Programmeringsuppgift

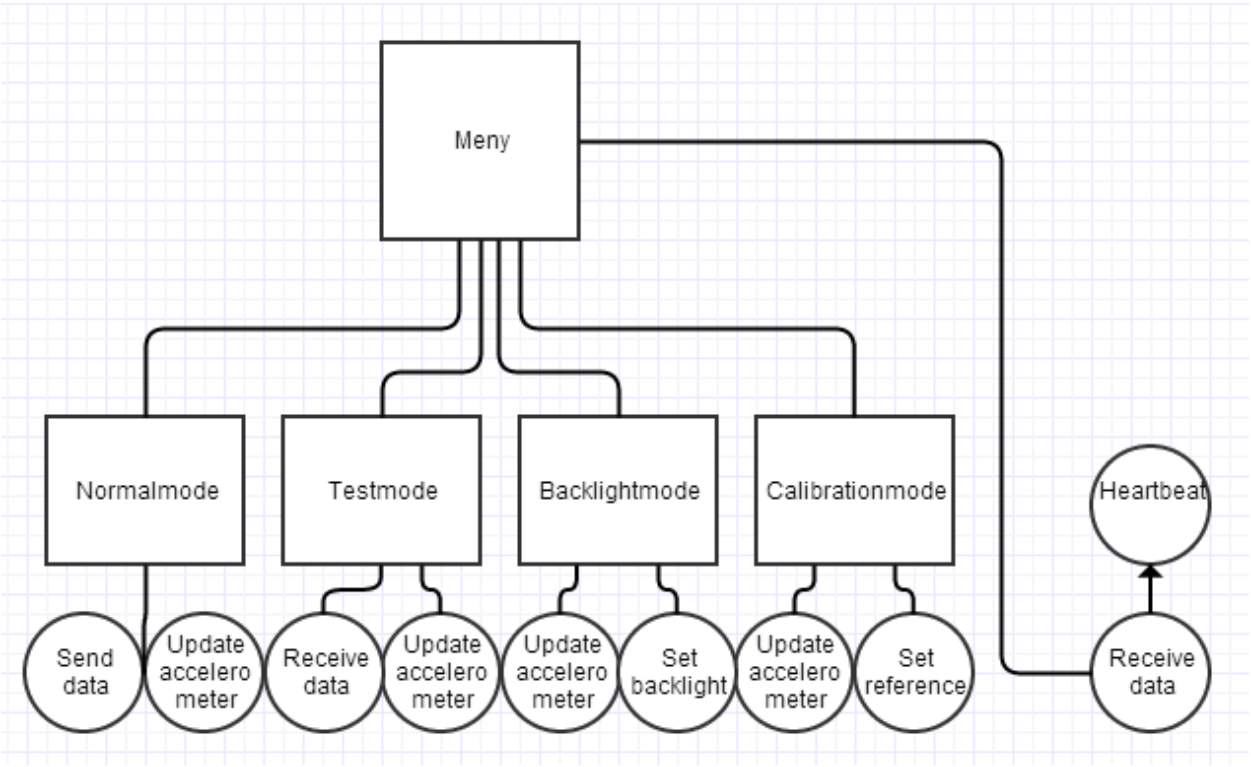
Tiltbaserad trådlös kontroll för radiobil

IE1206 - Inbyggd Elektronik

Karl Gäfvert
Rasmus Linusson

Produkt

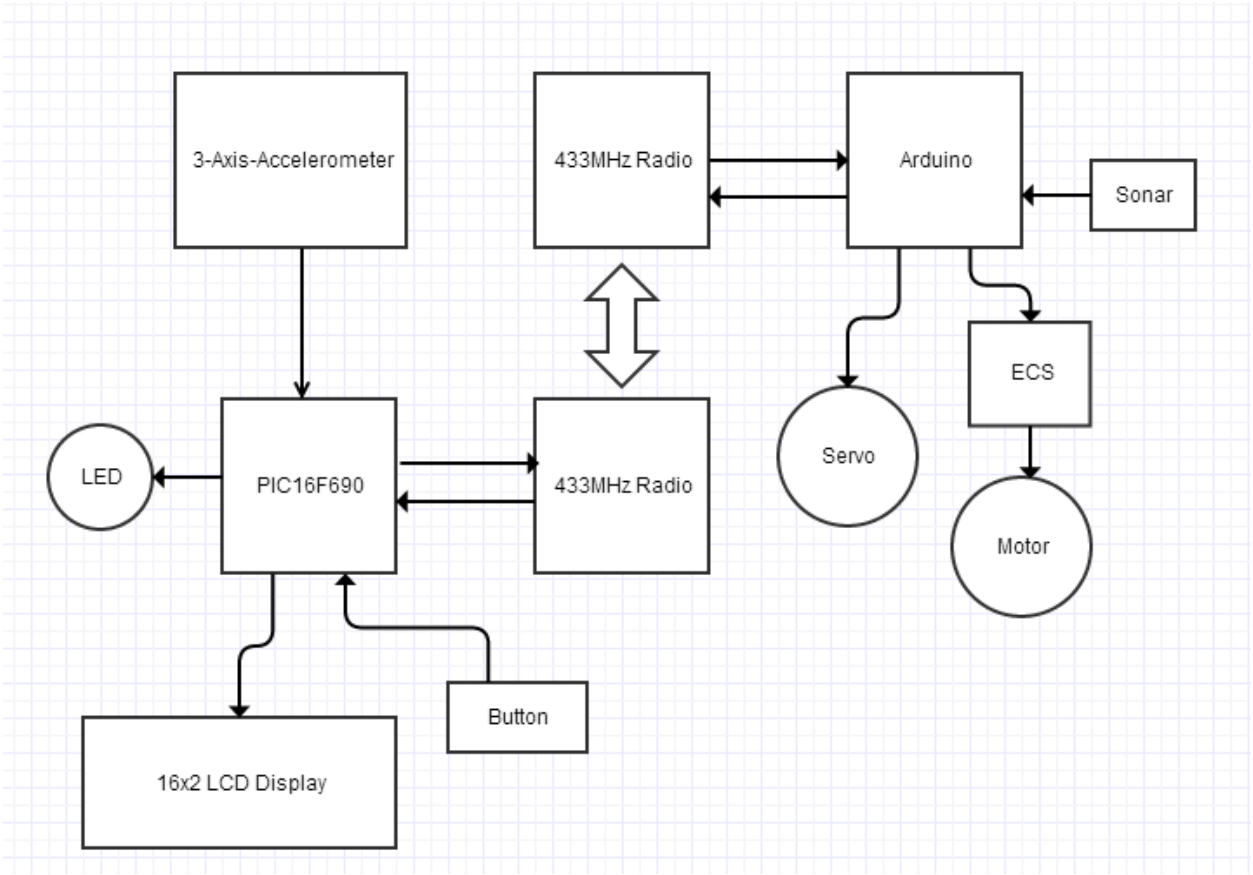
Kontroll som styr en radiostyrd leksaksbil genom att känna skillnader i lutning. Med hjälp av en accelerometer kan kontrollens orientering, eller lutning, beräknas. Dessa värden konverteras sedan till gas- och styrvärden som skickas med hjälp av en radiosändare till bilen. Bilen sänder samtidig information tillbaka till kontrollen med data från en sonar som sitter monterad längst fram på bilen. Kontrollen har en enkel meny som kontrolleras av en knapp. I menyn det går att byta mellan olika lägen, i dessa kan man, kalibrera utgångspositionen för kontrollen, bakgrundsljusstyrkan på LCD-displayen, ett testlägen som visar accelrometerns värden och referensvärden och ett normalläge där avståndet till närmasta objekt framför bilen presenteras tillsammans med en indikator som visar hur mycket bilen svänger och ett procentuellt värde över hur mycket användaren gasar. Kontrollen kan även känna om den är ansluten till bilen och indikera detta.



