SORTES - Clock

Dieter Castel (0256149)
Jonas Devlieghere (0256709)

November 18, 2013

Contents

1 User Documentation

Two buttons allow the user to interact with the device. The bottommost is **button 1** and is used to browse trough a items such as numbers or menu items. The topmost is **button 2** and is generally used to confirm your selection or to enter configuration mode.

1.1 Configuration Mode

To enter configuration mode from regular mode (i.e. the clock is displayed) press **button 2**. A menu will display allowing you to change the current time and alarm or quit configuration mode. When the devices powers on, you will automatically enter configuration mode since the current time has not been set. This means you will not have to press **button 2** to enter configuration mode.

1.2 Setting the Clock

To configure the current time, press **button 2** to enable configuration mode if not yet enabled. Press **button 1** until **Set time?** is displayed. Confirm this choice by pressing **button 2**. You will be able to configure the clock in 3 simple steps respectively setting the hours, minutes and seconds. Use **button 1** to increase the value of each property. The input will automatically wrap around when the maximum value is reached. For example, pressing button 1 when the current hour value is 23 will yield a value of 0. Confirm each input value by pressing button 2. When all values are set you will return to the configuration menu.

1.3 Setting the Alarm

Configuring the alarm is almost identical to configuring the current time. Press **button 2** to enter configuration mode, navigate to **Set alarm?** by pressing **button 1** and press **button 2** once more. Follow the steps mentioned above to configure the alarm time as desired.

2 System Documentation

We provided a makefile to compile the source code. Run the following command:

```
$ make clock
```

To deploy the clock.hex file to the PIC a shell script is available. The script will start tftp and wait for input from the user.

```
$ ./deploy.sh
```

Enter the following command but do not press return just yet. Reset the PIC and wait for the corresponding LED on the router to light up, then press return.

```
put clock.hex
```

When all of this succeeds, you'll see something like this. The amount of time and bytes may differ.

```
$ make clock
#################### BUILD INIT ####################
sdcc -mpic16 -p18f97j60 -L /usr/local/lib/pic16 -
   llibio18f97j60.lib -llibdev18f97j60.lib -llibc18f
   .lib -L include objects/clock.o objects/
   LCDBlocking.o objects/newtime.o
message: using default linker script "/usr/local/
   share/gputils/lkr/18f97j60.lkr"
#################### BUILD DONE ####################
$ ./deploy.sh
starting tftp to 192.168.97.6
put clock.hex
tftp> tftp> Sent 35956 bytes in 2.1 seconds
```

3 System Design

A Source Code

Listing 1: strings header file

```
#define HOURS "Hours:"
#define MINUTES "Minutes:"
#define SECONDS "Seconds:"
#define CM_STRING "Choose mode:"
#define CM_QUIT_STRING "Quit config mode."
#define CM_ALARM_STRING "Set alarm?"
#define SM_ALARM_STRING "Set alarm:"
```

```
#define CM_CLOCK_STRING "Set clock?"
#define SM_CLOCK_STRING "Set clock:"
```

