An Arduino alarm clock

Koen Bolhuis

1 Problem description

Our project idea is to make a clock which can set an alarm by using Arduino, LCD keypad shield and a piezo buzzer.

2 Problem analysis

In order to specify this project, a program is required to show a current time and an alarm time on the LCD pad. In this requirement, the most important idea is how to build up program which displays the current time increased a second after a second. Besides, it is necessary to build up to set the alarm time, change the current time and turn off the buzzer ringing by using buttons. It means that we have to assign each tasks to each buttons. Since 5 buttons, which is named "Up", "Down", "Right", "Left" and "Select" on LCD keypad, can be used, we assigned "Right" button to change selected time which is the current time or the alarm time. "Left" button is to change time units like a minute or an hour. "Up" and "Down" buttons are assigned to increase or decrease time with selected time unit in assigned clock. For example, if selected time unit was an hour then "Up" button was pressed in the current time, 1 hour would be increased in the current time. It would be better to solve a minor problem that we cannot easily recognize what option is selected in a current state. So, we determined to express ^ symbol in the second line on LCD to indicate what option is selected, for instance, whether hours or minutes and whether the current time or the alarm time. The last button, "Select", would perform to turn off the buzzer when alarm is activated.

3 Design

The hardware side of this project consists of three parts:

- The Arduino itself;
- A DFROBOT LCD keypad shield, which is placed on top of the Arduino;
- A piezo buzzer, connected directly to the ground (GND) of the Arduino, and, via a 10K ohm resistor, to digital pin #2.

The figures below show the complete setup:

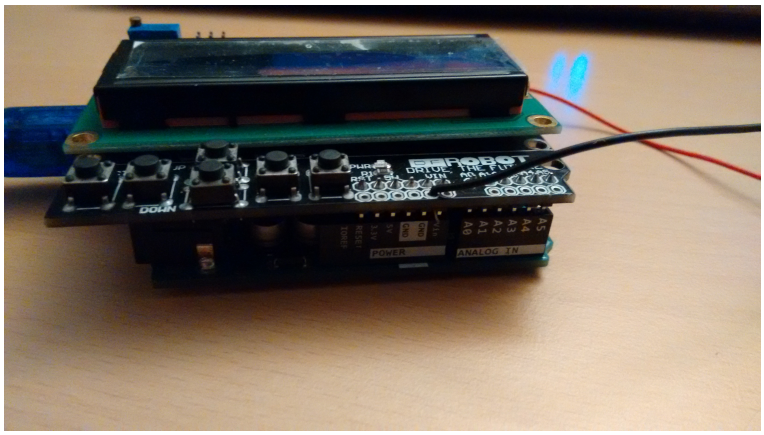

Figure 1: Side view of the hardware setup

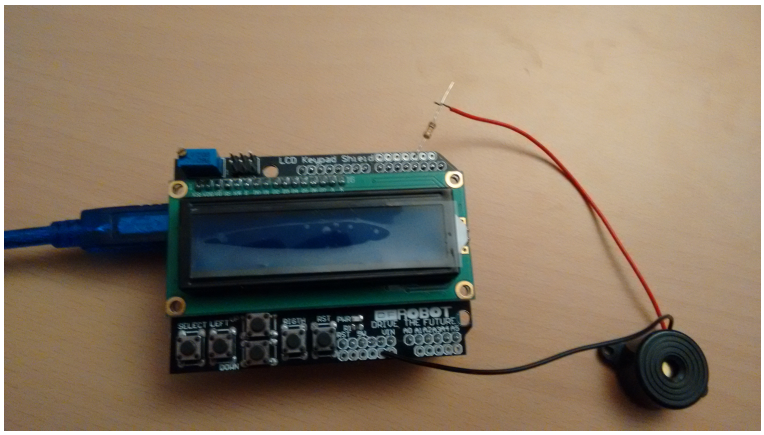

Figure 2: Top-down view of the setup

To control the LCD, a specialised Arduino library is available: `LiquidCrystal.h`. This library allows us to create an object of type `LiquidCrystal` to interface with the LCD. The constructor of this object takes the pins used to communicate with the LCD as arguments; in our case, the LCD on the DFROBOT LCD shield uses pins 8, 9, 4, 5, 6 and 7. We create the LCD object and store it in a global variable as follows: `LiquidCrystal lcd(8, 9, 4, 5, 6, 7);`

