Alarm Clock
Embedded Systems Project Report

Monday 10:00 lab

Jakub Pawlak
`234767@edu.p.lodz.pl`

Artur Pietrzak
`234768@edu.p.lodz.pl`

Juliusz Szymajda
`234769@edu.p.lodz.pl`

June 3, 2022

Devices used:

Eduboard LPC2148 v1.0

Interfaces used:

GPIO, I^{2}C, SPI

Devices used:

1. LCD display
2. RTC
3. Button
4. Joystick
5. Buzzer
6. Timer
7. EEPROM

Contents

1 Project Description **4**

1.1 General description 4

1.2 Setting the alarm 4

1.3 Changing the current time 4

1.4 Turning off the alarm 4

2 Peripherals and interface configuration **5**

2.1 LCD Display 5

2.2 GPIO 6

2.3 I^{2}C 6

2.4 SPI 6

2.5 Debounce Timer 6

2.6 Joystick 6

2.7 RTC 7

3 Failure Mode and Effect Analysis **10**

References **10**

Code Listings

1 LCD setup function 5

1 Project Description

1.1 General description

The project is an digital clock with the ability to set an alarm for a given hour. The device displays current time on the LCD display. Using button and joystick it is possible to set the current time, or set an alarm. If the alarm is set and the alarm time is the current time, the buzzer emits a sound until the alarm is turned off.

1.2 Setting the alarm

The alarm setting mode it entered by pressing down the joystick center push-down switch. When in alarm setting mode, the user can change the alarm time by selecting the digit with left/right movement, and selecting the value with up/down movement. After the correct time is selected, alarm is set by pressing the button.

When alarm is turned on, it can be turned off by short press of the button.

1.3 Changing the current time

The time setting mode is entered by holding down the button. Then, the time is set with the joystick in the similar way, the alarm is set. After setting the time it is set by short press of the button.

1.4 Turning off the alarm

When the alarm clock starts emmiting sound, the user can turn it off by pressing the button.

2 Peripherals and interface configuration

2.1 LCD Display

```c
void DisplayInit(void) {
    IODIR1 |= (LCD_DATA | LCD_E | LCD_RS);
    IOCLR1  = (LCD_DATA | LCD_E | LCD_RS);

    IODIR0 |= LCD_RW;
    IOCLR0  = LCD_RW;

    IODIR0 |= LCD_BACKLIGHT;
    IOCLR0  = LCD_BACKLIGHT;

    LcdCommand(0x30);
    delay2ms();
    LcdCommand(0x30);
    delay37us();
    LcdCommand(0x30);
    delay37us();

    LcdCommand(0x38); // set 8-bit, 2 line mode
    delay37us();

    LcdCommand(0x08); // display off
    delay37us();

    clearDisplay();

    LcdCommand(0x06); // cursor direction - increment, no shift
    delay2ms();

    LcdCommand(0x0c); // display on, cursor off
    delay2ms();

    LcdCommand(0x02); // cursor to home position
    delay2ms();
}
```

Listing 1: LCD setup function

2.2 GPIO

2.3 I^{2}C

2.4 SPI

2.5 Debounce Timer

2.6 Joystick

Listing 1: Joystick get input function

```c
char getJoyInput(void) {
    if (((IOPIN0 & KEYPIN_UP) == 0)) {
        return 'u';
    }
    if (((IOPIN0 & KEYPIN_DOWN) == 0)) {
        return 'd';
    }
    if (((IOPIN0 & KEYPIN_RIGHT) == 0)) {
        return 'r';
    }
    if (((IOPIN0 & KEYPIN_LEFT) == 0)) {
        return 'l';
    }
    if (((IOPIN0 & KEYPIN_CENTER) == 0)) {
        return 'c';
    }
    return 0;
}
```

The function checks if the joystick is utilized and return char corresponding to the current position of the joystick.

Listing 2: Await joystick input function

```c
char waitForJoyInput(void){
    while (1) {
        char result = getJoyInput();
        if (result != 0)
            return result;
        delay37us();
    }
}
```

This function runs in an infinite loop, which ends when the result of the JoyInputFunction is not 0 (when there is some input from the joystick).

