Aviles, Jean-Ralph
EEL3744
Section 1539
November 3rd, 2015
Lab 6

Questions

1. If you were working on another microcontroller and you wanted to add an 8-bit LCD to it, what is the minimum amount of signals required from the microcontroller to get it working?

   According to the manual for the LCD we need 8 data pins, an enable, and a RS to get the LCD working.

2. In this lab our reference range is ideally from 0V to 5V. If the range was 0 to 2.0625V (a possible internal reference) and 12-bit unsigned mode was used, what is the resolution (volts/bit) and what is the digital value for a voltage of 0.37 V.

   12 bits yields 4096 possible values, so with a range of 2.0625V our resolution is

$$\frac{2.0625}{12} = 0.17\frac{V}{bit} \tag{1}$$

   The digital value for 0.37V would be...

$$\frac{0.37V}{2.0625V} \times 4096 = 734 \tag{2}$$

3. Assume you wanted a voltage reference range from 1 V to 3 V, with a signed 12-bit ADC. What are the voltages if the ADC output is 0xA42 and 0xD37?

$$0xA42 = 2626.V = \frac{2626}{4096} \times 2V + 1V = 2.28V \tag{3}$$

$$0xD37 = 3383.V = \frac{3383}{4096} \times 2V + 1V = 2.65V \tag{4}$$

4. What is the difference in conversion ranges between 12- bit unsigned and signed conversion modes? List both ranges.
   $Unsigned : 0 - 2^{12} - 1 = 0 - 4095.Signed - 2^{-11} - 2^{11} - 1$

Problems Encountered

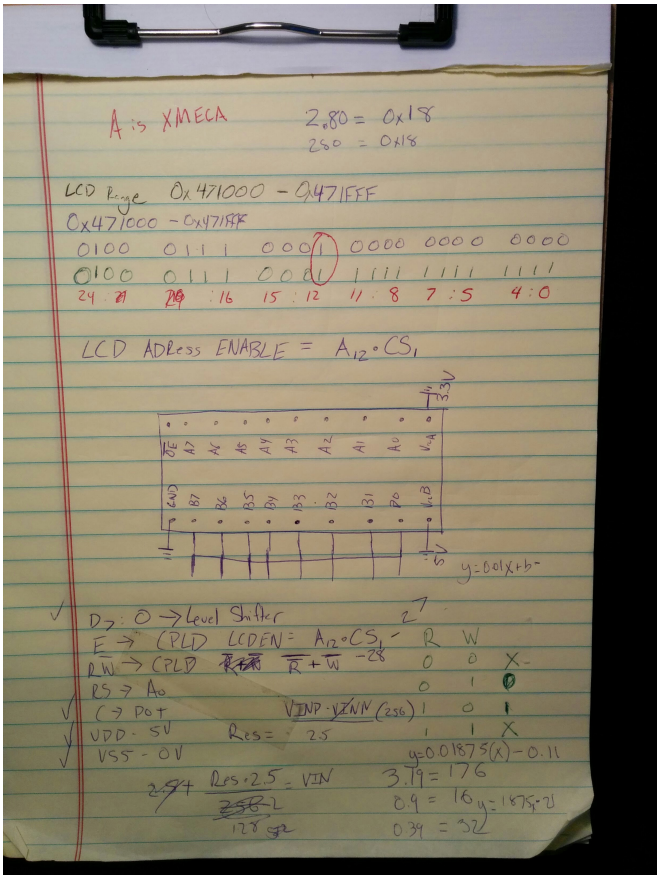
1. I could not poll the buzy flag on the LCD screen. I ended up just using a flat delay because of this. Check out lcd_busy() in lcd.h.

Future Work/Applications

Writing to the LCD screen can give us –the user– easier access to what is going on with the microcontroller and allows us to write more user friendly programs in the future.

Appendix

Scratch Work



Pseudocode/Flowcharts

Lab6_lcd_keypad.ps

```
LAST_KEY = '\0'
LAST_SERIAL = '\0'

def main():
  init()
  switch (LAST_KEY):
    case '0':
```

```
    case '1':
      lcd_write("JR  Aviles")
      sleep()
    case '2':
    case '3':
      lcd_clear()
      sleep()
    case '4':
    case '5':
      toggle_lcd()
      sleep()
    case '6':
    case '7':
      lcd_clear()
      lcd_write("May the Schwartz\nBe with you!")
      sleep()
    case '*':
    case '#':
      while LAST_SERIAL == '*' or LAST_SERIAL == '#':
        x = str(convert(adc_value()))
        lcd_write("%s V (%s)" % (x, hex(x)))
        delay(500)
    default:
      sleep()

def Serial_RX_ISR():
  LAST_SERIAL = serial_data

def Keydown_ISR():
  LAST_KEY = read_key()
```