Beskrivning av funktionsprototyp

Vår funktionsprototyp består huvudsakligen av de hårdvarukomponeter som ilusteras på den vänstra sidan i blockdiagramet nedan. Den högra sidan beskriver komponeterna i bilen. Accelerometern beter sig som ett gyro när den tiltas och ger tre värden som analoga spänningar, två av dessa AD-omvandlas till 10-bitars heltal som sedan används för att beräkna gas och styrning. Så mäter alltså kontrollen lutningen på sig själv (breadboarden). PICen kommunicerar med radiomodulen med hjälp av sin inbyggda enhet för seriell kommunikation. Via radiomodulen skickas styr- och gasvärden till bilen i form av enkla kommandon som består av att först ett kommando och sedan ett värde skickas. PICen accepterar också inkommande data från radiomodulen med hjälp av en interrupt-rutin. Dessa data är distansvärdet som bilen skickar, det används även för att kontrollera anslutningen till bilen. En LCD-display med 2x16 tecken används för smart interaktion med användaren. På den skrivs menyer och värden beroende på hur användaren använder kontrollen. Förutom normalt läge och testläge går det att ställa bakgrundsbelysningen på LCD-displayen i en meny. Den generas av en PWM i PICen och sparas när kontrollen är avstängd i EEPROM. LCD-displayens kontrast ställs med en potentiometer. En enkel tryckknapp används för användaren ska kunna interagera med kontrollen och menyerna. Det finns också en grön LED som är tänd när kontrollen har en anslutning till bilen. Kontrollen strömförsörjs med hjälp av en 7-35V in, 5V 1A ut linjärregulator. Detta gör att kontrollen kan strömförsörjas med ett litet lätt 9V-batteri.

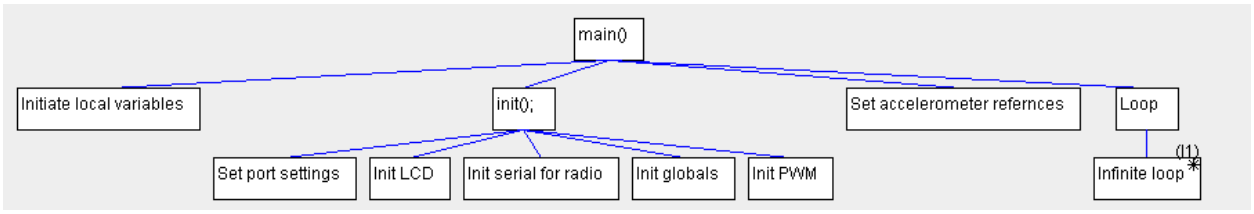
Diagram över kontrollen, bilen och hur de kommunicerar:

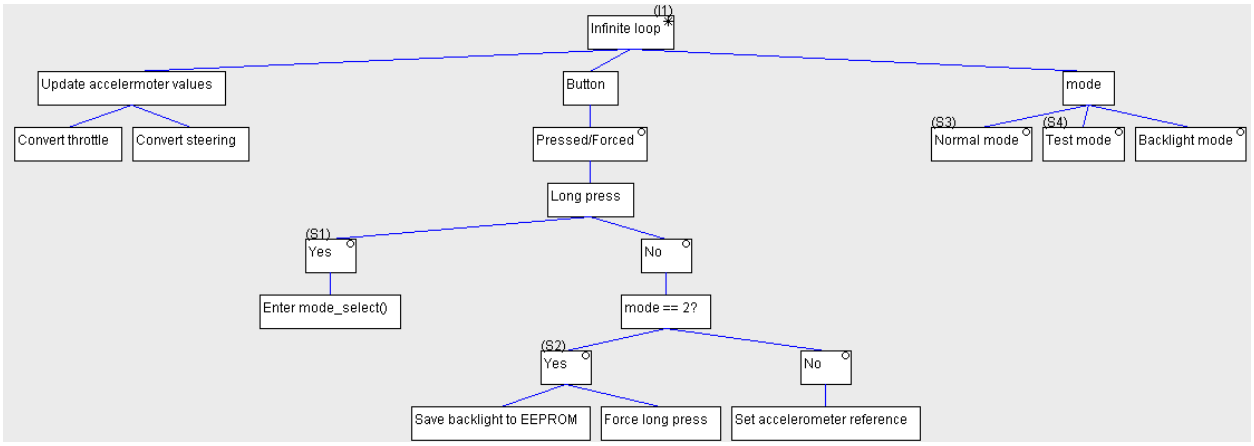


Checklista:

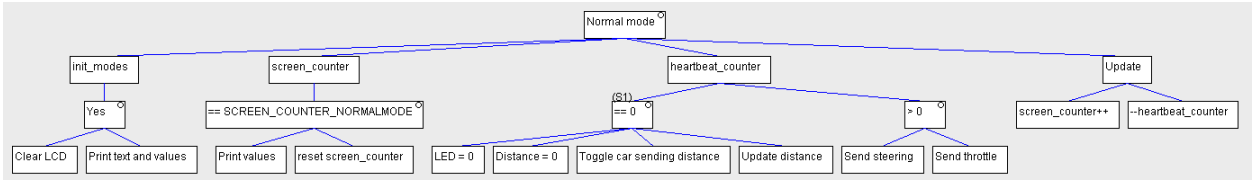
- **Meny**
  - Visar att menyn roterar vid kort klick
  - Visar att det går att komma till alla program i menyn vid långt klick
- **Testmode**
  - Visar att olika värden mottages från accelerometern och att dessa är lämpade för "simulera" ett gyroskop.
  - Kalibrering, kort tryck på knappen kalibrerar kontrollen till att utgå ifrån den orientering den har just nu.
  - Långt tryck återgår till menyn
- **Normalmode**
  - Sänder kommandon till bilen, vid tiltning åt något håll ska ge motsvarande önskad reaktion hos bilen.
  - Ett stapeldiagram på displayen som illusterar hur accelerometer-värden konverteras till något som går att tolka som styrning i höger och vänster.
  - En gas-mätare som visar hur mycket gas, med riktning, som ges till motorn.
  - Kalibrering, kort tryck på knappen kalibrerar kontrollen till att utgå ifrån den orientering den har just nu.
  - Heartbeat, visa hur kontrollen känner att bilen inte längre är kontaktbar och gröna LEDen slocknar och tänds igen när kommunikation återupprättats.
  - En distansmätare som visar avstånd till närmaste objekt framför bilen (upplösning: 0.2m ~ 10m)
  - Lågt tryck återgår till menyn
- **Backlightmode**
  - Tilta (höger-vänster) för att ändra duty cycle och backgrundsbelysningens styrka.
  - Tryck på knappen (spara).
  - Starta om, visa styrkan sparades i EEPROM och har samma värde nu.
- **Calibration mode**
  - Kalibrerar och återgå till menyn