The keypad provides all button input on a single analog input pin, using different values for the different buttons on the Arduino shield. The pin used by the shield is Analog Input 0, which translates to pin number 0 in code. To read from the pin, we first initialise it as being an input pin using `pinMode(0, INPUT)`, which is called inside the `setup()` Arduino function. After that, we can read the button values from the pin using `analogRead(pin)`. We use a helper function to translate these read values into actual button numbers.

Buzzer output is provided on digital pin 2, an arbitrary (free) digital pin which we set to use for output using `pinMode(2, OUTPUT)`. After that, `tone(pin, frequency)` and `noTone(pin)` allow us to turn the buzzer on and off respectively.

To make writing the program easier, we have decided to split it up into smaller units:

- `alarmclock.ino`: the main program, which contains the initialisation function (`setup()`) and the main loop (`loop()`).
- `constants.h`: a file in which several useful constants are defined, such as the pin numbers for the button input and buzzer output, a mapping of button values to button identifiers,

and some other named constants.

- `display.h`: creates the LCD object and contains some display-related functions, such as a function to print a clock to the LCD screen.
- `functions.h`: other miscellaneous functions, including a function that translates button values to button identifiers.

The main program can reside in two states: `STATE_CLOCK` and `STATE_ALARM`, meaning "viewing the current time and alarm time" and "alarm is going off" respectively.

Regardless of the state, a timer increments the current time every second. On the main screen, this current time, as well as the alarm time, are printed, and the program listens for button input. The time units are changed accordingly. Once the current time becomes equal to the alarm time, which is checked using a simple if-statement, the state is set to the alarm state.

When the alarm goes off, a timer is used to turn the buzzer on and off periodically. The program also listens for a button press, and if the "Select" button is pressed, the state is returned to the clock state.

The resulting program, running on the Arduino, is shown in the figures below:

