```
/* ****************************************************************************
 * Infra Red Receiver Module                                                  *
 *                                                                            *
 * Decodes Infra Red Pulse Width Modulation Code                              *
 *                                                                            *
 * ---------------------------------------------------------------------------*
 * Copyright (C) 2004 KOCH Engineering                                        *
 * Henrik J. Koch                                                             *
 * email:   mail@koch-engineering.com                                         *
 * web:     http://www.koch-engineering.com                                   *
 * (MPLAB-IDE 6.43)                                                           *
 * Version 1.0  Marts 14. 2004                                                *
 **************************************************************************** */
#include <pic.h>
#include "receiver.h"   // BYTE
#include "infrared.h"
#include <stdio.h>      // printf, sprintf
#include <string.h>     // string
#include "hal.h"

//#define DEBUG


/* ****************************************************************************
 * Infra Red Receiver Outer State Machine                                     *
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                                     *
 * The outer state machine receives '0's, '1's, and synchronisation           *
 * events from the inner state machine and assembles this information to a    *
 * valid Button Code. When a valid code has been received, the                *
 * function xxx is called with the received code.                             *
 *                                                                            *
 **************************************************************************** */
void ir_receiver_osm (BYTE event)
{

        if (event == EVENT_RESYNC)
        {
            osm_state = OSM_RESYNC;
            ism_state = ISM_HIGH;
        }
        else
        {
            if (event == EVENT_STARTBIT)
            {
                osm_state = OSM_START;
                ism_state = ISM_HIGH;
            }
        }

    switch (osm_state)
    {
        // first 16 bits always the same for every keypress
        case OSM_START:                                 break;
        case OSM_BIT01: code_common  = (event * 32768); break;
        case OSM_BIT02: code_common += (event * 16384); break;
        case OSM_BIT03: code_common += (event * 8192);  break;
        case OSM_BIT04: code_common += (event * 4096);  break;
        case OSM_BIT05: code_common += (event * 2048);  break;
        case OSM_BIT06: code_common += (event * 1024);  break;
        case OSM_BIT07: code_common += (event * 512);   break;
        case OSM_BIT08: code_common += (event * 256);   break;
        case OSM_BIT09: code_common += (event * 128);   break;
        case OSM_BIT10: code_common += (event * 64);    break;
        case OSM_BIT11: code_common += (event * 32);    break;
        case OSM_BIT12: code_common += (event * 16);    break;
        case OSM_BIT13: code_common += (event * 8);     break;
        case OSM_BIT14: code_common += (event * 4);     break;
        case OSM_BIT15: code_common += (event * 2);     break;
        case OSM_BIT16: code_common += (event );

            // now check if the first 16 common bits are correct
            if (code_common != IR_COMMON_CODE)
            {
```

```
                    LED_0 =
  /* This default processing is performed for all cases that don't return.
     If a startbit sync unexpectedly happens in the middle of a sequence, the
     state machine is reset to await the start of a sequence.  Otherwise
     (a ZEROBIT or ONEBIT is received), the state machine is advanced to
     await the next bit. */

            osm_state++;    // point to next osm state
}


/* ************************************************************************
 * Infra Red Receiver Inner State Machine                                 *
 *  ~~~~~~~~~~~~~~~~~~~~~~~~~                                             *
 * The inner state machine picks bits out of the received infra red       *
 * signal and sends them to the outer state machine.                      *
 * ********************************************************************** */
void ir_receiver_ism (void)
{
    WORD    timer1_copy;

    switch (ism_state)
    {
        case ISM_HIGH:      //          [0]
            // pulse has just gone high
            peripheral_interrupt_disable();
            // now high therefore wait for next falling edge
            set_CCP1_mode(CAPTURE_FALLING_EDGE);
            clear_CCP1_interrupt();
            peripheral_interrupt_enable();
            ism_state = ISM_LOW;
            start_timer1();     // ready to start timer on next rising edge
            IR_RECEIVE_BLINK = 0;   // turn off blink LED
            break;

        case ISM_LOW:       //          [1]
            // now we are on the falling edge of on pulse.
            // Timer1 now contains the pulse width measurement.
            // set interrupt ready for next for rising edge
            stop_timer1();  // will be started in other state
            peripheral_interrupt_disable();
            // read out timer1 value (~pulse width)
            timer1_copy = get_timer1();
            IR_RECEIVE_BLINK = 1;   // turn on blink LED