Strukturdiagram:

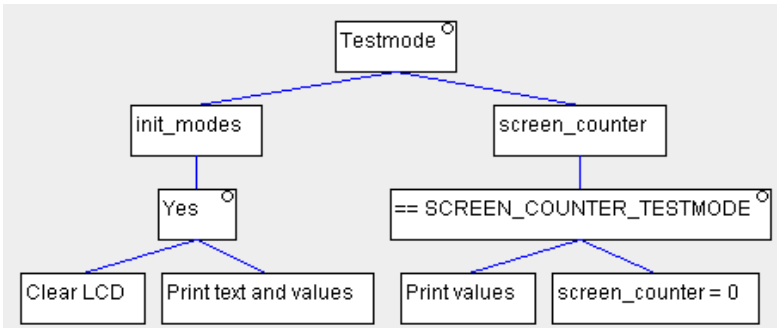




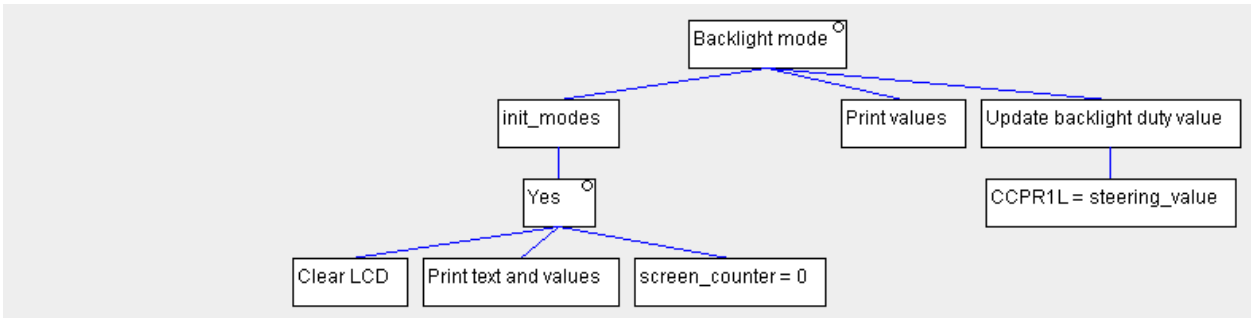
Mode - Normal



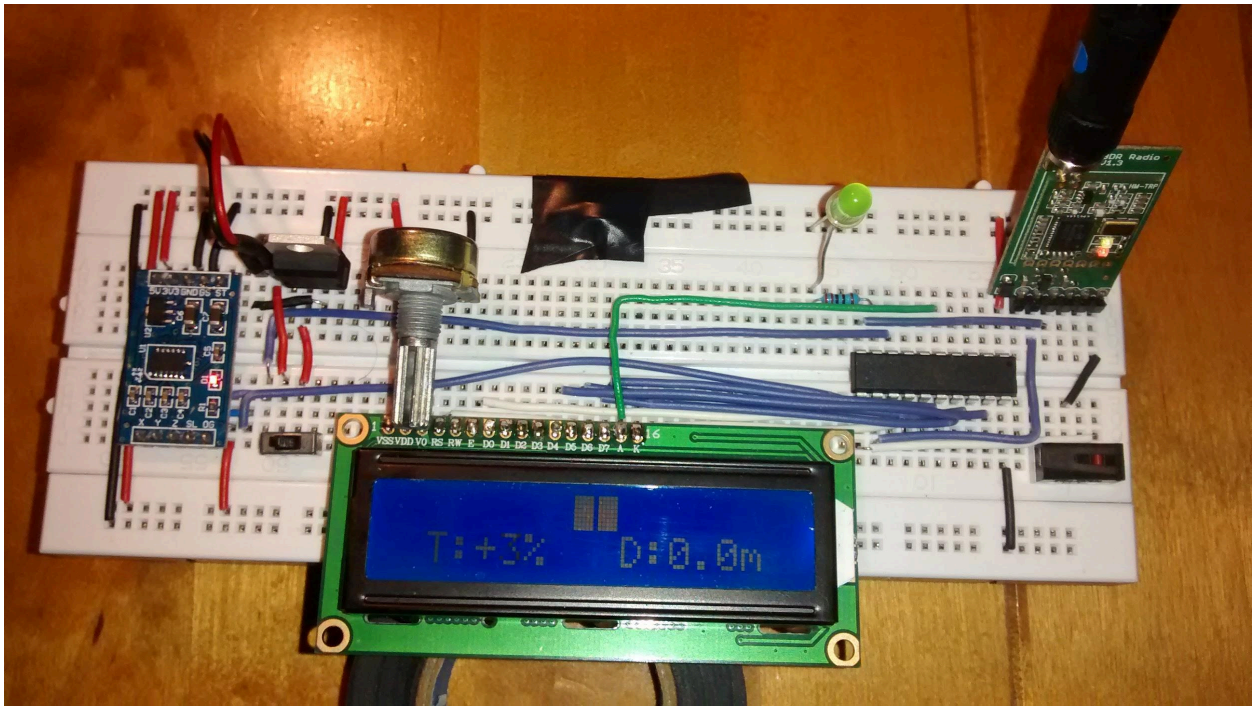
Mode - Test



Mode - Backlight



Då vi inte hade ett standard projekt eller standard komponenter har vi även bifogat ett foto av kopplingsdäcket.



Tejpat på baksidan sitter ett 9v-batteri.