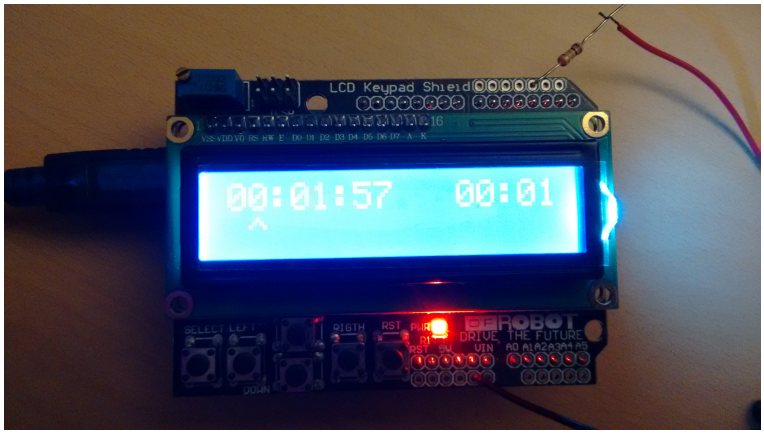

Figure 3: The display showing the current time, alarm time and selection arrow

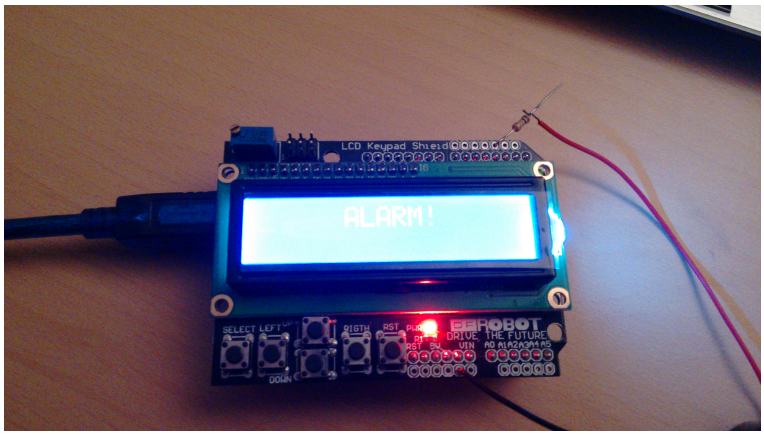

Figure 4: The alarm going off

For a more detailed description of the program design, the (well-commented) source code is available below.

4 Program code

alarmclock.ino

```
/*
 * Arduino Alarm Clock
 * Version 1.2
 * Koen Bolhuis & Siheon Lee, 2016
 *
 * alarmclock.ino:
 * Main program.
 */

#include "constants.h"
#include "display.h"
#include "functions.h"

/** Initialise global variables **/

// Used for timing and updating the current time
unsigned long timer, timerPrev;

// Used for timing the alarm buzzer
unsigned long alarmTimer, alarmTimerPrev;

/** Main program **/

void setup() {
  // Initialise a 16x2 character LCD display
  lcd.begin(16, 2);
  lcd.clear();

  // Designate the appropriate pins with the correct I/O mode
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_BUTTONS, INPUT);

  // Initialise the timer values
  timerPrev = millis();
  alarmTimerPrev = millis();
}

void loop() {
  /** Initialise static variables **/

  // Used to detect button presses
  static int button, buttonCheck, buttonPrev;

  // The time values for the two on-screen clocks
  static int currentTime[3] = {0, 0, 55};
  static int alarmTime[2] = {0, 1};

  /*
   * The current program state: STATE_CLOCK or STATE_ALARM,
   * depending on whether the alarm is buzzing.
   */
  static int state = STATE_CLOCK;

  // The currently selected time unit and clock that are to be changed
```

```
55    static int selectedUnit = UNIT_H;
56    static int selectedClock = CLOCK_CURRENT;
57
58    static bool buzzer = false;
59
60    /** Main loop code **/
61
62    // Update the current time
63    timer = millis();
64    if (timer - timerPrev >= SECOND) {
65      timerPrev = timer;
66
67      updateCurrentTime(currentTime);
68    }
69
70    if (state == STATE_CLOCK) {
71      // Check for input
72      button = getButton();
73      if (button != buttonPrev && button != BUTTON_NONE) {
74        // Wait a bit to debounce the button
75        delay(50);
76
77        // Verify the button
78        buttonCheck = getButton();
79        if (buttonCheck == button && buttonCheck != BUTTON_NONE) {
80          int timeDelta = 0;
81
82          switch (buttonCheck) {
83            case BUTTON_RIGHT:
84              selectedClock = 1 - selectedClock;
85              break;
86            case BUTTON_LEFT:
87              selectedUnit = 1 - selectedUnit;
88              break;
89            case BUTTON_UP:
90              timeDelta = 1;
91              break;
92            case BUTTON_DOWN:
93              timeDelta = -1;
94              break;
95          }
96
97          // Update selection arrow if left or right was pressed
98          clearLine(1);
99          printSelection(selectedClock, selectedUnit);
100
101         // Update time if up or down was pressed
102         if (timeDelta != 0) {
103           switch (selectedClock){
104             case CLOCK_CURRENT:
105               changeTime(currentTime, selectedUnit, timeDelta);
106               break;
107             case CLOCK_ALARM:
108               changeTime(alarmTime, selectedUnit, timeDelta);
109               break;
110           }
111         }
112       }
```

```
    }
    buttonPrev = button;

    // Print time
    lcd.setCursor(0, 0);
    printTime(currentTime, true);
    lcd.setCursor(11, 0);
    printTime(alarmTime, false);

    printSelection(selectedClock, selectedUnit);

    // Check if alarm should go off
    if (currentTime[UNIT_H] == alarmTime[UNIT_H]
        && currentTime[UNIT_M] == alarmTime[UNIT_M]
        && currentTime[UNIT_S] == 0) {
      state = STATE_ALARM;
    }
  }
  else if (state == STATE_ALARM) {
    // Turn buzzer on or off periodically
    alarmTimer = millis();
    if (alarmTimer - alarmTimerPrev >= BUZZ_DURATION) {
      alarmTimerPrev = alarmTimer;

      buzzer = !buzzer;

      lcd.clear();

      if (buzzer) {
        tone(PIN_BUZZER, BUZZER_FREQ);

        lcd.setCursor(5, 0);
        lcd.print("ALARM!");
      }
      else {
        noTone(PIN_BUZZER);
      }
    }

    // End the alarm if the select button is pressed
    button = getButton();
    if (button == BUTTON_SELECT) {
      // Wait a bit to debounce the button
      delay(50);

      // Verify the button
      buttonCheck = getButton();
      if (buttonCheck == button && buttonCheck == BUTTON_SELECT) {
        buzzer = false;
        noTone(PIN_BUZZER);
        lcd.clear();

        state = STATE_CLOCK;
      }
    }
  }
}
```