Lab6_lcd_name.ps

```
def main():
  init()
  while True:
    clear_lcd()
    lcd_write("Jean-Ralph")
    delay(500)
```

Lab6_lcd_voltage.ps

```
def main():
```

```
  init()
  while True:
    lcd_clear()
    x = str(convert(read_adc()))
    lcd_write("%s V (%s)" % (x, hex(x)))
    delay(500)
```

adc.h

```
def init():
  ADCB_CTRLA = 0x01
  PORTA_DIRCLR = 0xFF
  PORTB_DIRCLR = 0x08
  ADCB_CTRLB = 0x0C
  ADCB_REFCLR = 0x30
  ADCB_PRESCALER = 0x07
  ADCB_CH0_CTRL = 0x81
  ADCB_CH0_MUXCTRL = 0x20

def read_adc():
  ADCB_CTRLA |= 0x4
  ADCB_CH0_CTRL = 0x81
  while ADCB_CH0_INTFLAGS & 0x01 is 0:
    pass
  ADCB_CH0_INTFLAGS = 0x01
  return ADCB_CH0_RES

def convert(reading):
  return 1.875 * reading - 20
```

helpers.h

```
def delay_ms(ms):
  for i in range(0, ms):
    for j in range(0, 390):
      pass

def uint_string(x):
  return str(x)

def init():
  PORTH_DIR = 0b110111
  PORTH_OUTSET = 0b11011
```

```
  PORTJ_DIR = 0xFF
  PORTK_DIR = 0xFF
  EBI_CTRL = 0x01
  EBI_CS0_CTRLA = 0b0010101
  EBI_CS0_BASEADDR = 0x8000
  EBI_CS1_CTRLA = 0b100001
  EBI_CS1_BASEADDR = 0x470000
```

keypad.h

```
KEYTAB = [
  ['1', '2', '3', 'A'],
  ['4', '5', '6', 'B'],
  ['7', '8', '9', 'C'],
  ['*', '0', '#', 'D']
]

def init():
  PORTF_DIR = 0xF0;
  PORTCFG_MPCMASK = 0x0F;
  PORTF_PIN0CTRL = 0x11;
  PORTF_INTCTRL = 0x03;
  PORTF_INT0MASK = 0x0F;
  PORTF_OUT = 0xF0;

char read_keypad(void) {
  for i in range(0, 4):
    PORTF_OUT = (0b0001 << i) << 4
    for j in range(0, 4):
      if PORTF_IN & (0b0001 << j) is not 0:
          return KEYTAB[i][j];
  return 'F';
}

ISR(PORTF_INT0_vect) {
  delay_ms(1)
  tmp = read_keypad();
  if tmp is not 'F':
    last_key = tmp;
  PORTF_INTFLAGS |= 0x01;
}
```

lcd.h

```
cursor_pos = 0

def init():
  delay(40)
  lcd_write_cmd(0b111100)
  delay(40)
  lcd_write_cmd(0b111100)
  lcd_cmd(0b1111)
  lcd_on = True
  lcd_cmd(0b110)
  lcd_clear()

def lcd_write_c(c):
  if c is '\n':
    lcd_next_line()
  else:
    wait_lcd()
    mem_write(LCD_DATA, c)
    if cursor_pos is 16:
      cursor_pos = 40
      lcd_cmd(0x80 | cursor_pos)
    elif cursor_pos is 56
      cursor_pos = 0
      lcd_cmd(0x80)

def lcd_write_s(x):
  for c in x:
    lcd_write_c(c)

def lcd_cmd(cmd):
  wait_lcd()
  mem_write(LCD_CMD, cmd)

def lcd_clear():
  lcd_cmd(1)
  cursor_pos = 0

def lcd_busy():
  return mem_read(LCD_CMD) & 0x80 is not 0

def wait_lcd():
  while lcd_busy:
    pass
```

```
def lcd_reset_cursor():
  lcd_cmd(0b10)

def lcd_next_line():
  if cursor_pos < 40:
    cursor_pos = 40
    lcd_cmd(0x80 | cursor_pos)
  else:
    cursor_pos = 0
    lcd_cmd(0x80)

def lcd_hpos():
  if cursor_pos < 16:
    return cursor_pos
  else:
    return cursor_pos - 40

def lcd_vpos():
  if cursor_pos < 16:
    return 0
  return 1

def set_pos(h, v):
  if v is 0:
    cursor_pos = h
  else:
    cursor_pos = h + 40
  lcd_cmd(0x80 | cursor_pos)

def toggle_lcd():
  lcd_on ^= 1
  lcd_cmd(0b1111 ^ (lcd_on << 2))

def lcd_move_cursor(dir):
  switch (dir):
    case LEFT:
      if lcd_vpos() is 1 && lcd_hpos() is 0:
        set_pos(15, 0)
      elif lcd_hpos() != 0:
        set_pos(lcd_hpos() - 1, lcd_vpos())
    case RIGHT:
      if lcd_vpos() is 0 && lcd_hpos() is 15:
        set_pos(0, 1)
      elif lcd_hpos() != 15:
```

```
        set_pos(lcd_hpos() + 1, lcd_vpos())
    case UP:
      if lcd_vpos() is 1:
        set_pos(lcd_hpos(), 0)
    case DOWN:
      if lcd_vpos() is 0:
        set_pos(lcd_hpos(), 1)
```

