```
//  Ultimatic7.c

// Objective: joystick interrupt outside of main loop

/*  Input: Morse keyer (Port E 4,5)
    Output: Lines for iambic keyer (Port B5, E6 or D0,1)
            (must disconnect speaker)
            Now debug output on LCD display, OK but slow

    Compiler variables to turn LCD off
    Compiler variable for output on PORT D

    Interrupts for joystick: Only up/down, center
        Up:   New mode
        Down: LEDs on/off
        Push: Exchange paddes

        Left/right joystick on same interrupt handler as paddle so not used

    Sverre Holm, 23 June 2013, LA3ZA


    No interrupt for keyer input
    No sleep mode

    Ca 2 mA current consumption LEDs
*/

/*

TODO:
- not use variable in PGMEM for LCD
- Use interrupts for keyer input

*/

#define LCD     1 // compiler directive to turn on/off LCD
#define PORTBE  1 // compiler directive to switch outputs to ports BE from D
// PORTD output is not compatible with LCD which uses the entire PORTD

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <inttypes.h>

#include "Ultimatic.h"

#include "lcd functions.h"
#include "lcd_driver.h"

//#define pLCDREG_test (*(char *)(0xEC))


// global variables
volatile char KEY = 0;
volatile char KEY_VALID = 0;

void paddle(void);

// declare global variables
int state=0, exchange=0, LEDs = 0;
volatile int keyer=0; // was 2
volatile int l_in, r_in, lll, l_out, r_out;

int main(void)
{
    char k;
    int firstEx=1; //, firstSign;

#if LCD
// mt static char   flash *statetext;
    PGM_P statetext;
    // Initial state variables
    statetext  = PSTR("DIR");


    LCD_Init();                 // initialize the LCD
#endif

    // Init port pins for keyer input
    DDRE = 0xCF; // set port E 4,5 for input

    // Init port pins for joystick PE2,3; PB4,6,7
    DDRB   |= 0xD8;
    DDRB   |= 0x00;
    PORTB  |= PINB_MASK;

    DDRE   |= 0x00;     // added | to keep inputs on E 4,5
//  DDRE   &= 0xF3;
//  PORTE  |= PINE_MASK; // no more input from E

    // Enable pin change interrupt on PORTB and PORTE
    PCMSK0 = PINE MASK; // comment out => Xchange function disappears
    PCMSK1 = PINB MASK;
    EIFR   = (1<<6)|(1<<7); // External interrupt flag register
    EIMSK  = (1<<6)|(1<<7); // External interrupt mask register


#if PORTBE // Output to keyer
// Set up output on PORT B5 (piezo) + E6 (side connector)
    DDRB |= 0x20;
    DDRE |= 0x40;

#else // Output to keyer and LEDs (only Port D)
    DDRD |= 0x0F; // D ports 0,1,2,3 - added | =or 1.6.2013
#endif


/////////////////////// main loop //////////////////////

    while(1)
    {

// chap 7 in Pardue C Programming for Microcontrollers for interrupt handling

    cli(); // disable interrupts so 'KEY' won't change while in use

        if (KEY_VALID) // check for unread key in buffer
        {
            k = KEY;
            KEY_VALID = 0;
        }
        else
            k = KEY_INVALID; // No key stroke available

    sei(); // enable interrupts


    if (k != KEY_INVALID)
    {
        switch(k)
        {
            case KEY UP:  // new mode
                keyer = keyer + 1;
                if (keyer >= 3)
                    keyer = 0;

                if (keyer==0)
                    statetext = PSTR("Dir");
                else if (keyer == 1)
                    statetext = PSTR("Ult");
                else if (keyer == 2)
                    statetext = PSTR("Sgl");
                break;
            case KEY DOWN: // Morse on LCDS on/off
                LEDs = !(0x01 & LEDs); // flip
                break;

            case KEY PUSH: // Exchange left-right
                exchange = !(0x01 & exchange); // flip
                firstEx = 1;
                break;
        }
    }

#if LCD

    if ((exchange == 1))// & (firstEx == 1))
        {
            LCD putc(3,'x'); // doesn't work with 4
            LCD UpdateRequired(1, 0);
            firstEx = 1;   // make sure LCD is changed only once
        }
        else
        {
            LCD putc(3,' ');
            LCD UpdateRequired(1, 0);
            firstEx = 1;
        }

if (statetext)
        {
            LCD puts f(statetext, 1);
            LCD Colon(0);
            statetext = NULL;
        }
#endif


// Read keyer input
    r in = (0x01) & (PINE>>4);
    l_in = (0x01) & (PINE>>5);

    if (exchange == 1) // switch left and right paddle
    {
        lll = l in;
        l in = r in;
        r_in = lll;
