SORTES - Clock

Dieter Castel (0256149)
Jonas Devlieghere (0256709)

November 19, 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:

```
1 $ 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

3.1 Specifications

The program provides an configurable clock with alarm function. At startup the user can set the clock and optionally the alarm using two button on the device. Once the time is set an orange led blinks each half a second. When the current time equals the alarm time an alarm goes of. This means two red LEDs start blinking one second on, one second of during the next 30 seconds. During operation both the alarm and time can be adjusted. Setting an alarm does not influence the current time.

3.2 Structural Choices

3.2.1 Timer

To effectively measure time we are using a hardware timer provided by the PIC. This timer will interrupt when its buffer overflows. We prefer this method over working with software delays because of it's increase in accuracy. Software timers are more easily influenced by (possible unknown) software implementations (i.e. arithmetics).

The timer can be operated in either 8 or 16 bit mode. This marks the length of its buffer and thus the delay between a software interrupt arises. Operating in 16 bit mode opposed to 8 bit may increase accuracy between interrupts but on the other hand might introduce a too rough granularity. A software counter is used to count the amount of interrupts between the elapse of one second. Empirical testing has shown us that this last effect is indeed a problem. Therefore, we have increased the amounts of interrupts and have chosen to operate the timer in 8 bit mode.

3.2.2 Buttons

Buttons are too implemented using interrupts. When a button is pressed a dummy register is set to true. The registers are only accessed using a "read-and-clear" operation. This ensures that every button press is read atomically. The alternative is reading the actually register. This approach requires a certain delay between two read operations to ensure a single press is not interpreted as more than one. Our approach does not have this problem. The disadvantage however consists of the impossibility to press a button continuously e.g. for increasing a value without releasing the button.

3.2.3 Time Structure

Storing the current time can be done in a number of ways. We could've kept a counter of nano- or milliseconds since the beginning of time, the start of the device or since midnight. We chose for a different approach in order to save space. A structure with 3 fields (hours, minutes and seconds) is used to store the current time. This had the additional advantage of introducing a certain level of abstraction. Furthermore, this structure also represents the time of the alarm.

Test value	Lower Value	Middle	Upper Value	Result
/	24320	24448	24576	/
24448	24320	24384	24448	fast
24384	24384	24416	24448	slow
24416	24384	24400	24416	fast
24400	24400	24408	24416	GOOD

Table 1: The approximation of the amount of counter overflows in one second by binary search.

3.3 Calibration

In order to calibrate our clock we needed to find out how many interrupts occur over the course of one second. At first instance, when we were still using the 16 bit timer, we found that the correct value was between 95 en 96 interrupts per second. This meant our counting system was to course-grained. This is when we switched to operating the timer in 8 bit mode, allowing more interrupts to occur.

Using a binary search to obtain the optimal value, we found that **24408** interrupts correspond to one second. In table 1 you can see how we acquired the optimal value. The first upper and lower values were found by using the counter in 16-bit mode. Using that mode we found that one second falls between 95 and 96 overflows of the counter. Since the counter in 8-bit mode overflows 256 times faster we find the initial values: 24320 $(256 * 95)$ and24576 $(256 * 96)$.

3.4 Technical Peculiarities

If you want to make use of button **1** you need to make use of register "INTCON3bits.INT**3**F". For button **2** instead, you need to use the register "INTCON3bits.INT**1**F". This is also poorly documented.

For using structures malloc() is needed (which is included in the C library for the PIC16) but what is not included is actually creating the stack. That's why you have to manually assign space for the stack. How this is done can be seen on line 7 of listing 6 on page 18.

Another headache was setting the correct size of the stack. Sometimes the stack was too large, sometimes it was too small. In both cases it caused the

time structure to represent garbage. Fortunately this was fairly easy visible when displaying the current time. Adjusting the size on the other hand was not, reasoning didn't seem to get us anywhere so we ended up using trail and error to determine a workable amount.