serial.h

```
LAST_SERIAL = '\0'
BSCALE = -7
BSEL = 1539

def init():
  PORTD_DIRSET = 0x08
  PORTD_OUTSET = 0x08
  PORTD_DIRCLR = 0x04
  PORTQ_DIRSET = 0x0A
  PORTQ_OUTCLR = 0x0A
  USARTD0_CTRLB = 0x18
  USARTD0_CTRLC = 0x03
  USARTD0_BAUDCTRLA = BSEL & 0xFF
  USARTD0_BAUDCTRLB = (((BSCALE << 4) & 0xF0) | ((BSEL >> 8) & 0x0F))
  USARTD0_CTRLA = 0x30

def serial_send(data):
  while USARTD0_STATUS & 0x20 is 0:
    pass
  USARTD0_DATA = data

def serial_send_s(string):
  for c in string:
    serial_send(c)

SERIAL_RX_ISR():
  LAST_SERIAL = USARTD0_DATA
```

Programs

Part A

LCD Name

```
/*
 * Lab6_lcd_name.c
 * Writes My Name to the LCD
 * Created: 10/27/2015 12:24:48 PM
 *  Author: Jean-Ralph Aviles
 */

#include "helpers.h"
#include "lcd.h"

int main() {
  init();
  init_lcd();
  while(1) {
    lcd_clear();
    lcd_write_s("Jean-Ralph\nPart A");
    delay_ms(1000);
  }
  return 0;
}
```

Part B

LCD Voltage

```
/*
* Lab6_lcd_voltage.c
* LCD Voltage Reader
* Created: 10/27/2015 12:24:48 PM
*  Author: Jean-Ralph Aviles
*/

#include "helpers.h"
#include "ebi_driver.h"
#include "lcd.h"
#include "adc.h"

int main(void) {
  init();
```

```
  init_adc();
  init_lcd();
  char val[10];
  while (1) {
    int x = convert(read_adc());
    __far_mem_write(IOP_START, x);
    uint_string(x, val, 10, 10);
    int len = 0;
    for (len = 0; val[len] != '\0'; ++len);
    int i;
    for (i = 0; i < len - 2; ++i) {
      lcd_write_c(val[i++]);
    }
    lcd_write_c('.');
    lcd_write_s(val + i);
    lcd_write_s(" V ");
    uint_string(x, val, 16, 10);
    lcd_write_s("(0x");
    lcd_write_s(val);
    lcd_write_s(")\nPart B");
    delay_ms(500);
    lcd_clear();
  }
  return 0;
}
```