Listing 2: clock body file

```
// SDCC specific defines.
#define __18F97J60
#define __SDCC__
#define THIS_INCLUDES_THE_MAIN_FUNCTION

#define OVERFLOW_CYCLES   93
#define CONFIG_MODE_QUIT  -1
#define CONFIG_MODE_ALARM   0
#define CONFIG_MODE_CLOCK   1

#include <stdlib.h>
#include <stdio.h>

#include "../Include/HardwareProfile.h"
#include "../Include/LCDBlocking.h"

#include "strings.h"
#include "time.h"
#include "clockio.h"

void init(void);
void init_config(void);
void init_time(time t, char *);

void toggle_second_led(void);
void toggle_alarm_led(void);

// Clock time
time _time;

// Alarm time
time _alarm;

// State indicators
int alarm_going_off;
```

```
// Counters
int alarm_counter;
int overflow_counter;

// Dummy button registers
int but1_pressed;
int but2_pressed;

// Flags for marking mode.
int config_called;
int config_mode_on;
int time_update_needed;

/**
 * Initializes the program and main loop for
    checking
 *  for configuration input and updating the LCD.
 */
int main(void){
  // Initialize variables.
  init();
  // Initialize configuration mode.
  init_config();
  // Do first display update.
  display_update(_time);
  while(1){
    if(time_update_needed){
      time_update_needed = 0;
        display_update(_time);
    }
    if(config_called){
      config_called =0;
      init_config();
    }
  }
}

/**
 * Start the configuration mode.
```

```
 *  This mode is only for setting the alarm or clock
    .
 */
void init_config(void){
  // -1 is quit, 0 is alarm , 1 is clock.
  int choice = CONFIG_MODE_ALARM;
  static char *choice_string = CM_ALARM_STRING;
  config_mode_on = 1;
  display_line(CM_STRING,choice_string);
  while(1){
    if(read_and_clear(&but2_pressed)){
      //Configure the selected config mode.
      switch(choice){
        case CONFIG_MODE_ALARM:
          LCDErase();
          init_time(_alarm, SM_ALARM_STRING);
          display_line(CM_STRING,choice_string);
          break;
        case CONFIG_MODE_CLOCK:
          LCDErase();
          init_time(_time, SM_CLOCK_STRING);
          T0CONbits.TMR0ON = 1;
          display_line(CM_STRING,choice_string);
          break;
        default:
          LCDErase();
          config_mode_on = 0;
          return;
      }
    }
    if(read_and_clear(&but1_pressed)){
      //Cycle trough the config modes.
      switch(choice){
        //For the alarm.
        case CONFIG_MODE_QUIT:
          LCDErase();
          choice = CONFIG_MODE_ALARM;
          choice_string = CM_ALARM_STRING;
          display_line(CM_STRING,choice_string);
          break;
```

```
        //For the clock.
        case CONFIG_MODE_ALARM:
          LCDErase();
          choice = CONFIG_MODE_CLOCK;
          choice_string = CM_CLOCK_STRING;
          display_line(CM_STRING,choice_string);
          break;
        //For quiting.
        case CONFIG_MODE_CLOCK:
          LCDErase();
          choice =CONFIG_MODE_QUIT;
          choice_string = CM_QUIT_STRING;
          display_line(CM_STRING,choice_string);
          break;
      }
    }
  }
}

/**
 * Sets the given timer with what the user inputs.
 */
void init_time(time t, char *mode){
  int h, m, s;
  h = get_input(24, HOURS,   mode, &but1_pressed, &
     but2_pressed);
  m = get_input(60, MINUTES, mode, &but1_pressed, &
     but2_pressed);
  s = get_input(60, SECONDS, mode, &but1_pressed, &
     but2_pressed);
  time_set(t,h,m,s);
}

/**
 * Toggle the first (red) LED.
 */
void toggle_second_led(void){
  LED0_IO^=1;
}
```

```
/**
 * Toggle the second and third (orange) LEDs.
 */
void toggle_alarm_led(void){
  LED1_IO^=1;
  LED2_IO^=1;
}

/**
 * Handles the high priority interupts.
 *  Currently both buttons and ticks have high
    priority.
 */
void highPriorityInterruptHandler (void) __interrupt
   (1){
  // Button 2 causes an interrupt
  if(INTCON3bits.INT1F == 1){
  if(!config_mode_on){
    config_called =1;
  } else {
    but2_pressed = 1;
  }
  if(BUTTON0_IO);
    INTCON3bits.INT1F = 0;
  }

  // Button 1 causes an interrupt
  if(INTCON3bits.INT3F  == 1){
    but1_pressed = 1;
    if(BUTTON1_IO);
    INTCON3bits.INT3F = 0;
  }

  // Timer 0 causes an interrupt
  if(INTCONbits.TMR0IF == 1) {
    overflow_counter++;
    if(overflow_counter == OVERFLOW_CYCLES/2){
      toggle_second_led();
    }else if(overflow_counter == OVERFLOW_CYCLES){
      if(time_equals(_alarm,_time)){
```

```
        alarm_going_off = 1;
      }
      if(alarm_going_off){
        alarm_counter++;
        toggle_alarm_led();
        if(alarm_counter==30){
          alarm_going_off =0;
          alarm_counter = 0;
        }
      }
      overflow_counter = 0;
      toggle_second_led();
      add_second(_time);
      if(!config_called && !config_mode_on){
        time_update_needed = 1;
      }
    }
    INTCONbits.TMR0IF = 0;
  }
}

/**
 * Inintializes all kinds of settings.
 */
void init(void){
  // Initialize LCD
  LCDInit();

  // Initialize time
  _time = time_create();
  _alarm = time_create();

  // Enable buttons
  BUTTON0_TRIS = 1;
  BUTTON1_TRIS = 1;

  // Enable interrupts
  INTCONbits.GIE = 1;
  INTCONbits.PEIE = 1;
  RCONbits.IPEN = 1;
```

```
// Disable timer
T0CONbits.TMR0ON = 0;

// Empty timer: high before low (!)
TMR0H = 0x00000000;
TMR0L = 0x00000000;

// Enable 16-bit operation
T0CONbits.T08BIT = 0;

// Use clock as clock source
T0CONbits.T0CS = 0;

// Unassign prescaler
T0CONbits.PSA = 1;

// Enable timer and interrupts
INTCONbits.TMR0IE = 1;

// Enable button interrupts
INTCON3bits.INT1IE = 1;
INTCON3bits.INT3IE = 1;

// Enable leds
LED0_TRIS = 0;
LED1_TRIS = 0;
LED2_TRIS = 0;
LED3_TRIS = 0;

// Disable all LED but backlight
LED0_IO = 0;
LED1_IO = 0;
LED2_IO = 0;
LED3_IO = 1;

// INITIALIZE OUR OWN VARIABLES.
// State indicators
alarm_going_off = 0;
```

```
  // Counters
  alarm_counter = 0;
  overflow_counter = 0;

  // Dummy button registers
  but1_pressed = 0;
  but2_pressed = 0;

  // FLAGS FOR MARKING MODE.
  config_called = 0;
  config_mode_on = 0;
  time_update_needed =0;
}
```