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   24408
#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"
```

```
20
21 void init(void);
22 void init_config(void);
23 void init_time(time t, char *);
24
25 void toggle_second_led(void);
26 void toggle_alarm_led(void);
27
28 // Clock time
29 time _time;
30
31 // Alarm time
32 time _alarm;
33
34 // State indicators
35 int alarm_going_off;
36
37 // Counters
38 int alarm_counter;
39 int overflow_counter;
40
41 // Dummy button registers
42 int but1_pressed;
43 int but2_pressed;
44
45 // Flags for marking mode.
46 int config_called;
47 int config_mode_on;
48 int time_update_needed;
49
50 /**
51  * Initializes the program and main loop for checking
52  *  for configuration input and updating the LCD.
53  */
54 int main(void){
55   // Initialize variables.
56   init();
57   // Initialize configuration mode.
58   init_config();
59   // Do first display update.
```

```
60    display_update(_time);
61    while(1){
62      if(time_update_needed){
63        time_update_needed = 0;
64          display_update(_time);
65      }
66      if(config_called){
67        config_called =0;
68        init_config();
69      }
70    }
71  }
72
73  /**
74   * Start the configuration mode.
75   *  This mode is only for setting the alarm or clock.
76   */
77  void init_config(void){
78    // -1 is quit, 0 is alarm , 1 is clock.
79    int choice = CONFIG_MODE_ALARM;
80    static char *choice_string = CM_ALARM_STRING;
81    config_mode_on = 1;
82    display_line(CM_STRING,choice_string);
83    while(1){
84      if(read_and_clear(&but2_pressed)){
85        //Configure the selected config mode.
86        switch(choice){
87          case CONFIG_MODE_ALARM:
88            LCDErase();
89            init_time(_alarm, SM_ALARM_STRING);
90            display_line(CM_STRING,choice_string);
91            break;
92          case CONFIG_MODE_CLOCK:
93            LCDErase();
94            init_time(_time, SM_CLOCK_STRING);
95            T0CONbits.TMR0ON = 1;
96            display_line(CM_STRING,choice_string);
97            break;
98          default:
99            LCDErase();
```

```
100             config_mode_on = 0;
101             return;
102         }
103       }
104       if(read_and_clear(&but1_pressed)){
105         //Cycle trough the config modes.
106         switch(choice){
107           //For the alarm.
108           case CONFIG_MODE_QUIT:
109             LCDErase();
110             choice = CONFIG_MODE_ALARM;
111             choice_string = CM_ALARM_STRING;
112             display_line(CM_STRING,choice_string);
113             break;
114           //For the clock.
115           case CONFIG_MODE_ALARM:
116             LCDErase();
117             choice = CONFIG_MODE_CLOCK;
118             choice_string = CM_CLOCK_STRING;
119             display_line(CM_STRING,choice_string);
120             break;
121           //For quiting.
122           case CONFIG_MODE_CLOCK:
123             LCDErase();
124             choice =CONFIG_MODE_QUIT;
125             choice_string = CM_QUIT_STRING;
126             display_line(CM_STRING,choice_string);
127             break;
128         }
129       }
130     }
131 }
132
133 /**
134  * Sets the given timer with what the user inputs.
135  */
136 void init_time(time t, char *mode){
137   int h, m, s;
138   h = get_input(24, HOURS,   mode, &but1_pressed, &
        but2_pressed);
```

```
139   m = get_input(60, MINUTES, mode, &but1_pressed, &
          but2_pressed);
140   s = get_input(60, SECONDS, mode, &but1_pressed, &
          but2_pressed);
141   time_set(t,h,m,s);
142 }
143
144 /**
145  * Toggle the first (red) LED.
146  */
147 void toggle_second_led(void){
148   LED0_IO^=1;
149 }
150
151 /**
152  * Toggle the second and third (orange) LEDs.
153  */
154 void toggle_alarm_led(void){
155   LED1_IO^=1;
156   LED2_IO^=1;
157 }
158
159 /**
160  * Handles the high priority interupts.