Publicerad kod:

```
/* Cc5x Knudsen Compiler - not ANCI C
*
* File:             kontroll.c
* Authors:          Kalle Gäfvert and Rasmus Linusson
* Description:      Wireless car controller for programming project in course IE1206
*                   at KTH Kista
* License:          This file is in the public domain, feel free to use it however
*                   you like :) Contact authors for git-access.
* Last modified:    2014-05-12
* Indentation:      spaces, size 4
*
*/

/* Includes */
#include "16F690.h"
#include "int16Cxx.h"
#pragma config |= 0x00D4

/* Constants */
#define HEARTBEAT_RESET 900
#define SCREEN_COUNTER_NORMALMODE 150
#define SCREEN_COUNTER_TESTMODE 70
#define SCREEN_CONTER_INNER 3
#define EEPROM_BACKLIGHT_ADDRESS 0

/* A/D Port selection settings
*
* 0b10.xxxx.xx - Set right justified, no voltage reference
* 0bxx.1000.xx - Set port to convert to AN8
* 0bxx.xxxx.01 - Not running, enabled
*/
#define X_AXIS 0b10.1000.01

/*
* 0b10.xxxx.xx - Set right justified, no voltage reference
* 0bxx.1010.xx - Set port to convert to AN10
* 0bxx.xxxx.01 - Not running, enabled
*/
#define Y_AXIS 0b10.1010.01

/* LCD pin definitions */
#pragma bit D4  @ PORTA.0
#pragma bit D5  @ PORTA.1
#pragma bit D6  @ PORTA.2
```

```
#pragma bit D7  @ PORTC.0
#pragma bit RS  @ PORTC.1   // Register select
#pragma bit EN  @ PORTC.2   // Clock enable/disable

/* Button pin definition */
#define BUTTON !PORTB.6

/* Red LED */
#pragma bit LED @ PORTC.7

/* Forward declarations */
void init();

char select_mode();

void set_reference(long*, long*);
void set_reference_with_message(long*, long*);
long convert_steering(long, long);
long convert_throttle(long, long);

void ad_update_xy(long*, long*);

void print_steering(char);
void print_throttle(char, bit);
void print_distance(char, bit);
void print_normalmode(char, char, char, bit);
void print_testmode(long, long, long, long, bit);
void print_backlightmode(char, bit);

void lcd_putchar(char);
void lcd_putstring(const char*);
void lcd_putlong(long);
void lcd_putchar_one_decimal(char);
void lcd_set_rowcol(bit, char);
void lcd_clear();
void lcd_trigger_clock();

char serial_read();
void serial_putchar(char);

void set_distance_to_obstacle(char*);
void send_steering(char);
void send_throttle(char);
void send_toggle_distance_enable();

void delay(char);
void putchar_eedata(char, char);
char getchar_eedata(char);
```

```
/*
* Global to keep track of the amount of loops
* since last transmission from car.
*/
long heartbeat_counter;

/*
* Global used to slow down updates of the throttle and distance values
* on the lcd screen in order to print values in a readable manner.
*/
long screen_inner_counter;

/*
* Globals used to optimize printing, reducing the time
* spent on printing to LCD and give a more real-time feeling.
*/
char old_left, old_right;

/*
* Global used to optimize printing of the throttle value.
* Without knowing how big the last value was we
* may waste up to three characters. One character is
* equal to about 1000 instructions so this global helps
* to save lots of instructions.
*/
char old_throttle;

/* Interrupt routine
*
* The interrupt routine is called when a new byte is received from the radio.
* It saves all registers stores the received value in a global, sets a global
* new-data-has-been-received status variable and then resets all saved registers
* before returning. The interrupt routine also handles overrun error in RCREG.
*
*/
bit rx_updated;
char rx_data;

#pragma origin 4  // Put interrupt routine at address 4
interrupt int_server(void)
{
    int_save_registers

    /* Read from TX register only if there is new data */
    if(RCIF)
    {
        /* Overrun error means the data might be old or faulty
```

```
           throw the bad data away and reset receive functionality */
        if (OERR)
        {
            char trash;

            trash = RCREG;
            trash = RCREG;

            CREN = 0;
            CREN = 1;
        }
        else
        {
            /* Get the freshest/newest value */
            while (RCIF)
                rx_data = RCREG;

            rx_updated = 1;
        }
    }

    int_restore_registers
}

/*
* Main contains all top level functionality and runs a selected mode.
*/
void main (void)
{
    /* Mode select helper */
    char mode = 2;                  // Set normal mode as default
    bit button_pressed = 0;         // Button is initially not pressed
    bit enable_button = 1;          // Enable button reading if-statement
    bit force_long_press = 1;       // Force long-press on init, so we land menu on
power-up
    long long_press_counter = 0;    // Long-press counter

    /* State holders */
    bit init_modes = 0;           // Indicates if a (re-)init is needed for the (new) mode
    char send_counter = 0;        // Indicates if steering or throttle should be sent to
car

    /* Screen update counter */
    long screen_counter = 0;

    /* Accelerometer reference values */
    long x_reference, y_reference;
```

```
/* Accelerometer values */
long x_axis_value, y_axis_value;

/* Calculated throttle and steering values */
char throttle_value, steering_value;

/* Variables to hold values sent from car here.
   Only distance now, but others can be added if needed */
char distance_value = 0;

/* Initialize A/D converter, LCD, other ports and serial communication unit */
init();

/* Set the initial position of the controller to current position */
set_reference(&x_reference, &y_reference);

while(1)
{
    /* Update x and y axis x value variables with fresh data from accelerometer */
    ad_update_xy(&x_axis_value, &y_axis_value);

    /* Update distance to obstacle */
    set_distance_to_obstacle(&distance_value);

    /* Calculate throttle and steering values */
    throttle_value = convert_throttle(x_axis_value, x_reference);
    steering_value = convert_steering(y_axis_value, y_reference);

    /* If button is being pressed we wait for release, if it
          has been pressed and released it enters select mode again. */
    if ((BUTTON && enable_button) || force_long_press)
    {
        button_pressed = 1;

        ++long_press_counter;

        /* Long press */
        if (long_press_counter == 130 || force_long_press)
        {
            /* Turn of led on mode exit */
            LED = 0;

            mode = select_mode();

            button_pressed = 0;
            enable_button = 0;
            force_long_press = 0;
            init_modes = 1;
```

```
        screen_counter = 0;
    }

    delay(5);
}
/* If button was pressed and released before long_press_counter finished
   it's a short press */
else if (!BUTTON && button_pressed)
{
    /* Save back-light PWM value in EEPROM if back-light mode
       set reference otherwise */
    if (mode == 2)
    {
        /* Save back-light value to EPROM */
        putchar_eedata(steering_value, EEPROM_BACKLIGHT_ADDRESS);

        /* Go back to mode_select next loop */
        force_long_press = 1;
    }
    else
        set_reference_with_message(&x_reference, &y_reference);

    button_pressed = 0;
    enable_button = 0;
    init_modes = 1;
}
else if (!BUTTON)
{
    /* Set button release states */

    enable_button = 1;
    long_press_counter = 0;
}

if (mode == 0)
{
    /* Normal mode */