Listing 3: clockio header file

```
#ifndef __CLOCKIO_H_
#define __CLOCKIO_H_

#define __18F97J60
#define __SDCC__

// Defines for easy use of the LCD.
#define START_FIRST_LINE 0
#define START_SECOND_LINE 16

// INCLUDES
#include <stdlib.h>
#include <stdio.h>

#include "../Include/HardwareProfile.h"
#include "../Include/LCDBlocking.h"

#include "time.h"

void display_string(BYTE pos, char* text);
void display_update(time t);
void display_line(char *top, char *bottom);
int get_input(int maxvalue, char *text, char *mode,
   int * btn_next, int *btn_confrm);
```

```
char* to_double_digits(int value);
int read_and_clear(int *variable);

#endif
```

Listing 4: clockio body file

```
#include "clockio.h"

/**
 * Displays the given string at the given position
    on the LCD.
 */
void display_string(BYTE pos, char* text){
  BYTE        l = strlen(text);
  BYTE      max = 32-pos;
  char       *d = (char*)&LCDText[pos];
  const char *s = text;
  size_t      n = (l<max)?l:max;
  if (n != 0)
    while (n-- != 0)*d++ = *s++;
  LCDUpdate();
}

/**
 * Updates the display and prints the current time.
 */
void display_update(time t){
  char display_line[32];
  time_print(t, display_line);
  display_string(0, display_line);
}

/**
 * Display strings on first and second line of LCD
    display.
 */
void display_line(char *top, char *bottom){
  display_string(START_FIRST_LINE, top);
  display_string(START_SECOND_LINE, bottom);
```

```
}

/**
 * Gets the desired value for the given setting.
 */
int get_input(int maxvalue, char *text, char *mode,
   int * btn_next, int *btn_confrm){
  BYTE length = strlen(text);
  int value = 0;
  display_line(mode, text);
  while(1){
    if(read_and_clear(btn_confrm)){
      LCDErase();
      return value;
    }
    if(read_and_clear(btn_next)){
      value = (++value)%maxvalue;
    }
    display_string(START_SECOND_LINE + length + 1,
       to_double_digits(value));
  }
}

/**
 * Returns a pointer to a string of the double digit
     representation of the given value.
 */
char* to_double_digits(int value){
  static char buffer[3];
  sprintf(buffer, "%02d", value);
  return buffer;
}

/**
 * Returns whether the given int represents true and
     sets it to false.
 */
int read_and_clear(int *variable){
  if(*variable){
    *variable = 0;
```

```
    return 1;
  }
  return 0;
}
```

Listing 5: time header file

```
#ifndef __NTIME_H_
#define __NTIME_H_

struct time_struct;
typedef struct time_struct *time;

time time_create();
void time_set(time t, int hours, int minutes, int
   seconds);

int set_hours(time t, int value);
int set_minutes(time t, int value);
int set_seconds(time t, int value);

void add_second(time t);
void add_minute(time t);
void add_hour(time t);

void time_print(time t, char* str);
int time_equals(time t1, time t2);

#endif
```

Listing 6: time body file

```
#include "time.h"

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

unsigned char _MALLOC_SPEC heap[32];

struct time_struct {
```

```
  int hours;
  int minutes;
  int seconds;
};

time time_create(){
    time t = (time)malloc(sizeof (struct time_struct
       ));
    time_set(t,0,0,0);
    return t;
}

void time_set(time t, int hours, int minutes, int
   seconds){
  set_hours(t,hours);
  set_minutes(t,minutes);
  set_seconds(t,seconds);
}

int set_hours(time t, int value){
  int overflow = value/24;
  t->hours = value%24;
  return overflow;
}

int set_minutes(time t, int value){
  int overflow = value/60;
  t->minutes = value%60;
  return overflow;
}

int set_seconds(time t, int value){
  int overflow = value/60;
  t->seconds = value % 60;
  return overflow;
}

void add_second(time t){
  if(set_seconds(t,t->seconds + 1) != 0)
    add_minute(t);
```

```
}

void add_minute(time t){
  if(set_minutes(t,t->minutes + 1) != 0)
    add_hour(t);
}

void add_hour(time t){
  set_hours(t,t->hours + 1);
}

void time_print(time t, char* str){
  sprintf(str, "%02d:%02d:%02d", t->hours, t->
     minutes, t->seconds);
}

int time_equals(time t1, time t2){
  if(t1->seconds != t2->seconds)
    return 0;
  if(t1->minutes != t2->minutes)
    return 0;
  if(t1->hours != t2->hours)
    return 0;
  return 1;
}
```