161  *  Currently both buttons and ticks have high
          priority.
162  */
163 void highPriorityInterruptHandler (void) __interrupt
        (1){
164   // Button 2 causes an interrupt
165   if(INTCON3bits.INT1F == 1){
166   if(!config_mode_on){
167     config_called =1;
168   } else {
169     but2_pressed = 1;
170   }
171   if(BUTTON0_IO);
172     INTCON3bits.INT1F = 0;
173   }
174
```

```
175   // Button 1 causes an interrupt
176   if(INTCON3bits.INT3F  == 1){
177     but1_pressed = 1;
178     if(BUTTON1_IO);
179     INTCON3bits.INT3F = 0;
180   }
181
182   // Timer 0 causes an interrupt
183   if(INTCONbits.TMR0IF == 1) {
184     overflow_counter++;
185     if(overflow_counter == OVERFLOW_CYCLES/2){
186       toggle_second_led();
187     }else if(overflow_counter == OVERFLOW_CYCLES){
188       if(time_equals(_alarm,_time)){
189         alarm_going_off = 1;
190       }
191       if(alarm_going_off){
192         alarm_counter++;
193         toggle_alarm_led();
194         if(alarm_counter==30){
195           alarm_going_off =0;
196           alarm_counter = 0;
197         }
198       }
199       overflow_counter = 0;
200       toggle_second_led();
201       add_second(_time);
202       if(!config_called && !config_mode_on){
203         time_update_needed = 1;
204       }
205     }
206     INTCONbits.TMR0IF = 0;
207   }
208 }
209
210 /**
211  * Inintializes all kinds of settings.
212  */
213 void init(void){
214   // Initialize LCD
```

```
215   LCDInit();
216
217   // Initialize time
218   _time = time_create();
219   _alarm = time_create();
220
221   // Enable buttons
222   BUTTON0_TRIS = 1;
223   BUTTON1_TRIS = 1;
224
225   // Enable interrupts
226   INTCONbits.GIE = 1;
227   INTCONbits.PEIE = 1;
228   RCONbits.IPEN = 1;
229
230   // Disable timer
231   T0CONbits.TMR0ON = 0;
232
233   // Empty timer: high before low (!)
234   TMR0H = 0x00000000;
235   TMR0L = 0x00000000;
236
237   // Enable 8-bit operation
238   T0CONbits.T08BIT = 1;
239
240   // Use clock as clock source
241   T0CONbits.T0CS = 0;
242
243   // Unassign prescaler
244   T0CONbits.PSA = 1;
245
246   // Enable timer and interrupts
247   INTCONbits.TMR0IE = 1;
248
249   // Enable button interrupts
250   INTCON3bits.INT1IE = 1;
251   INTCON3bits.INT3IE = 1;
252
253   // Enable leds
254   LED0_TRIS = 0;
```

```
255   LED1_TRIS = 0;
256   LED2_TRIS = 0;
257   LED3_TRIS = 0;
258
259   // Disable all LED but backlight
260   LED0_IO = 0;
261   LED1_IO = 0;
262   LED2_IO = 0;
263   LED3_IO = 1;
264
265   // INITIALIZE OUR OWN VARIABLES.
266   // State indicators
267   alarm_going_off = 0;
268
269   // Counters
270   alarm_counter = 0;
271   overflow_counter = 0;
272
273   // Dummy button registers
274   but1_pressed = 0;
275   but2_pressed = 0;
276
277   // FLAGS FOR MARKING MODE.
278   config_called = 0;
279   config_mode_on = 0;
280   time_update_needed =0;
281 }
```

Listing 3: clockio header file

```
1  #ifndef __CLOCKIO_H_
2  #define __CLOCKIO_H_
3
4  #define __18F97J60
5  #define __SDCC__
6
7  // Defines for easy use of the LCD.