    /* Print text that don't need updating once */
    if (init_modes)
    {
        lcd_clear();

        screen_inner_counter = 0;
        print_normalmode(steering_value, throttle_value, distance_value, 1);
        screen_inner_counter = SCREEN_CONTER_INNER;

        init_modes = 0;
```

```
    }

    /* Don't update LCD to often */
    if (screen_counter == SCREEN_COUNTER_NORMALMODE)
    {
        print_normalmode(steering_value, throttle_value, distance_value, 0);
        screen_counter = 0;
    }

    /* Wait until a distance value has been received from the car */
    if (!heartbeat_counter)
    {
        LED = 0;
        distance_value = 0;

        send_toggle_distance_enable();
        set_distance_to_obstacle(&distance_value);
    }
    else
    {
        send_throttle(throttle_value);
        send_steering(steering_value);
    }

    /* Turn LED on when car makes contact after
       the contact has been lost or didn't exist */
    if (distance_value && heartbeat_counter == HEARTBEAT_RESET)
        LED = 1;

    ++screen_counter;

    if (heartbeat_counter)
        --heartbeat_counter;
}
else if (mode == 1)
{
    /* Test-mode */

    /* Print text that don't need updating once */
    if (init_modes)
    {
        lcd_clear();
        print_testmode(x_axis_value, y_axis_value, x_reference, y_reference, 1);

        init_modes = 0;
    }

    /* Print values */
```

```
            if (screen_counter == SCREEN_COUNTER_TESTMODE)
            {
                print_testmode(x_axis_value, y_axis_value, x_reference, y_reference, 0);
                screen_counter = 0;
            }

            ++screen_counter;
        }
        else if (mode == 2)
        {
            /* Back-light select mode */

            /* Print text that don't need updating once */
            if (init_modes)
            {
                lcd_clear();
                print_backlightmode(steering_value, 1);

                init_modes = 0;
            }

            /* Update bar */
            print_backlightmode(steering_value, 0);

            /* Set the back-light value in the */
            CCPR1L = steering_value; // Duty value
        }
        else if (mode == 3)
        {
            /* Recalibration mode */

            /* Set reference and then force long press to enter menu */
            if (init_modes)
            {
                set_reference_with_message(&x_reference, &y_reference);
                init_modes = 0;
            }
            else
                force_long_press = 1;
        }
    }
}

/*
* Function to gather all of the initializations that needs to be done
* once the processor starts up.
*
* There is inits for:
```

```
*   * Globals
*   * Used ports
*   * LCD display
*   * Serial communication with the radio
*   * Interrupt on serial receive
*   * Connection LED
*   * Back-light PWM
*   * Communication heartbeat
*
*/
void init()
{
    /* Make sure all ports are in general i/o mode */
    ANSEL = 0;

    /*
    * 0bxxxxx.1.x.x - Set AN10 as analog input (Y-axis accelerometer)
    * 0bxxxxx.x.x.1 - Set AN8 as analog input (X-axis accelerometer)
    * Other ports are set to general i/o mode
    */
    ANSELH = 0b00000.1.0.1;

    /*
    * 0bxxxxx.000 - Set LCD Data-pin 4-7 as output
    */
    TRISA = 0b11111.000;

    /*
    * 0b1.x.x.x.xxxx - Set RX as input
    * 0bx.1.x.x.xxxx - Set Button as input
    * 0bx.x.0.x.xxxx - Set TX as output
    * 0bx.x.x.1.xxxx - Set Accelerometer X-axis as input
    */
    TRISB = 0b1.1.0.1.0000;

    /*
    * 0b1.x.x.xx.xx.x - Set red LED as output
    * 0b1.x.0.xx.xx.x - Set PWM as output
    * 0bx.x.x.xx.00.x - Set LCD EN and RS as output
    * 0bx.x.x.xx.xx.0 - Set LCD Data-pin 7 as output
    */
    TRISC = 0b0.1.0.11.00.0;

    /* Init LCD */
    delay(40);      // Give LED-controller some time to boot

    /* Set LCD-controller to command-mode */
    RS = 0;
```

```
/* LCD starts in 8-bit mode, so change to 4-bit mode */
lcd_putchar(0b0011.0011);
delay(5);
lcd_putchar(0b0011.0010);
delay(5);
lcd_putchar(0b0011.0010);
delay(5);
lcd_putchar(0b0011.0010);

/* LCD settings */
lcd_putchar(0b00101000);    // Two lines (16+16 chars)
lcd_putchar(0b00001100);    // Display on, cursor off and blink off
lcd_putchar(0b00000001);    // Clear display
lcd_putchar(0b00000110);    // Increment mode, shift off

/* LCD in character-mode */
RS = 1;

/* Serial interrupt init */
RCIE = 1;       // Local interrupt enable
PEIE = 1;       // Peripheral interrupt enable
GIE = 1;        // Global interrupt enable

/* Serial communication init for RX/TX with radio.
   8n1 19200 Baud */
TXEN = 1;       // Transmit enable
SYNC = 0;       // Asynchronous operation
TX9  = 0;       // 8-bit TX
SPEN = 1;

BRGH  = 0;      // Settings for 19200 Baud
BRG16 = 1;      // Settings for 19200 Baud
SPBRG = 12;     // Settings for 19200 Baud

CREN = 1;       // Continuous receive enabled
RX9  = 0;       // 8-bit RX

/*
* Globals to optimize printing, reducing the time
* spent on printing to LCD and give a more real-time
* response to the car - init
*/
old_left = 1;
old_right = 1;

/* Set to 0 initially as we probably haven't read any valid data yet */
rx_updated = 0;
```

```
    /* Initially turn LED off */
    LED = 0;

    /* Initialize PWM */
    T2CON = 0b00000.1.00;       // Prescale 1:1
    CCP1CON = 0b00.00.1100;     // PWM-mode
    PR2 = 200;                  // Max value

    /* Load last back-light value from EPROM */
    CCPR1L = getchar_eedata(EEPROM_BACKLIGHT_ADDRESS);

    /* Set initial value of heartbeat counter so we'll start communication with car */
    heartbeat_counter = HEARTBEAT_RESET;
}

/*
*  Select mode prints a menu and returns the selected mode in the menu when the user
*  does a long press on the button. Short press will rotate to next option.
*/
char select_mode()
{
    /* Menu text */
    static const char menu_items[30] = {
        'N', 'o', 'r', 'm', 'a', 'l',
        'T', 'e', 's', 't',
        'B', 'a', 'c', 'k', 'l', 'i', 'g', 'h', 't',
        'R', 'e', 'c', 'a', 'l', 'i', 'b', 'r', 'a', 't', 'e'
    };

    static const char menu_item_offset[5] = {0, 6, 10, 19, 30};     // Text offsets

    /* The selected mode. 0 = Normal, 1 = Test, 2 = Back-light and 3 = Recalibrate*/
    char selected = 0;
    bit pressed = 0;
    bit update_choice = 1;
    unsigned long counter = 0;
    int i = 0;

    /* Helpers and temporaries for printing */
    char current_menu_item_offset = 0;
    char next_menu_item_offset = 0;
    char item_display_offset = 0;

    /* Clear screen */
    lcd_clear();

    /* Print once only */
```

```
lcd_putstring(" Hold to select");

while (1)
{
    /* Update screen when user has done a short press */
    if (update_choice)
    {
        lcd_set_rowcol(1, 0);

        current_menu_item_offset = menu_item_offset[selected];
        next_menu_item_offset = menu_item_offset[selected + 1];

        item_display_offset = 16 - (next_menu_item_offset -
                                    current_menu_item_offset);
        item_display_offset -= 4;
        item_display_offset /= 2;

        for (i = 0; i < item_display_offset; i++)
            lcd_putchar(' ');

        lcd_putchar(126);
        lcd_putchar(' ');

        for (i = current_menu_item_offset; i < next_menu_item_offset; i++)