constants.h

```
/*
 * constants.h:
 * Various named constants.
 */

#ifndef __CONSTANTS_H
#define __CONSTANTS_H

const int PIN_BUZZER = 2;
const int PIN_BUTTONS = 0;

const int BUTTON_COUNT = 5;

/*
 * Button analog input values
 * These have been taken from http://www.okaphone.nl/files/LCD1602KEY%20V1.1.pdf
 */
const int BUTTON_VALUES[BUTTON_COUNT] = {
  30, // Right
  150, // Up
  350, // Down
  535, // Left
  760 // Select
};

// Button identifiers
const int BUTTON_NONE = -1;
const int BUTTON_RIGHT = 0;
const int BUTTON_UP = 1;
const int BUTTON_DOWN = 2;
const int BUTTON_LEFT = 3;
const int BUTTON_SELECT = 4;

// 1000 milliseconds in one second
const int SECOND = 1000;

// Buzzer sound frequency & duration
int BUZZER_FREQ = 1000;
int BUZZ_DURATION = SECOND/4;

// State identifiers
const int STATE_CLOCK = 0;
const int STATE_ALARM = 1;

// Identifiers for the different units
// These also work as an index into an array storing a time
const int UNIT_H = 0;
const int UNIT_M = 1;
const int UNIT_S = 2;

// Identifiers for the two clocks (current time and alarm time)
const int CLOCK_CURRENT = 0;
const int CLOCK_ALARM = 1;

#endif // __CONSTANTS_H
```

display.h

```
/*
 * display.h:
 * Defines various functions related to printing to the LCD display.
 * Also declares a global variable lcd that is used to access the display directly.
 */

#ifndef __DISPLAY_H
#define __DISPLAY_H

#include <LiquidCrystal.h>
#include "constants.h"

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void clearLine(int line) {
  // Print 16 spaces to clear the line
  lcd.setCursor(0, line);
  lcd.print("                ");
}

void printPadded(int n) {
  if (n < 10) {
    lcd.print("0");
  }
  lcd.print(n);
}

void printTime(int *timeArray, bool showSeconds) {
  printPadded(timeArray[UNIT_H]);
  lcd.print(":");
  printPadded(timeArray[UNIT_M]);

  if (showSeconds) {
    lcd.print(":");
    printPadded(timeArray[UNIT_S]);
  }
}

void printSelection(int selectedClock, int selectedUnit) {
  int pos, i;

  lcd.setCursor(11*selectedClock + 3*selectedUnit + 1, 1);
  lcd.print("^");
}

#endif
```

functions.h

```
/*
 * functions.h:
 * Several miscellaneous functions.
 */

#ifndef __FUNCTIONS_H
#define __FUNCTIONS_H

