```
/********************************************************************
 * File:        rtc_sys_isr.c
 * Author:      Bryant Gonzaga
 * Created:     4/12/2018
 * Modified:    4/12/2018
 *
 * Notes:
 *  Intended for ATmega128.
 *
 * Description:
 *  Interrupt subroutines.
 *******************************************************************/

//********************************************************************
*******
//
// File Name            : keyscan_isr.c
// Title                : Keypad scan interrupt service routine
// Date                 : 02/07/10
// Version              : 1.0
// Target MCU           : ATmega128 @  MHz
// Target Hardware      ;
// Author               : Ken Short
// DESCRIPTION
// When keypad interrupt occurs, key matirx is scanned and is encoded
using
// a table lookup. The keypad is connected to PORTC. See diagram in
laboratory
// description.
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//********************************************************************
******

//#define debug   //this can be uncommented to remove delays for
simulation

/* Include Libraries */
#include <iom128.h>         //Atmega128 definitions
#include <intrinsics.h>     //Intrinsic functions.
#include <avr_macros.h>     //Useful macros.

#include "fsm_defs.h"
#include "ds1306_rtc_driver.h"
/*
* Port pin numbers for columns and rows of the keypad
*/
//PORT Pin Definitions.
#define COL1  7   //pin definitions for PortB
```

```
#define COL2  6
#define COL3  5
#define COL4  4
#define ROW1  3
#define ROW2  2
#define ROW3  1
#define ROW4  0

#define INT0  0   //pin definitions for PortD

/* Lookup table declaration */
const char tbl[16] = {1, 2, 3, 15, 4, 5, 6, 14, 7, 8, 9, 13, 10, 0,
11, 12};

/* FSM Function in fsm.c */
extern void fsm(state ps, key key);

//==============================================================//
//                    Static Functions                          //
//==============================================================//
static void check_release(void)
{
#ifndef debug
  while(!TESTBIT(PIND,INT0));     //Check that keypad key is released.

  __delay_cycles(50000);          //Delay (.05secs) / (1 / 1MHz)
cycles.

  while(!TESTBIT(PIND,INT0));     //Check that key has stopped
bouncing.
#endif
}

static key keycode_to_keyenum(unsigned char key_code)
{
    if (key_code < 10) {
        return num_keys;
    }
    switch (key_code) {
        case  SET_ALARM_KEY:
            return set_alarm_key;
        case  SET_TIME_KEY:
            return set_time_key;
        case  CONFIRM_KEY:
            return confirm_key;
        case  CANCEL_KEY:
            return cancel_key;
    }
    return eol;
}

//==============================================================//
//                Interrupt Subroutines                         //
//==============================================================//
/**
 * Interrupt subroutine for key presses.
 */
```

```
#pragma vector=INT0_vect          // Declare vector location.
__interrupt void keypad_isr(void)   // Declare interrupt function
{
  extern char key_code;                   // Holds key_code
  extern state present_state;

  //Note: TESTBIT returns 0 if bit is not set and a non-zero number
otherwise.

  if(!TESTBIT(PINC,ROW1))         //Find Row of pressed key.
    key_code = 0;
  else if(!TESTBIT(PINC,ROW2))
    key_code = 4;
  else if(!TESTBIT(PINC,ROW3))
    key_code = 8;
  else if(!TESTBIT(PINC,ROW4))
    key_code = 12;

  DDRC = 0x0F;                    //Reconfigure PORTC for Columns.
  PORTC = 0xF0;

#ifndef debug
  __delay_cycles(256);            //Let PORTC settle.
#endif

  if(!TESTBIT(PINC,COL1))         //Find Column.
    key_code += 0;
  else if(!TESTBIT(PINC,COL2))
    key_code += 1;
  else if(!TESTBIT(PINC,COL3))
    key_code += 2;
  else if(!TESTBIT(PINC,COL4))
    key_code += 3;

  DDRC = 0xF0;                    //Reconfigure PORTC for Rows.
  PORTC = 0x0F;

  key_code = (tbl[key_code]);

  fsm(present_state, keycode_to_keyenum(key_code));

  check_release();                //Wait for keypad release.
}

/**
 * Interrupt subroutine for the 1Hz output from the rtc.
 */
#pragma vector=INT1_vect
__interrupt void rtc_1hz_isr()
{
    /* Display New Data */
    fsm(present_state, rtc_1hz_key);
}

/**
 * Interrupt subroutine for alarm 0, generated by the rtc.
 */
```

```
#pragma vector=INT2_vect
__interrupt void alarm_zero_isr()
{
}
```