Listing 3: Get button input function

```c
uint8 isButtonPressed(void) {
    if ((IOPIN0 & PIN_BUTTON) == 0) {
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

This function check wether the button on the board is pressed.

2.7 RTC

Listing 4: RTC initialization

```c
    RTC_Time setTime, alarmTime, currentTime;


    CCR = 0x01;
```

First, the structs for time to set, alarm time and current time are created. The RTC interrupts are disabled by writing 0x0 to Interrupt Location Register (ILR). The clocks tick counter is also reset by writing 0x02 to the Clock Control Register (CCR), and disabled by writing 0x0. Next, the Counter Increment Interrupt (CIIR), which is responsible for incrementing seconds, is disabled.

Next, the prescaler is configured. Prescaler is a circuit used to reduce a high frequency electrical signal (for example from RTC) to a lower frequency by integer division. After calculating the value of a prescaler integer:

$$(PCLK/32768)\text{–}1$$

and value of prescaler fraction:

$$PCLK\text{–}((PREINT + 1) * 32768)$$

Those values are written to Prescaler Integer Register (PREINT) and Prescaler Fraction Register (PREFRAC) registers.

Finally, the time is read from EEPROM, written into setTime struct and set using RTC_SetTime function. At the end of initialization 0x01 is writtent to Clock Control Register (CCR), which enables the timer counters.

Listing 5: RTC struct

```c
typedef struct
{
    uint8 seconds;
    uint8 minutes;
    uint8 hours;
}
```

The struct consists of three 8 bit unsigned integer numbers, each one for seconds, minutes and hours fields.

Listing 6: Setting time in the register function

```c
void RTC_SetTime(RTC_Time Time)
{
    SEC = Time.seconds;
    MIN = Time.minutes;
    HOUR = Time.hours;
}
```

The function uses data stored in the parsed struct and writes it to counters from Time Register Group. SEC and MIN are 6 bit, and the HOUR counter is 5 bit.

Listing 7: Setting alarm time in the register function

```c
void RTC_Set_AlarmTime(RTC_Time AlarmTime)
{
    ALSEC = AlarmTime.seconds;
    ALMIN = AlarmTime.minutes;
    ALHOUR = AlarmTime.hours;
}
```

The function uses data stored in the parsed struct and writes it to counters from Alarm Register Group. Similarly to the Time Register Group, ALSEC and ALMIN are 6 bit, and the ALHOUR counter is 5 bit.

Listing 8: Getting time from the register function

```c
RTC_Time RTC_GetTime(void)
{
    RTC_Time time;

    time.seconds = SEC;
    time.minutes = MIN;
    time.hours = HOUR;

    return time;
}
```

The function takes data from SEC, MIN and HOUR registers stored in Time Register Group, assigns it to the newly created RTC_Time struct and returns it.

Listing 9: Getting alarm time from the register function

```
RTC_Time RTC_GetAlarmTime(void)
{
    RTC_Time AlarmTime;

    AlarmTime.seconds = ALSEC;
    AlarmTime.minutes = ALMIN;
    AlarmTime.hours = ALHOUR;

    return AlarmTime;
}
```

Analogously, to the RTC_GetTime function, this function takes data from ALSEC, ALMIN and ALHOUR registers stored in Alarm Register Group, assigns it to the newly created RTC_Time struct and returns it.

Listing 10: RTC_Time struct comparator function

```
bool RTC_Compare(RTC_Time t1, RTC_Time t2)
{
    return (t1.seconds == t2.seconds
        && t1.minutes == t2.minutes
        && t1.hours == t2.hours);
}
```

The function is used to compare two RTC_Time structs by comparing each field of the struct (seconds, minutes and hours).

3 Failure Mode and Effect Analysis

Component	**Severity**
Microcontroller	Critical
Power Supply	Critical[1]
RTC	Critical
LCD Display	High
Speaker	High
Button	High
Joystick	High

Table 1: Severity of component's failure

References

[1] Embedded Artists. *LPC2148 Education Board User's Guide*, 2006. EA2-USG-0601 v1.2 Rev B.

[2] Hitachi. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)*. ADE-207-272(Z) '99.9 Rev. 0.0.

[1]Long-term power supply failures are of critical severity, but in case of short pause in power delivery, the system is able to recover using the RTC and the data stored in EEPROM