            lcd_putchar(menu_items[i]);

        lcd_putchar(' ');
        lcd_putchar(126);

        for (i = 0; i < item_display_offset; i++)
            lcd_putchar(' ');

        update_choice = 0;
    }

    /* Check if button was pressed and return choice if it was long enough */
    if (BUTTON)
    {
        pressed = 1;

        ++counter;

        /* Return if long press */
        if (counter == 160)
            return selected;

        delay(5);
    }
```

```
        else
        {
            /* Short press  */
            if (pressed)
            {
                ++selected;
                selected %= 4;

                pressed = 0;
                counter = 0;
                update_choice = 1;
            }
        }
    }
}

/*
* Set reference
*
* This function calculates the average of 10 values in the controllers current
* position and sets each pointer to it's corresponding axis-value.
*
* param long*    Pointer to x-axis reference
*       long*    Pointer to y-axis reference
*
*/
void set_reference(long * x_reference, long * y_reference)
{
    long average_x = 0, average_y = 0;
    long x_axis_value, y_axis_value;
    char i;

    /* Let ADC stabilize before using it */
    for (i = 0; i < 5; i++)
        ad_update_xy(&x_axis_value, &y_axis_value);

    /* Set reference to the average of 10 values */
    for (i = 0; i < 10; i++)
    {
        ad_update_xy(&x_axis_value, &y_axis_value);

        average_x += x_axis_value;
        average_y += y_axis_value;

        delay(10);
    }

    average_x /= (uns16) 10;
```

```
    average_y /= (uns16) 10;

    *x_reference = average_x;
    *y_reference = average_y;
}

/*
* Function to set the reference to the y- and x-axis and print a message
* to the LCD display that it has been done
*
* param long*   Address to the variable to hold the x-axis reference
*       long*   Address to the variable to hold the y-axis reference
*/
void set_reference_with_message(long * x_reference, long * y_reference)
{
    set_reference(x_reference, y_reference);

    lcd_clear();

    lcd_set_rowcol(0, 1);
    lcd_putstring("Recalibration");

    lcd_set_rowcol(1, 5);
    lcd_putstring("done!");

    int i = 0;

    for (; i < 2000/255; i++)
        delay(255);
}

/*
* Function to convert the raw accelerometer data into something
* that we can use to control the car
*
* param long    Raw accelerometer data
*       long    Reference to the same axis
*
* return long   Value between 0 and 200
*/
long convert_steering(long data, long ref)
{
    signed long steering_value = data - ref;

    steering_value += 100;

    if (steering_value < 0)
        return 0;
```

```
    else if (steering_value > 200)
        return 200;

    return steering_value;
}


/*
* Function to convert the raw accelerometer data into something
* that we can use to control the car
*
* param long    Raw accelerometer data
*       long    Reference to the same axis
*
* return long   Value between 0 and 200
*/
long convert_throttle(long data, long ref)
{
    signed long throttle_value = data - ref;

    throttle_value += 100;

    if (throttle_value < 0)
        return 0;
    if (throttle_value > 200)
        return 200;

    return throttle_value;
}

/*
* Prints a line of blocks going out from the middle
* representing how much the user is turning.
* with a scale of 8 - 0 - 8
*
* param char    Value between 0-200 where 0 is full bar to the left
                and 200 is a full bar to the right
*/
void print_steering(char steering_value)
{
    char i = 0;
    char signs_left = 1, signs_right = 1;

    /* A safe zone where there will always be two bars in the middle */
    if (steering_value < 95)
    {
        signs_left = steering_value / 11;
        signs_left = 8-signs_left;
```

```
        if (signs_left > 200)
            signs_left = 1;

        signs_right = 1;
    }
    else if (steering_value > 105)
    {
        signs_left = 1;

        signs_right = steering_value - 100;
        signs_right = signs_right / 12;
    }
    else
    {
        signs_left = signs_right = 1;
    }

    /* If we do not want to print any extra blocks to the left */
    lcd_set_rowcol(0, 8-old_left);

    for (i = 8-old_left; i < 8+old_right; i++)
    {
        if (signs_left + i > 7 && i <= 8)
            lcd_putchar(255);
        else if (i - signs_right < 8 && i >= 7)
            lcd_putchar(255);
        else
            lcd_putchar(' ');
    }

    /* Update the globals as they are used to optimize the printouts to go faster */
    old_left = signs_left;
    old_right = signs_right;
}

/*
* Prints the value of the throttle, as T:±80%
* as to optimize this function we have set a couple of different
* cases where depending on the old throttle value if we need to
* remove an old percentage-sign
*
* param char    Value between 0-200 where 0 is -100% and 200 is +100%
*/
void print_throttle(char throttle_value, bit re_print)
{
    if (re_print)
    {
```

```
        lcd_set_rowcol(1, 1);

        lcd_putchar('T');
        lcd_putchar(':');
        lcd_putchar(' ');
    }

    lcd_set_rowcol(1, 3);

    if (throttle_value > 100)
    {
        throttle_value -= 100;
        lcd_putchar('+');
    }
    else
    {
        throttle_value = 100 - throttle_value;
        lcd_putchar('-');
    }

    lcd_putlong(throttle_value);

    /*
    * For optimization:
    *
    * Determines the difference in number of characters
    * between the last throttle value and the current one.
    * If we know the difference we know if we need to overwrite
    * an old '%' with a space or if we've overwritten
    * the last '%' with the throttle value.
    */
    if (old_throttle == throttle_value) ;   // There is no need to do anything, it looks
great

    else if (old_throttle == 100 && throttle_value == 100) ;    // Same goes here

    else if (old_throttle == 100 && throttle_value < 10)
    {
        lcd_putchar('%');
        lcd_putchar(' ');
        lcd_putchar(' ');
    }
    else if (old_throttle == 100 && throttle_value >= 10)
    {
        lcd_putchar('%');
        lcd_putchar(' ');
    }
    else if (old_throttle >= 10 && throttle_value < 10)
```

```
    {
        lcd_putchar('%');
        lcd_putchar(' ');
    }
    else if (old_throttle < 10 && throttle_value >= 10) ;       // Same goes over here

    else                        // if (old_throttle < throttle_value)
        lcd_putchar('%');

    old_throttle = throttle_value;
}

/*
* Prints the distance to closest object in-front of the car
* in the lower right corner, like: D: 1.4m
*
* param char    Distance to closest obstacle in-front of the car
*       bit     Whether to print text or just values
*/
void print_distance(char distance_value, bit re_print)
{
    // If we should print the whole thing or just print the value
    if (re_print)
    {
        lcd_set_rowcol(1, 9);
        lcd_putchar('D');
        lcd_putchar(':');
        lcd_putchar(' ');

        lcd_set_rowcol(1, 14);
        lcd_putchar('m');
    }

    lcd_set_rowcol(1, 11);

    lcd_putchar_one_decimal(distance_value);
}

/*
* Function to to print what is needed
* when the controller is in normal-mode
*
* param char    Value between 0 - 200 representing how much to turn, 100 being straight.
        char    Value between 0 - 200 representing how much gas to give, 0-99