Part C

LCD Keypad

```
/*
* Lab6_lcd_keypad.c
* LCD Keypad Control
* Created: 10/27/2015 12:24:48 PM
*  Author: Jean-Ralph Aviles
*/

#include "adc.h"
#include "helpers.h"
#include "keypad.h"
#include "lcd.h"
#include "serial.h"

extern volatile char last_key;
```

```
extern volatile char last_serial;

int main(void) {
  init();
  init_adc();
  init_lcd();
  init_keypad();
  init_usart();
  sleep_mode_set(SLEEP_MODE_IDLE);
  enable_sleep();
  intmask_set(0x04);
  enable_int();
  while (1) {
    switch (last_key) {
      case '0':
      case '1':
        lcd_write_s("JR\nAviles");
        sleep();
        break;
      case '2':
      case '3':
        lcd_clear();
        sleep();
        break;
      case '4':
      case '5':
        toggle_lcd();
        sleep();
        break;
      case '6':
      case '7':
        lcd_clear();
        lcd_write_s("May the Schwartzbe with you!");
        sleep();
        break;
      case '*':
      case '#':
        while (last_key == '*' || last_key == '#') {
          lcd_clear();
          char val[10];
          int x = convert(read_adc());
          uint_string(x, val, 10, 10);
          int len = 0;
          for (len = 0; val[len] != '\0'; ++len);
          int i;
```

```
      for (i = 0; i < len - 2; ++i) {
        lcd_write_c(val[i++]);
      }
      lcd_write_c('.');
      lcd_write_s(val + i);
      lcd_write_s(" V ");
      uint_string(x, val, 16, 10);
      lcd_write_s("(0x");
      lcd_write_s(val);
      lcd_write_c(')');
      delay_ms(500);
    }
    break;
  case 'A': ;
    bool insert_mode = FALSE;

    while (last_key == 'A') {
      sleep();
      if (insert_mode == TRUE) {
        if (last_serial == 0x1B) {
          insert_mode = FALSE;
        } else if (last_serial == 0x7F) {
          lcd_move_cursor(LEFT);
          lcd_write_c(' ');
          lcd_move_cursor(LEFT);
        }else {
          lcd_write_c(last_serial);
        }
      } else {
        switch (last_serial) {
          case 'h':
            lcd_move_cursor(LEFT);
            break;
          case 'k':
            lcd_move_cursor(UP);
            break;
          case 'l':
            lcd_move_cursor(RIGHT);
            break;
          case 'j':
            lcd_move_cursor(DOWN);
            break;
          case 'i':
            serial_send_s("INSERT MODE!\n\r");
            insert_mode = TRUE;
```

```
                break;
              case 0x7F:
                lcd_move_cursor(LEFT);
                lcd_write_c(' ');
                lcd_move_cursor(LEFT);
              default:
                serial_send_s("NORMAL MODE!\n\r");
                break;
            }
          }
        }
        break;
      default:
        sleep();
        break;
    }
  }
  return 0;
}
```

Headers

adc.h

```
#ifndef ADC_H
#define ADC_H

#include "helpers.h"

void init_adc(void);
int read_adc(void);
int convert(int);

void init_adc(void) {
  ADCB_CTRLA = 0x01; // Enable ADC
  PORTA_DIRCLR = 0xFF; // Input
  PORTB_DIRCLR = 0x08; // Input
  ADCB_CTRLB = 0x0C; // Signed 8 bit mode
  ADCB_REFCTRL = 0x30; // AREFB
  ADCB_PRESCALER = 0x07; // Al-gore
  ADCB_CH0_CTRL = 0b10000001; // Singled ended input
  ADCB_CH0_MUXCTRL = 0b100000;
}
```

```
int read_adc(void) {
  ADCB_CTRLA |= 0x4;
  ADCB_CH0_CTRL = 0b10000001; // Start
  while ((ADCB_CH0_INTFLAGS & 0x01) == 0x00) {
    // Wait for completion
    break;
  }
  volatile int x = ADCB_CH0_RES;
  ADCB_CH0_INTFLAGS = 0x01; // Clear flag
  return x;
}

int convert(int i) {
  return 1.875 * i - 20;
}

#endif
```

