Alarm Clock For Heavy Sleepers

Introduction

Description: The project consists of an alarm clock that, on ringing, will diplay a series of math questions and the user will have to correcty answer three in a row for the alarm to stop. Alternatively, if math proves to be too difficult at the time, the user can stop the alarm by yelling at it with sufficient volume.

Purpose: The purpose of the device is to make sure that the user is actually awake when turning off the alarm.

Utility: It's inteded to be usefull for people who frequently turn off their morning alarm and imediately fall back asleep.

General description

Normal state Device displays current time. The microcontroller reads the time from the RTC module and displays it on the LCD. From this state the user can press the 1 key to open settings.

Settings menu Time is replaces by settings menu. The LCD will show on the top row the name of a setting (ex: “set alarm”) and on the bottom row the options “1back 2ok 3next”. If the user presses 1 the clock goes back to normal state, if they press 2 the bottom row is replaced with input options for the setting in question” and if they press 3 the top row will display next setting. The possible settings are the following:

- set alarm - the user can input the time for the alarm to ring or turn off alarm (a LED will be lit if alarm is set)
- ragequit mode - on/off setting that enables or disables the function to turn off alarm by yelling (a LED will be lit if mode is enabled)
- volume - allows user to adjust alarm volume
- brightness - allows user to adjust display brightness

Alarm time When current time reaches the value set for the alarm, the microcontroller will send the alarm tone signal to the buzzer for it to ring. On the display, the current time will be replaced by a math question generated by the microcontroller. The user will be able to enter answer trough keypad and the microcontroller will keep track of the number of consecutive correct anwers. If this number reaches 3, the alarm will turn off and the clock will go back to normal state.

At the same time, if ragequit mode is enabled, the microcontroller will check the inputs from the noise

sensor and turn the alarm off if a loud enough sound was detected.

Hardware Design

No	Component	Role
1	atmega328p-xPlained Mini	microcontroller
2	16×2 LCD	displays time + math questions + settings menu
3	4×4 Keypad	user input for settings and question answers
4	Noise Sensor	detects if sound exceeds a treshold (user yelled loud enough)
5	Passive Buzzer	produces alarm sound
6	RTC Module	keeps time
7	LED (x2)	display certain settings (1 for showing if an alarm time is set, 1 for showing if ragequiting mode is enabled)
8	220ohmRezistors (x2)	creating the LED circuits
9	Breadboard	connecting other components
10	Dunport wires (various types)	connecting other components

Pin connections

The LCD display and the RTC module communicate with the board through its I2C bus, so I connected themto the designated I2C pins of the board.

The keypad, I initially thought communicates trough I2C as well as that is what is says on the product page. However, by reading the linked library I noticed that while it does use only 2 pins to send data, it doesn't use I2C. The pins used in the library's example aren't the one used for I2C and the device doesn't have an I2C address. As a result I updated the pin connections and connected the keypad to 2 GPIO pins instead of the I2C bus.

For the buzzer, I chose to use a passive buzzer in order to include PWM concepts and have a more customizable alarm tone. Because of that, I connected the it to a pin that supports PWM.

The noise sensor has strictly digital output so it could be connected to a GPIO pin.

The 2 LEDs are also connected to GPIO pins as I only need them to show whether certain settings are activated or deactivated.

No	Component	component pin	microcontroller pin
1	LCD	SDA	PC4
		SCL	PC5
		VCC	5V
		GND	GND
2	Keypad	SDO	PB0
		SCL	PB1
		VCC	5V
		GND	GND