                being backwards and 101-200 forward.
        char    Distance to closest obstacle in-front of the car
        bit     Whether or not to print text or just values
*/
```

```
void print_normalmode(char steering_value,
                      char throttle_value,
                      char distance_value,
                      bit re_print)
{
    /* Print steering column thing */
    print_steering(steering_value);

    /* Print throttle and distance every so often so that you actually can
       read the value but it won't impact responsiveness in car handling */
    if (!screen_inner_counter)
    {
        /* Reset the counter */
        screen_inner_counter = SCREEN_CONTER_INNER;

        /* Print throttle */
        print_throttle(throttle_value, re_print);

        /* Print distance */
        print_distance(distance_value, re_print);
    }

    --screen_inner_counter;
}

/*
* Function to to print what is needed
* when the controller is in test-mode
*
* param long    Raw x-axis value
*       long    Raw y-axis value
*       long    X-axis rest-position reference
*       long    Y-axis rest-position reference
*       bit     Whether or not to print text, or just values
*/
void print_testmode(long x_axis_value,
                    long y_axis_value,
                    long x_reference,
                    long y_reference,
                    bit re_print)
{
    if (re_print)
    {
        lcd_set_rowcol(0, 0);
        lcd_putchar('x');
    }
    else
        lcd_set_rowcol(0, 1);
```

```
    lcd_putlong(x_axis_value);
    lcd_putchar(' ');
    lcd_putlong(x_reference);

    if (re_print)
    {
        lcd_set_rowcol(1, 0);
        lcd_putchar('y');
    }
    else
        lcd_set_rowcol(1, 1);

    lcd_putlong(y_axis_value);
    lcd_putchar(' ');
    lcd_putlong(y_reference);
}

/*
* Function to print a bar representing the amount of back-light to use
*
* param char    A value between 0 - 200 where 0 means that the back-light
*               is completely off and 200 means that it's on full brightness
*       bit     Whether or not to print text or just values
*/
void print_backlightmode(char backlight_value, bit re_print)
{
    if (re_print)
    {
        lcd_set_rowcol(0, 1);

        lcd_putstring("Click to save!");
    }

    char i = 0;
    char signs = backlight_value / 14;

    // If we do not want to print any extra blocks to the left
    lcd_set_rowcol(1, 1);

    for ( ; i < 16; i++)
    {
        if (i < signs)
            lcd_putchar(255);
        else
            lcd_putchar(' ');
    }
}
```

```
/*
* Update two variables provided in the parameters with the
* values of the accelerometers x- and y-axis from the
* analog to digital converter
*
* param long* Address to variable to hold the x-axis value
*       long* Address to variable to hold the y-axis value
*/
void ad_update_xy(long * x_axis_value, long * y_axis_value)
{
    /* Variables to hold accelerometer values */
    long x_temp, y_temp;

    /* Change A/D to X-axis */
    ADCON0 = X_AXIS;

    /* Convert analog value to digital */
    GO=1;
    while(GO);

    x_temp = ADRESH*256;
    x_temp += ADRESL;

    /* Change A/D to Y-axis */
    ADCON0 = Y_AXIS;

    /* Convert analog value to digital */
    GO=1;
    while(GO);

    y_temp = ADRESH*256;
    y_temp += ADRESL;

    /* Write values */
    *x_axis_value = x_temp;
    *y_axis_value = y_temp;
}

/*
* Function borrowed from laborations at KTH
* Print a character to the LCD display
* or if RS == 1 sends a command to be interpreted
* by the HD44780 driver
*
* param char    Character to print or 8 bits of
*               data to send to the driver
*/
```

```
void lcd_putchar( char data )
{
    /* Send upper nybble */
    D7 = data.7;
    D6 = data.6;
    D5 = data.5;
    D4 = data.4;

    lcd_trigger_clock();

    /* Send lower nybble */
    D7 = data.3;
    D6 = data.2;
    D5 = data.1;
    D4 = data.0;

    lcd_trigger_clock();
}

/*
* Function borrowed from laborations at KTH
* Prints a string of characters to the LCD display
*
* param const char*     A c-style string to print
                        to the LCD display
*/
void lcd_putstring(const char * string)
{
    char i, k;

    for(i = 0; ; i++)
    {
        k = string[i];

        if( k == '\0')
            return;     // Found end of string

        lcd_putchar(k);
    }

    return;
}

/*
* Modified functionality from function with similar name
* from KTH
*
* Prints a long variable to the LCD display
```

```
*
* param long    Which number to print
*/
void lcd_putlong(long number)
{
    bit nonzero = 0;    // Indicates if found a number or zeros only when printing
    char string[6];     // Temporary buffer for reordering characters
    char i, temp;

    string[5] = '\0';

    for (i = 4; ; i--)
    {
        temp = (uns16) number % 10;
        temp += '0';

        string[i]=temp;

        if (i==0)
            break;

        (uns16) number /= 10;
    }

    for(i = 0; ; i++)
    {
        if (string[i] == '0' && !nonzero && i <= 3)
            continue;

        nonzero = 1;

        if(string[i] == '\0')
            return;     // Found end of string

        lcd_putchar(string[i]);
    }
}

/*
* Takes a char as parameter and prints it with a period as
* decimal separator between the last two digits
* Partially based on files found at KTH IE1206 laboration
*
* param char    What number to print
*/
void lcd_putchar_one_decimal(char number)
{
    char string[4];     // Temporary buffer for reordering characters
```

```
    char i, temp;

    string[3] = '\0';

    for (i = 2; ; i--)
    {
        temp = (uns16) number % 10;
        temp += '0';

        string[i]=temp;

        if (i==0)
            break;

        (uns16) number /= 10;
    }

    for(i = 0; ; i++)
    {
        if (string[i] == '0' && i == 0)
            continue;

        if(string[i] == '\0')
            return;     // Found end of string

        if (i == 2)
            lcd_putchar('.');

        lcd_putchar(string[i]);
    }
}

/*
* Move the cursor on the LCD display to a new position
*
* param bit     0/1 - top or bottom row
*       char    0-15 - which column to set the cursor to
*/
void lcd_set_rowcol(bit row, char column)
{
    /* Set cursor position command */
    char code = 0b10000000;

    /* Set row */
    code.6 = row;

    /* Set column */
    code |= column;
```

```
    RS = 0;
    lcd_putchar(code);
    RS = 1;
}

/*
* Clear all positions on the LCD display
*/
void lcd_clear()
{
    RS = 0;
    lcd_putchar(0b00000001);
    RS = 1;
}

/*
* Send a clock-pulse to the LCD display for it to process
* the four/(eight) bits on the data ports
*/
void lcd_trigger_clock()
{
    EN = 0;
    nop();
    EN = 1;
    delay(2);
}

/*
* If new serial data has been received from the radio, update
* the heartbeat counter, as the car obviously is communicating,
* and return the data that was received
*
* return char   255 (-1) is a reserved return state for when
*               there is no value to return.
*               Otherwise it's the received byte from the car
*/
char serial_read()
{
    if (rx_updated)
    {
        heartbeat_counter = HEARTBEAT_RESET;
        rx_updated = 0;
        return rx_data;
    }
    else
        return 255;
}
```

```
/*