helpers.h

```
#ifndef HELPERS_H
#define HELPERS_H

#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "ebi_driver.h"

#define  IOP_START    0x8000
#define  IOP_SIZE     0x2000
#define  SRAM32_START 0x472000
#define  SRAM32_SIZE  0x8000


void delay_ms(int ms) {
  for (int i = 0; i < ms; ++i) {
    for (uint16_t j = 0; j < 390; ++j) {
      asm volatile("nop");
    }
  }
}

char* uint_string(uint16_t n, char* s, int radix, int len) {
  for (int i = 0; i < len; ++i) {
    s[i] = '\0';
```

```
  }
  if (n == 0) {
    s[0] = '0';
    return s;
  }
  int i = 0;
  while (n > 0) {
    char tmp = (n % radix) + '0';
    if (tmp > '9') {
      tmp = tmp - '9' - 1 + 'A';
    }
    s[i++] = tmp;
    n /= radix;
  }
  len = i;
  for(i = i - 1; i >= len/2; --i) {
    char tmp = s[i];
    s[i] = s[len - 1 - i];
    s[len -1 - i] = tmp;
  }
  return s;
}


void init(void) {
  // Output address lines and chip selects
  PORTH_DIR = 0b110111; // /WE, /RE, /CS0, /CS1 outputs
  PORTH_OUTSET = 0b110011; // Output 1 to active low outputs
  PORTJ_DIR = 0xFF; // Set D7-D0 as outputs
  PORTK_DIR = 0xFF; // Set A15-A0 as outputs
  EBI_CTRL = 0x01; // Select 3 port SRAM ALE Mode
  // Switches and LEDS
  EBI_CS0_CTRLA = 0b0010101; // Set CTRLA to 8K size
  EBI_CS0_BASEADDR = (uint16_t)((IOP_START >> 8) & 0xFFFF); // Set upper
  // 32K SRAM + Block
  EBI_CS1_CTRLA = 0b100001; // Set CS1 to 64K
  EBI_CS1_BASEADDR = (uint16_t)((0x470000 >> 8) & 0xFFFF); // Set upper 1
  // Zero out LEDs
  __far_mem_write(IOP_START, 0x00);
}

void intmask_set(uint8_t mask) {
  PMIC_CTRL = mask;
}
```

```
inline void enable_int(void) {
  sei();
}

inline void disable_int(void) {
  cli();
}

void sleep_mode_set(uint8_t mode) {
  set_sleep_mode(mode);
}

void enable_sleep() {
  sleep_enable();
}

void sleep() {
  sleep_cpu();
}

void disable_sleep() {
  sleep_disable();
}

#endif
```

keypad.h

```
#ifndef KEYPAD_H
#define KEYPAD_H

#include <avr/interrupt.h>
#include "helpers.h"

char KEYTAB[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

static volatile char last_key = '\0';

void init_keypad(void);
```

```
char read_keypad(void);

void init_keypad(void) {
  PORTF_DIR = 0xF0;
  PORTCFG_MPCMASK = 0x0F;
  // High Level Interrupt on Key press.
  PORTF_PIN0CTRL = 0x11;
  PORTF_INTCTRL = 0x03;
  PORTF_INT0MASK = 0x0F;
  PORTF_OUT = 0xF0;
}

char read_keypad(void) {
  for (int i = 0; i < 4; ++i) {
    PORTF_OUT = (0b0001 << i) << 4;
    for (int j = 0; j < 4; ++j) {
      asm volatile("nop\nnop\nnop");
      if (PORTF_IN & (0b0001 << j)) {
        PORTF_OUT = 0xF0;
        return KEYTAB[i][j];
      }
    }
  }
  PORTF_OUT = 0xF0;
  return 'F';
}

ISR(PORTF_INT0_vect) {
  delay_ms(1); // Debounce
  char tmp = read_keypad();
  if (tmp != 'F') {
    last_key = tmp;
  }
  PORTF_INTFLAGS |= 0x01;
}

#endif
```