    }

// Main routine for all paddle handling

    paddle();

//firstSign

#if PORTBE  // final version output on ports b and e
    if (l_out == 1)
    {
        PORTE &= ~0x40;
        if (LEDs == 1) // too slow due to too often LCD Update
        {
            LCD putc(4,'-');
            LCD_UpdateRequired(1, 0);
        }
    }
    else
    {
        PORTE |= 0x40;
        if (LEDs == 1)
        {
            LCD putc(4,' ');
            LCD_UpdateRequired(1, 0);
        }
    }
    if (r_out == 1)
    {
        PORTB &= ~0x20;
        if (LEDs == 1)
        {
            LCD putc(5,'-');
            LCD_UpdateRequired(1, 0);
        }
    }
    else
    {
        PORTB |= 0x20;
        if (LEDs == 1)
        {
            LCD putc(5,' ');
            LCD_UpdateRequired(1, 0);
        }

    }


#else   // debug output on port D

// Output for [LEDS -> gnd] on pins D 2, 3:
// Output for next Butterfly, inverse, on pins D 0, 1
    if (LEDs == 1)
        PORTD = (0x0C & (l_out<<2|r_out<<3)) | (0x03 & ~(l_out<<0|r_out<<1));
    else
        PORTD = (0x03 & ~(l_out<<0|r_out<<1));
#endif
    }
}

//////////////////////// end of main loop ///////////////////////


void paddle()
{

    if (keyer == 0)   // Direct: output = input
      {
          l_out = !(0x01 & l_in); r_out= !(0x01 & r_in); // Boolean inverse
      }
      else
      {

     /*
        Direct implementation of table 3 in "K Schmidt (W9CF)
        "An ultimatic adapter for iambic keyers"
        http://fermi.la.asu.edu/w9cf/articles/ultimatic/ultimatic.html

        with the addition of the Single-paddle emulation mode
     */
        if (state==0)
        {
            if ((l in==0) & (r in==0))
            // two paddles closed, right first
            {
                state = 0;

                if (keyer==1)       // Ultimatic
                {
                    l_out = 1; r_out = 0; // change to left
                }
                else if (keyer==2)  // Single-paddle emulation
                {
                    l_out = 0; r_out = 1; // keep right
                }

            }
            else if ((l_in==0) & (r_in==1))
            {
                state = 1; l_out = 1; r_out = 0;
            }
            else if ((l_in==1) & (r_in==0))
            {
                state = 0; l_out = 0; r_out = 1;
            }
            else if ((l_in==1) & (r_in==1))
            {
                state = 0; l_out = 0; r_out = 0;
            }
        }

        else if (state==1)
        {
        if ((l in==0) & (r in==0))
        // two paddles closed, left first
            {
                state = 1;

                if (keyer==1)       // Ultimatic
                {
                    l_out = 0; r_out = 1; // change to right
                }
                else if (keyer==2)  // Single-paddle emulation
                {
                    l_out = 1; r_out = 0; // keep left
                }

            }
            else if ((l_in==0) & (r_in==1))
            {
                state = 1; l_out = 1; r_out = 0;
            }
            else if ((l_in==1) & (r_in==0))
            {
                state = 0; l_out = 0; r_out = 1;
            }
            else if ((l_in==1) & (r_in==1))
            {
                state = 0; l_out = 0; r_out = 0;
            }
        }
      }
}

/*
    These are interrupt handling routines
    based on chap 7 in Pardue C
    "Programming for Microcontrollers for interrupt handling"
*/

/*
SIGNAL(SIG_PIN_CHANGE0)  // keyer, joystick on PORTE
{
    PinChangeInterrupt();
}
*/


SIGNAL(SIG_PIN_CHANGE1)  // joystick on PORTB
{
    PinChangeInterrupt();
}


void PinChangeInterrupt(void)
{
    char buttons;
    char key;

    buttons = (~PINB) & PINB MASK;
    buttons |= (~PINE) & PINE_MASK;

    // Output virtual keys
    if      (buttons & (1<<BUTTON_A))
        key = KEY UP;
    else if (buttons & (1<<BUTTON_B))
        key = KEY DOWN;
    else if (buttons & (1<<BUTTON_C))
        key = KEY LEFT;
    else if (buttons & (1<<BUTTON_D))
        key = KEY RIGHT;
    else if (buttons & (1<<BUTTON_O))
        key = KEY_PUSH;
    else
        key = KEY_INVALID;

    if(key != KEY_INVALID)
    {
        if (!KEY_VALID)
        {
            KEY = key;          // Store key in global key buffer
            KEY_VALID = 1;
        }
    }

    EIFR = (1<<PCIF1) | (1<<PCIF0);     // Delete pin change interrupt flags
}
```