 ;Q
 .DW 0b_1000_1110_0101_0001 ;R
 .DW 0b_0001_1011_0100_0001 ;S
 .DW 0b_0010_0000_1000_0001 ;T
 .DW 0b_0001_0101_0101_0000 ;U
 .DW 0b_1000_0001_0011_0000 ;V
 .DW 0b_1100_0101_0101_0000 ;W
 .DW 0b_1100_0000_0010_1000 ;X
 .DW 0b_0010_0000_0010_1000 ;Y
 .DW 0b_0101_0000_0000_1001 ;Z
 .DW 0b_0001_0100_0100_0001 ;[
 .DW 0b_1000_0000_0010_0000 ;\
 .DW 0b_0001_0001_0001_0001 ;]
 .DW 0b_0000_0000_0110_0000 ;^
 .DW 0b_0001_0000_0000_0000 ;_
 .DW 0b_0000_0000_0000_1000 ;'
 .DW 0b_1110_1010_1010_1000 ;*
 .DW 0b_0010_1010_1000_0000 ;+
 SPACE:
 .DW 0B_0000_0000_0000_0000 ;(SPACE)
 .DW 0b_0000_1010_0000_0000 ;-
 .DW 0b_0100_0000_0000_0000 ;.
 .DW 0b_0100_0000_0000_1000 ;/
 .DW 0b_1000_0000_0000_1000 ;<
 .DW 0b_0001_1010_0000_0000 ;=
 .DW 0b_0100_0000_0010_0000 ;>
```

AN LCD SCROLLING PROGRAM

In the last program we used to the LCD to tell which switch of the Butterfly joystick was depressed. The messages were six or less characters long. To display a longer message on the LCD we scroll it across the screen from right to left.

First we setup a speed constant that is used in the pause/delay routine which is called inside our scrolling routine.

```
.SET SPEED = 6      ;USED TO SET SCROLL SPEED

PAUSE:
DELAY: PUSH A
       LDI A,SPEED
DLUPE: DEC R0
        BRNE DLUPE
         DEC R1
          BRNE DLUPE
           DEC A
            BRNE DLUPE
             POP A
              RET
```

The main loop of the program simply points to our message, then calls a scroll routine in an endless loop:

```
MAIN:
LOOP:  LDI YL,LOW(MESSAGE*2) ;SET A POINTER TO MESSAGE
       LDI YH,HIGH(MESSAGE*2)
       RCALL SCROLL          ;SCROLL MESSAGE
DONE:    RJMP LOOP
```

Our message is much longer than six characters and ends with a period "." and it is inside quotes. We use blank spaces so the message scrolls onto and completely off the screen each time.

```
MESSAGE: .DB "     HELLO TO THE WORLD FROM INSIDE THE AVR169 BUTTERFLY USING
ASSEMBLY LANGUAGE     ."
```

The scroll routine copies our pointer for the message to the Z-Pointer for the SHOWBUF routine and after it is displayed on the LCD the Y-pointer is incremented and we do this over and over until we hit the period:

```
SCROLL: MOVW Z,Y                  ;MOVE FROM 1ST POINTER TO 2ND
        PUSH YL                   ;SAVE 1ST POINTER
        PUSH YH
        RCALL SHOWBUF             ;DISPLAY WHAT Z POINTS AT
        RCALL DELAY               ;WAIT
        POP YH                    ;RESTORE 1ST POINTER
        POP YL
        ADIW YH:YL,1              ;INCREMENT POINTER
        CPI A,'.'                 ;STOP AT PERIOD '.'
         BRNE SCROLL
          RET
```

THE LCD SCROLLING PROGRAM LISTING

```
;------------------------------------------;
;         HELLO WORLD #3 (SCROLLING)       ;
;         =============================    ;
;                                          ;
; DANIEL J, DOREY AKA RETRODAN @GMAIL.COM  ;
; 05-OCT-09: CREATED LAST UPDATE:04-SEP-10 ;
;                                          ;
; SCROLLS LONG MESSAGES ACROSS LCD SCREEN  ;
; MESSAGE TERMINATED WITH A PERIOD (.)     ;
;------------------------------------------;

.INCLUDE "M169DEF.INC"    ;BUTTERFLY DEFS

;---------------------------------:
; RENAME/DEFINE WORKING REGISTERS ;
;---------------------------------;

.SET SPEED   = 6      ;USED TO SET SCROLL SPEED SEE PAUSE ROUTINE
.SET CHR6BUF = 2      ;6 CHAR BUFFER IS [R2,R3,R4,R5,R6,R7]

.DEF ZERO    = R8
.DEF T1      = R11
.DEF T2      = R12

.DEF A       = R16    ;R16:R31 CAN BE LOADED IMMEDIATE (LDI)
.DEF AH      = R17
.DEF B       = R18
.DEF C       = R19
.DEF D       = R20
.DEF I       = R21
.DEF J       = R22
.DEF K       = R23
.DEF N       = R24