lcd.h

```
#ifndef LCD_H
#define LCD_H

#include "ebi_driver.h"
```

```
#include "helpers.h"

#define  LCD_CMD    0x471000
#define  LCD_DATA   0x471001

#define FALSE 0
#define TRUE 1
typedef int bool;

#define LEFT  0
#define RIGHT 1
#define DOWN  2
#define UP    3
typedef int dir;

static uint8_t cursor_pos;
static bool lcd_on = FALSE;

void init_lcd(void);
void lcd_cmd(short);
void lcd_clear(void);
bool lcd_busy(void);
void wait_lcd(void);
void lcd_write_c(char);
void lcd_write_s(char*);
void lcd_reset_cursor(void);
void lcd_next_line(void);
int lcd_hpos(void);
int lcd_vpos(void);
void set_pos(int h, int v);
void toggle_lcd(void);
void lcd_move_cursor(dir);

void init_lcd(void) {
  // Ensure it has been at least 40ms since start. See data sheet.
  delay_ms(40);
  // Two lines
  __far_mem_write(LCD_CMD, 0b111100);
  delay_ms(40);
  __far_mem_write(LCD_CMD, 0b111100);
  delay_ms(40);
  // Now, we can safely poll the BF Flag. See data sheet.
  lcd_cmd(0b1111); // Set Cursor and Display on
  wait_lcd();
  lcd_on = TRUE;
```

```
  lcd_cmd(0b110); // Cursor increments right and shifts
  lcd_clear();
}

void lcd_write_c(char c) {
  if (c == '\n') {
    lcd_next_line();
    return;
  }
  wait_lcd();
  __far_mem_write(LCD_DATA, (int)c);
  ++cursor_pos;
  if (cursor_pos == 16) {
    cursor_pos = 40;
    lcd_cmd(0x80 | cursor_pos);
  } else if (cursor_pos == 56) {
    cursor_pos = 0;
    lcd_cmd(0x80);
  }
}

void lcd_write_s(char* string) {
  while (*string != '\0') {
    lcd_write_c(*(string++));
  }
}

void lcd_cmd(short cmd) {
  wait_lcd();
  __far_mem_write(LCD_CMD, cmd);
}

void lcd_clear(void) {
  lcd_cmd(0b1);
  cursor_pos = 0;
}

bool lcd_busy(void) {
  delay_ms(5);
  return FALSE;
  int x = __far_mem_read(LCD_CMD);
  return (x & 0x80) != 0;
}

void wait_lcd(void) {
```

```
  do {
  } while(lcd_busy() != FALSE);
}

void lcd_reset_cursor(void) {
  lcd_cmd(0b10);
}

void lcd_next_line(void) {
  if (cursor_pos < 40) {
    cursor_pos = 40;
    lcd_cmd(0x80 | cursor_pos);
  } else {
    cursor_pos = 0;
    lcd_cmd(0x80);
  }
}

int lcd_hpos(void) {
  if (cursor_pos < 16) {
    return cursor_pos;
  } else {
    return cursor_pos - 40;
  }
}

int lcd_vpos(void) {
  if (cursor_pos < 16) {
    return 0;
  } else {
    return 1;
  }
}

void set_pos(int h, int v) {
  if (v == 0) {
    lcd_cmd(0x80 | h);
    cursor_pos = h;
  } else {
    if (h - lcd_hpos() == 1) {
      lcd_cmd(0x14);
      cursor_pos += 1;
    } else if (h - lcd_hpos() == -1) {
      lcd_cmd(0x10);
      cursor_pos -= 1;
```

```
    } else {
      cursor_pos = 40;
      lcd_cmd(0x80 | 40);
      for (int i = 0; i < h; ++i) {
        lcd_move_cursor(RIGHT);
      }
    }
  }
}

void toggle_lcd(void) {
  lcd_on ^= TRUE;
  lcd_cmd(0b1111 ^ (lcd_on << 2));
}

void lcd_move_cursor(dir d) {
  switch (d) {
    case LEFT:
      if (lcd_vpos() == 1 && lcd_hpos() == 0) {
        set_pos(15, 0);
      } else if (lcd_hpos() != 0) {
        set_pos(lcd_hpos() - 1, lcd_vpos());
      }
      break;
    case RIGHT:
      if (lcd_vpos() == 0 && lcd_hpos() == 15) {
        set_pos(0, 1);
      } else if (lcd_hpos() != 15) {
        set_pos(lcd_hpos() + 1, lcd_vpos());
      }
      break;
    case UP:
      if (lcd_vpos() == 1) {
        set_pos(lcd_hpos(), 0);
      }
      break;
    case DOWN:
      if (lcd_vpos() == 0) {
        set_pos(lcd_hpos(), 1);
      }
      break;
  }
}

#endif
```

serial.h

```
#ifndef SERIAL_H
#define SERIAL_H

#include <avr/interrupt.h>

#define BSCALE -7
#define BSEL 1539

static volatile char last_serial = '\0';

void init_usart(void) {
  PORTD_DIRSET = 0x08;  // Tx to output
  PORTD_OUTSET = 0x08;  // Tx default output
  PORTD_DIRCLR = 0x04;  // Rx to input
  PORTQ_DIRSET = 0x0A;  // PortD -> USB
  PORTQ_OUTCLR = 0x0A;  // Default output
  USARTD0_CTRLB = 0x18; // Enable Rx, Tx
  USARTD0_CTRLC = 0x03; // No Parity, 1 stop bit.
  USARTD0_BAUDCTRLA = BSEL & 0xFF; // BSEL
  USARTD0_BAUDCTRLB = (((BSCALE << 4) & 0xF0) | ((BSEL >> 8) & 0x0F));
  USARTD0_CTRLA = 0x30; // High level interrupts for Rx
}

void serial_send(uint8_t data) {
  while ((USARTD0_STATUS & 0x20) == 0) {
  }
  USARTD0_DATA = data;
}

void serial_send_s(char* string) {
  while (*string) {
    serial_send(*(string++));
  }
}

ISR(USARTD0_RXC_vect) {
  last_serial = USARTD0_DATA;
}

#endif
```