```

```
#include "constants.h"

/*
 * Reads an analog button value from analog pin A0,
 * and returns the corresponding button identifier.
 */
int getButton() {
  int value, i;

  value = analogRead(PIN_BUTTONS);

  for (i = 0; i < BUTTON_COUNT; i++) {
    if (value < BUTTON_VALUES[i]) {
      return i;
    }
  }

  return BUTTON_NONE;
}

void updateCurrentTime(int *timeArray) {
  timeArray[UNIT_S]++;
  if (timeArray[UNIT_S] >= 60) {
    timeArray[UNIT_S] = 0;
    timeArray[UNIT_M]++;
  }
  if (timeArray[UNIT_M] >= 60) {
    timeArray[UNIT_M] = 0;
    timeArray[UNIT_H]++;
  }
  if (timeArray[UNIT_H] >= 24) {
    timeArray[UNIT_H] = 0;
  }
}

void changeTime(int *timeArray, int unit, int change) {
  timeArray[unit] += change;

  switch (unit) {
    case UNIT_H:
      if (timeArray[UNIT_H] < 0) {
        timeArray[UNIT_H] = 23;
      }
      if (timeArray[UNIT_H] > 23) {
        timeArray[UNIT_H] = 0;
      }
      break;
    case UNIT_M:
      if (timeArray[UNIT_M] < 0) {
        timeArray[UNIT_M] = 59;
      }
      else if (timeArray[UNIT_M] > 59) {
        timeArray[UNIT_M] = 0;
      }
      break;
  }
}

```

```
67 #endif // __FUNCTIONS_H
```

5 Test results

Since the program cannot be tested in a "conventional" way, we chose to do some hands-on testing.

- First of all, compiling the program gives the following result:

Compilation output

```
Sketch uses 4,228 bytes (13%) of program storage space. Maximum is 32,256 bytes.
Global variables use 133 bytes (6%) of dynamic memory, leaving  1,915 bytes for
    local variables. Maximum is 2,048 bytes.
```

- On the Arduino, pressing the "Left" button causes the selection arrow to move between the selected time units (hours/minutes).
- Pressing the "Right" button moves the selection arrow from the current time to the alarm time.
- Pressing "Up" or "Down" correctly increments/decrements the currently selected time unit of the selected clock:
  - Incrementing or decrementing the hours past the upper limit (23) or lower limit (00), the number wraps around.
  - Incrementing or decrementing minutes past the upper limit (59) or lower limit (00), the number wraps around as well.
- The "seconds" unit of the current time automatically increments every second; when it reaches 60, it is set back to 00, and the "minutes" number increases. When that in turn reaches 60, the "hours" unit is incremented.
- When the current time becomes equal to the alarm time (whether it happens automatically, or as the result of user input), the alarm goes off, displaying "ALARM!" on the LCD display, as well as outputting a buzzing sound at the predetermined interval and frequency, on a piezo buzzer connected to the Arduino.
- When the alarm is active, pressing the "Select" button turns the alarm off. The current time will have continued to update in the background, and the correct current time will be shown.

6 Evaluation

Overall, the alarm clock is well activated in our frame, however, it is quite limited program because one thing we does not consider is a date. In our project, the date does not exist, in other words, alarm could be set and activated only in 24 hours. You could not make an alarm in any specific date and see which date is today.

Due to solve this problem, we should consider how the date could be expressed on the LCD display and how it would be controlled by buttons. Moreover, the program would be more complicated because a leap day in February would make us to calculate much more complexly. Also, we have to redesign the LCD screen because a size of the LCD (16*2) is very limited while the length of date is much longer. Both the current time and the alarm time can no longer be displayed at the same time.

Secondly, our program can save only one alarm time. It means that you cannot make two or more alarms. In order to solve it, we need to build up way to make and save a new alarm while other alarms exist.

Lastly, we could consider making an alarm be able to set a regular alarm. For example, We can choose a daily alarm or weekly alarm in a certain date.

However, these are all extra features that we *could* implement, but chose not to, to keep the program simple. To summarise, we think the program does what it needs to do, and nothing more.