.ORG $0000
       RJMP RESET

;-----------------;
; INITIALIZATIONS ;
;-----------------;
RESET: CLR ZERO
       LDI A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT SPH,A          ;AT TOP OF MEMORY AND
       LDI A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT SPL,A
       RCALL LCD_INIT     ;INITIALIZE LCD
       RCALL LCD_CLR      ;CLEAR LCD SEGMENTS

;-----------;
; MAIN LOOP ;
;-----------;
MAIN:
LOOP:  LDI YL,LOW(MESSAGE*2) ;SET A POINTER TO MESSAGE
       LDI YH,HIGH(MESSAGE*2)
       RCALL SCROLL          ;SCROLL MESSAGE
DONE:    RJMP LOOP
MESSAGE: .DB "    HELLO TO THE WORLD FROM INSIDE THE AVR169 BUTTERFLY USING
ASSEMBLY LANGUAGE     ."
```

```
;---------------------;
; SCROLL MESSAGE LOOP ;
;---------------------;
SCROLL: MOVW Z,Y                  ;MOVE FROM 1ST POINTER TO 2ND
        PUSH YL                   ;SAVE 1ST POINTER
        PUSH YH
        RCALL SHOWBUF             ;DISPLAY WHAT Z POINTS AT
        RCALL DELAY               ;WAIT
        POP YH                    ;RESTORE 1ST POINTER
        POP YL
        ADIW YH:YL,1              ;INCREMENT POINTER
        CPI A,'.'                 ;STOP AT PERIOD '.'
         BRNE SCROLL
          RET
;-------------------------------;
; COPIES TEXT TO DISPLAY BUFFER ;
; MUST LOAD (Z) FIRST           ;
;-------------------------------;
SHOWBUF:LPM  A,Z+
        MOV  R7,A
        LPM  A,Z+
        MOV  R6,A
        LPM  A,Z+
        MOV  R5,A
        LPM  A,Z+
        MOV  R4,A
        LPM  A,Z+
        MOV  R3,A
        LPM  A,Z+
        MOV  R2,A
        PUSH A
        RCALL DISPN
        POP A
         RET
;-----------------------------------------------;
; DISPN - DISPLAY THE NUMBER IN R7:R2 REGISTERS ;
; NOTE CHR6BUF MUST BE POINTING 6 CHAR BUFFER   ;
;-----------------------------------------------;
DISPN:   LDI  XL,LOW(CHR6BUF)  ;POINTS BUFFER-6
         LDI  XH,HIGH(CHR6BUF)
;-------------------------;
; ENTER HERE IF XH:XL SET ;
;-------------------------;
LCD_DSP: LDI  N,6       ;SIX CHARS
         LDI  B,$F0     ;BITMASK

DSPNXT:  LD   A,X+      ;FETCH THE CHAR TO DISP
         CPI  A,' '     ;SPACE?
          BRNE NOSPC    ;SPACE XLATION
         LDI  A,SPACE-LCD_TABLE
NOSPC:
         CPI  A,'a'     ;CHARACTER XLATION
          BRLO NOSMLET  ;SMALL LETTERS?
         SUBI A,$20     ;FOLD#1 a=>A

NOSMLET: CPI  A,'A'    ;CAP LETTERS
          BRLO NOBGLET  ;
         SUBI A,$37     ;FOLD#2 A=>10

NOBGLET: CPI  A,'0'    ;ASCII NUMBERS
          BRLO NOANUM   ;
         SUBI A,$30     ;FOLD#3 "0"=>0

NOANUM:  LSL  A         ;POINT Z INTO TABLE
         LDI  ZL,LOW(LCD_TABLE*2)
         LDI  ZH,HIGH(LCD_TABLE*2)
         ADD  ZL,A      ;OFFSET INTO
         ADC  ZH,ZERO   ;CHARACTER TABLE

         LDI  YL,LOW(LCDDR1)-1 ;(=251)POINTS TO
         CLR  YH        ;LCD SEGMENTS
         MOV  A,N       ;USE COUNTER
         DEC  A
         LSR  A         ;AS OFFSET TO
         ADD  YL,A      ;SEGMENTS
         SET
         LDI  I,4
DISPLUP: CPI  YL,LOW(LCDDR8) ;PAST CHECK POINT?
          BRLO NOZINC   ;PAST 2ND READ?
           BRTC NOZINC  ;SHOULD WE INCZ?
         ADIW  ZH:ZL,1  ;INCZ AFTER 2ND READ
         CLT            ;STOP FURTHER INCZ
NOZINC:  LPM   A,Z      ;LOAD SEGMENT DATA
         SBRS  I,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN SEGS
         SBRC  N,0      ;USE BIT0
         SWAP  A        ;SWAP ON EVEN DIGITS
POTRIP:  AND   A,B      ;MASK NEEDED INFO
         COM   B        ;INVERT MASK
         LD    C,Y      ;READ-IN SEGMENT
         AND   C,B      ;CLEAR A SPOT
         OR    A,C      ;SHOVE-IN NEW
         ST    Y,A      ;WRITE-BACK
         COM   B        ;RE-INVERT MASK
         ADIW  YH:YL,5  ;NEXT SEG
         DEC   I
          BRNE DISPLUP  ;DONE 4 SEGS?
SKPNUM:  COM  B         ;INVERT BIT-MASK
         DEC  N         ;DONE 6 DIGITS?
NOINC:    BRNE DSPNXT
           RET