* Send 8 bits of data to the car
*
* param char    8 bits of data to send
*/
void serial_putchar(char d_out)
{
    while (!TXIF) ;     // Wait until previous character transmitted
    TXREG = d_out;
}

/*
* Load the distance to closest obstacle in-front of the car
* into the address of the variable that is in the parameter
*
* param char*   Address to a variable to store the distance
*               to the closest obstacle in-front of the car
*/
void set_distance_to_obstacle(char * distance_value)
{
    char received_data = serial_read();
    if (received_data != 255 && received_data < 100)
        *distance_value = received_data;
}

/*
* Sends a command to the car telling it how much to turn
*
* param char    Value between 0 - 200 where 0 is full left
*               100 is straight forward and 200 is full right
*/
void send_steering(char steering_value)
{
    serial_putchar(230);                // Steering command
    serial_putchar(steering_value);     // 0 - 200
}

/*
* Sends a command to the car telling it to change the
* throttle to the parameter
*
* param char    Value between 0 - 200 where 0 is full speed backwards
*               100 is full stop and 200 is full speed forward
*/
void send_throttle(char throttle_value)
{
    serial_putchar(240);                // Throttle command
```

```
    serial_putchar(throttle_value);     // 0 - 200
}

/*
* Send a toggle command to the car, telling it to send
* sonar data to the controller
*/
void send_toggle_distance_enable()
{
    /* This command toggles the ability to send
       distance telemetry data from the car */
    serial_putchar(250);
}

/*
* Utilities
* These functions have been borrowed from
* laborations in IE1206 at KTH
*/
void delay(char millisec)
{
    OPTION = 2;     // prescaler divide by 8
    do {
        TMR0 = 0;
        while ( TMR0 < 125)     // 125 * 8 = 1000
            ;
    } while ( -- millisec > 0);
}

void putchar_eedata(char data, char adress)
{
/* Put char in specific EEPROM-address */
      /* Write EEPROM-data sequence                          */
      EEADR = adress;     /* EEPROM-data address 0x00 => 0x40 */
      EEPGD = 0;          /* Data, not Program memory        */
      EEDATA = data;      /* data to be written              */
      WREN = 1;           /* write enable                    */
      EECON2 = 0x55;      /* first Byte in command sequence    */
      EECON2 = 0xAA;      /* second Byte in command sequence   */
      WR = 1;             /* write                           */
      while( EEIF == 0) ; /* wait for done (EEIF=1)          */
      WR = 0;
      WREN = 0;           /* write disable - safety first    */
      EEIF = 0;           /* Reset EEIF bit in software      */
      /* End of write EEPROM-data sequence                   */
}
```

```
char getchar_eedata(char adress)
{
/* Get char from specific EEPROM-address */
      /* Start of read EEPROM-data sequence                */
      char temp;
      EEADR = adress;  /* EEPROM-data address 0x00 => 0x40  */
      EEPGD = 0;       /* Data not Program -memory         */
      RD = 1;          /* Read                             */
      temp = EEDATA;
      RD = 0;
      return temp;     /* data to be read                  */
      /* End of read EEPROM-data sequence                  */
}
```

```
/* ********************************************************************** */
/*                                HARDWARE                                */
/* ********************************************************************** */

/*
                                PIC 16F690


                          _____________  _____________
                         |             \/             |
                  +5V---|Vdd        16F690        Vss|---Gnd
                         |RA5                RA0/(PGD)|->-LCD D4
                         |RA4                RA1/(PGC)|->-LCD D5
                         |RA3/(Vpp)                RA2|->-LCD D6
Backlight anode LCD A-<-|RC5/PWM                  RC0|->-LCD D7
                         |RC4                      RC1|->-LCD RS (Register select)
                         |RC3                      RC2|->-LCD E  (Negative edge-triggered
clock)
 Accelerometer X-axis->-|RC6/AN8             AN10/RB4|-<-Accelerometer Y-axis
            Green LED-<-|RC7                   RB5/Rx|-<-Radio TX
             Radio RX-<-|RB7/Tx                   RB6|-<-Button
                         |_____________________________|

*/
```

```
/*
                        Hitachi HD44780 LCD

                         _________________________
                        |                         |
                        | O                   O   |
                        |       _____________     |
                        |      |             |    |
                        |      |             |    |
                        |      |             |    |
                        |      |             |    |
                        |      |             |    |
                    Gnd---|K     |             |    |
                PIC PWM->-|A     |             |    |
                PIC RA0->-|D7    |             |    |
                PIC RA1->-|D6    |             |    |
                PIC RA2->-|D5    |             |    |
                PIC RC0->-|D4    |             |    |
                          |D3    |             |    |
                          |D2    |             |    |
                          |D1    |             |    |
                          |D0    |             |    |
                PIC RC2->-|E     |             |    |
                    Gnd---|RW    |             |    |
                PIC RC1->-|RS    |             |    |
           Potentiometer->-|V0    |             |    |
                    +5V---|Vdd   |             |    |
                    Gnd---|Vss   |_____________|    |
                          |                         |
                          | O                   O   |
                          |_________________________|
*/
```

```
/*
                    MMA7361 Accelerometer

                   _______________________
                  |                       |
          AN08->-|X                    5V|-<-+5v
                  |                       |
          AN10->-|y       MMA7361    3.3V|
                  |     Accelerometer     |
                  |z                   GND|-<-GND
                  |                       |
           +5V->-|SL                   GS|-<-GND
                  |                       |
                  |0G                   ST|
                  |_______________________|
*/

/*
                    3DR Radio 433MHz
                   _______________________
                  |                       |
                  |RTS                    |
                  |                       |
            RX->-|TX                     |
                  |        3DR Radio      |
            TX-<-|RX                     |
                  |         433 MHz       |
           +5V->-|+5V                    |
                  |                       |
                  |CTS                    |
                  |                       |
           GND->-|GND                    |
                  |_______________________|
*/
```