8  #define START_FIRST_LINE 0
9  #define START_SECOND_LINE 16
10
```

```
11 // INCLUDES
12 #include <stdlib.h>
13 #include <stdio.h>
14
15 #include "../Include/HardwareProfile.h"
16 #include "../Include/LCDBlocking.h"
17
18 #include "time.h"
19
20 void display_string(BYTE pos, char* text);
21 void display_update(time t);
22 void display_line(char *top, char *bottom);
23 int get_input(int maxvalue, char *text, char *mode,
      int * btn_next, int *btn_confrm);
24 char* to_double_digits(int value);
25 int read_and_clear(int *variable);
26
27 #endif
```

Listing 4: clockio body file

```
1  #include "clockio.h"
2
3  /**
4   * Displays the given string at the given position on
       the LCD.
5   */
6  void display_string(BYTE pos, char* text){
7    BYTE        l = strlen(text);
8    BYTE      max = 32-pos;
9    char       *d = (char*)&LCDText[pos];
10   const char *s = text;
11   size_t      n = (l<max)?l:max;
12   if (n != 0)
13     while (n-- != 0)*d++ = *s++;
14   LCDUpdate();
15 }
16
17 /**
18  * Updates the display and prints the current time.
```

```
19   */
20  void display_update(time t){
21    char display_line[32];
22    time_print(t, display_line);
23    display_string(0, display_line);
24  }
25
26  /**
27   * Display strings on first and second line of LCD
        display.
28   */
29  void display_line(char *top, char *bottom){
30    display_string(START_FIRST_LINE, top);
31    display_string(START_SECOND_LINE, bottom);
32  }
33
34  /**
35   * Gets the desired value for the given setting.
36   */
37  int get_input(int maxvalue, char *text, char *mode,
       int * btn_next, int *btn_confrm){
38    BYTE length = strlen(text);
39    int value = 0;
40    display_line(mode, text);
41    while(1){
42      if(read_and_clear(btn_confrm)){
43        LCDErase();
44        return value;
45      }
46      if(read_and_clear(btn_next)){
47        value = (++value)%maxvalue;
48      }
49      display_string(START_SECOND_LINE + length + 1,
          to_double_digits(value));
50    }
51  }
52
53  /**
54   * 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);
```

```
19 int time_equals(time t1, time t2);
20
21 #endif
```

Listing 6: time body file

```
1  #include "time.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <malloc.h>
6
7  unsigned char _MALLOC_SPEC heap[128];
8
9  struct time_struct {
10   int hours;
11   int minutes;
12   int seconds;
13 };
14
15 time time_create(){
16     time t = (time)malloc(sizeof (struct time_struct))
           ;
17     time_set(t,0,0,0);
18     return t;
19 }
20
21 void time_set(time t, int hours, int minutes, int
       seconds){
22   set_hours(t,hours);
23   set_minutes(t,minutes);
24   set_seconds(t,seconds);
25 }
26
27 int set_hours(time t, int value){
28   int overflow = value/24;
29   t->hours = value%24;
30   return overflow;
31 }
32
```

```
33 int set_minutes(time t, int value){
34   int overflow = value/60;
35   t->minutes = value%60;
36   return overflow;
37 }
38
39 int set_seconds(time t, int value){
40   int overflow = value/60;
41   t->seconds = value % 60;
42   return overflow;
43 }
44
45 void add_second(time t){
46   if(set_seconds(t,t->seconds + 1) != 0)
47     add_minute(t);
48 }
49
50 void add_minute(time t){
51   if(set_minutes(t,t->minutes + 1) != 0)
52     add_hour(t);
53 }
54
55 void add_hour(time t){
56   set_hours(t,t->hours + 1);
57 }
58
59 void time_print(time t, char* str){
60   sprintf(str, "%02d:%02d:%02d", t->hours, t->minutes,
         t->seconds);
61 }
62
63 int time_equals(time t1, time t2){
64   if(t1->seconds != t2->seconds)
65     return 0;
66   if(t1->minutes != t2->minutes)
67     return 0;
68   if(t1->hours != t2->hours)
69     return 0;
70   return 1;
71 }
```