;---------------------------;
; CLEAR ALL SEGMENTS ON LCD ;
;---------------------------;
LCD_CLR: LDI YL,LOW(LCDDR0)
         CLR YH
CLRLUPE: ST  Y+,ZERO
         CPI YL,LCDDR18+1
          BRNE CLRLUPE
           RET

;-------------------------------;
; INITIALIZE LCD DISP REGISTERS ;
;-------------------------------;
LCD_INIT: PUSH A
          LDI A,0b1011_0111  ;SET CLOCK, DUTY CYCLE AND # PINS
          STS LCDCRB, A      ;ENABLE ALL SEGEMENTS
          LDI A,0b0000_0111  ;SET FRAME RATE TO 32Hz
          STS LCDFRR, A      ;SET PRESCALER TO 32KHz
          LDI A,0b0000_1110  ;SET THE CONTRAST TO 3.3 VOLTS
          STS LCDCCR, A      ;SET THE CONTRAST
          LDI A,0b1100_0000  ;ENABLE LCD POWER-SAVE WAVE FROM
          STS LCDCRA, A      ;ENABLE THE LCD
          POP A
           RET

;---------------------;
; PAUSE/DELAY ROUTINE ;
;---------------------;
PAUSE:
DELAY: PUSH A
       LDI A,SPEED
DLUPE: DEC R0
        BRNE DLUPE
         DEC R1
          BRNE DLUPE
           DEC A
            BRNE DLUPE
             POP A
              RET
```

```
;------------------------------------------------------------;
;    RETRO DAN'S IMPROVED LCD CHARACTER TABLE V1.2           ;
;
;    ALTERATIONS FROM ORIGINAL CHAR SET FOUND IN APP NOTES   ;
; 1. CONVERTED TO BINARY FROM HEX FOR LEGIBITIY              ;
; 2. REPLACED SOME CHARS                                     ;
; 3. PLACED ALPHABET RIGHT AFTER NUMBERS EASES TABLE         ;
;    LOOKUPS WHEN USING HEX: 10 OVERFLOWS INTO A, 11=>B ETC ;
; 4. ZERO MOVED TO FIRST ENTRY TO EASE TABLE LOOKUPS         ;
;------------------------------------------------------------;
LCD_TABLE:
;    --mpndlegcjfhbk--a <------> LCD SEGMENTS
 .DW 0b0001010101010001 ;ZERO
 .DW 0b0010000010000000 ;1
 .DW 0b0001111000010001 ;2
 .DW 0b0001101100010001 ;3
 .DW 0b0000101101010000 ;4
 .DW 0b0001101101000001 ;5
 .DW 0b0001111101000001 ;6
 .DW 0b0000000101010001 ;7
 .DW 0b0001111101010001 ;8
 .DW 0b0001101101010001 ;9
 .DW 0b0000111101010001 ;A
 .DW 0b0011100110010001 ;B        -----a-----
 .DW 0b0001010001000001 ;C       | \   |   / |
 .DW 0b0011000110010001 ;D       f  h  j  k  b
 .DW 0b0001111001000001 ;E       |   \ | /   |
 .DW 0b0000111001000001 ;F        --g-- --l--
 .DW 0b0001110101000001 ;G       |   / | \   |
 .DW 0b0000111101010000 ;H       e  p  n  m  c
 .DW 0b0010000010000000 ;I       | /   |   \ |
 .DW 0b0001010100010000 ;J        -----d-----
 .DW 0b1000011001001000 ;K
 .DW 0b0001010001000000 ;L
 .DW 0b0000010101111000 ;M
 .DW 0b1000010101110000 ;N
 .DW 0b0001010101010001 ;0
 .DW 0b0000111001010001 ;P
 .DW 0b1001010101010001 ;Q
 .DW 0b1000111001010001 ;R
 .DW 0b0001101101000001 ;S
 .DW 0b0010000010000001 ;T
 .DW 0b0001010101010000 ;U
 .DW 0b1000000100110000 ;V
 .DW 0b1100010101010000 ;W
 .DW 0b1100000000101000 ;X
 .DW 0b0010000000101000 ;Y
 .DW 0b0101000000001001 ;Z
 .DW 0b0001010001000001 ;[
 .DW 0b1000000000100000 ;\
 .DW 0b0001000100010001 ;]
 .DW 0b0000000001100000 ;^
 .DW 0b0001000000000000 ;_
 .DW 0b0000000000001000 ;'
 .DW 0b1110101010101000 ;*
 .DW 0b0010101010000000 ;+
 SPACE:.DW 0 ;(SPACE)   ;
 .DW 0b0000101000000000 ;-
 .DW 0b0100000000000000 ;.
 .DW 0b0100000000001000 ;/
 .DW 0b1000000000001000 ;<
 .DW 0b0001101000000000 ;=
 .DW 0b0100000000100000 ;>
```