#ifdef DEBUG
            // for debug (read out pulse widths)
            if (osm_state < 15)
            {
                capture_value[osm_state] = timer1_copy;
            }
#endif


            if (  (timer1_copy > PULSE_START_LIMIT)     )
            {
                ir_receiver_osm(EVENT_STARTBIT);        // ---> START received
            }
            else
            {
                if (timer1_copy < PULSE_SHORT_LIMIT)
                {
                    ir_receiver_osm(EVENT_ZEROBIT);     // ---> '0' received
                }
                else
                {
                    if (    (timer1_copy > PULSE_LONG_LIMIT_L) && (timer1_copy < P
                    {
                        ir_receiver_osm(EVENT_ONEBIT);  // ---> '1' received
                    }
                    else
                    {
                        ir_receiver_osm(EVENT_RESYNC);  // ---> garbage received
```

```
                    }
                }
            }
            // this state has been reached on a falling edge
            //  now wait for next rising edge
            set_CCP1_mode(CAPTURE_RISING_EDGE);
            clear_CCP1_interrupt();
            peripheral_interrupt_enable();
            // new pulse detected
            clear_timer1();
            ism_state = ISM_HIGH;
            break;
    } // switch
} // function

void ir_init (void)
{
  peripheral_interrupt_disable();
  ism_state = ISM_HIGH;  // wait for falling edge
  osm_state = OSM_RESYNC;
  set_CCP1_mode(CAPTURE_FALLING_EDGE);
  clear_CCP1_interrupt();
  peripheral_interrupt_enable();
}


BYTE get_remote_button (WORD code)
{
    switch (code)
    {
        case IR_KEY0_CODE:
            return REMOTE_BUTTON0;
        case IR_KEY1_CODE:
            return REMOTE_BUTTON1;
        case IR_KEY2_CODE:
            return REMOTE_BUTTON2;
        case IR_KEY3_CODE:
            return REMOTE_BUTTON3;
        case IR_KEY4_CODE:
            return REMOTE_BUTTON4;
        case IR_KEY5_CODE:
            return REMOTE_BUTTON5;
        default:
            return REMOTE_BUTTON_NO_SUPPORT;
    }
}

void set_output_pin (BYTE keynumber)
{
    switch (keynumber)
    {
        case REMOTE_BUTTON0:
            KEY_OUTPUT_0 = 1;
            KEY_OUTPUT_1 = 0;
            KEY_OUTPUT_2 = 0;
            KEY_OUTPUT_3 = 0;
            KEY_OUTPUT_4 = 0;
            KEY_OUTPUT_5 = 0;
            break;
        case REMOTE_BUTTON1:
            KEY_OUTPUT_0 = 0;
            KEY_OUTPUT_1 = 1;
            KEY_OUTPUT_2 = 0;
            KEY_OUTPUT_3 = 0;
            KEY_OUTPUT_4 = 0;
            KEY_OUTPUT_5 = 0;
            break;
        case REMOTE_BUTTON2:
            KEY_OUTPUT_0 = 0;
            KEY_OUTPUT_1 = 0;
            KEY_OUTPUT_2 = 1;
            KEY_OUTPUT_3 = 0;
            KEY_OUTPUT_4 = 0;
```

```
                KEY_OUTPUT_5 = 0;
                break;
            case REMOTE_BUTTON3:
                KEY_OUTPUT_0 = 0;
                KEY_OUTPUT_1 = 0;
                KEY_OUTPUT_2 = 0;
                KEY_OUTPUT_3 = 1;
                KEY_OUTPUT_4 = 0;
                KEY_OUTPUT_5 = 0;
                break;
            case REMOTE_BUTTON4:
                KEY_OUTPUT_0 = 0;
                KEY_OUTPUT_1 = 0;
                KEY_OUTPUT_2 = 0;
                KEY_OUTPUT_3 = 0;
                KEY_OUTPUT_4 = 1;
                KEY_OUTPUT_5 = 0;
                break;
            case REMOTE_BUTTON5:
                KEY_OUTPUT_0 = 0;
                KEY_OUTPUT_1 = 0;
                KEY_OUTPUT_2 = 0;
                KEY_OUTPUT_3 = 0;
                KEY_OUTPUT_4 = 0;
                KEY_OUTPUT_5 = 1;
                break;
            default:
                KEY_OUTPUT_0 = 0;
                KEY_OUTPUT_1 = 0;
                KEY_OUTPUT_2 = 0;
                KEY_OUTPUT_3 = 0;
                KEY_OUTPUT_4 = 0;
                KEY_OUTPUT_5 = 0;
                break;
    }
}

BYTE ReadKey(void)
{
// service function to return
// what button has been pressed (0-5)
    return get_remote_button(code_separate);
}
```