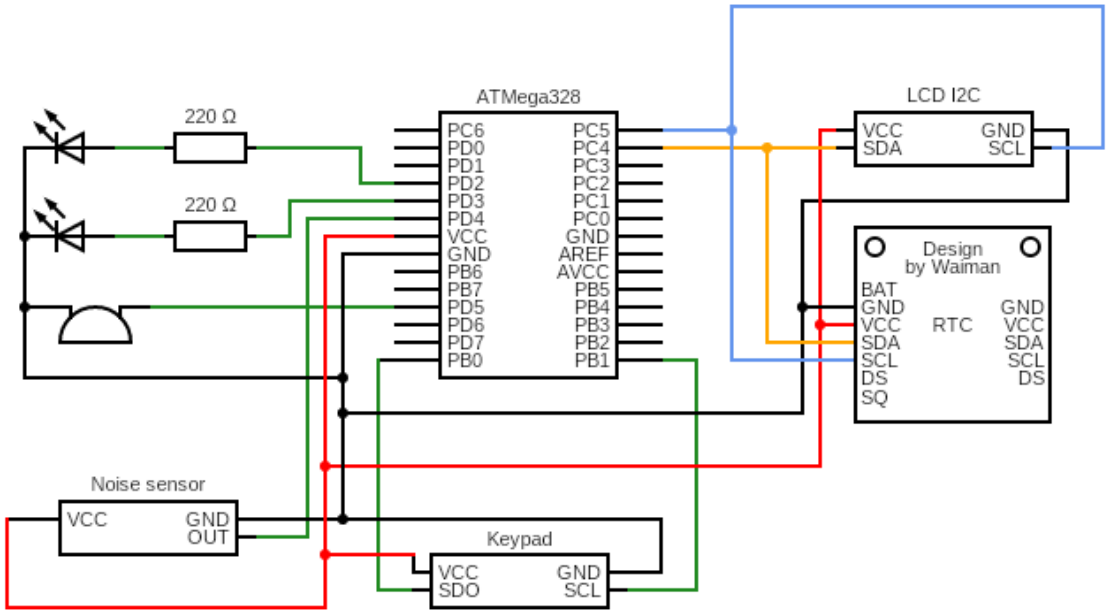
4	Noise Sensor	OUT	PD4
		VCC	5V
		GND	GND
5	Passive Buzzer	+	PD5
		-	GND
6	RTC Module	SDA	PC4
		SCL	PC5
		VCC	5V
		GND	GND
7	LED1	+	PD2
		-	GND
8	LED2	+	PD3
		-	GND

Electrical diagram

This is an electrical diagram showing the connections as described in the table above, with the addition of showing where resistors will be connected.

This is the color coding I used in making the diagram:

- in red are VCC connections
- in black are GND connections
- in blue are I2C SCL connections
- in orange are I2C SDA connections
- in green are other types of input/output



Keypad input showcase

This is a picture of only the keypad part of the device (i disconected the other components for better visibility). To showcase the fact that the microcontroller received user input from the keypad while not yet having connected the LCD display, I used the USART protocol to print the keypad input on the

serial monitor. This picture is taken after pressing the series of keys 13, 13, 14, 15, 16 on the keypad and it can be observed that the same series of keys was printed on the laptop screen.

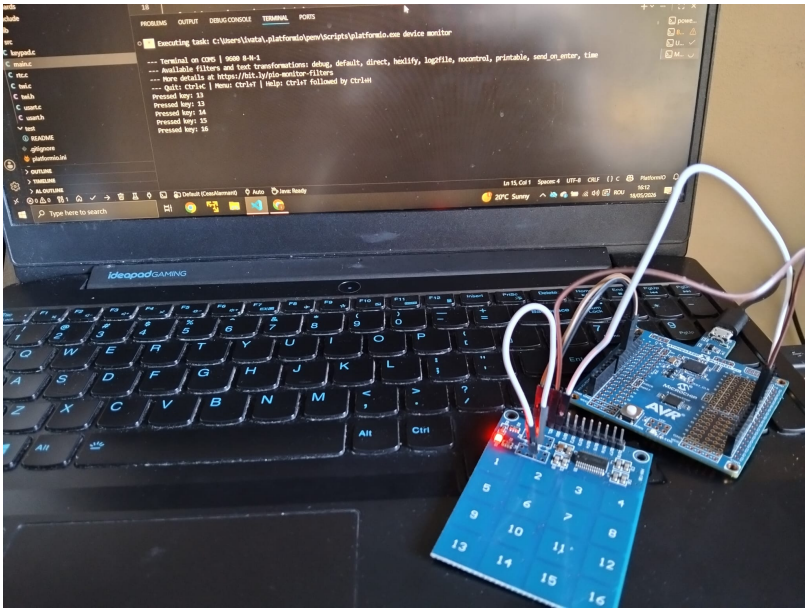
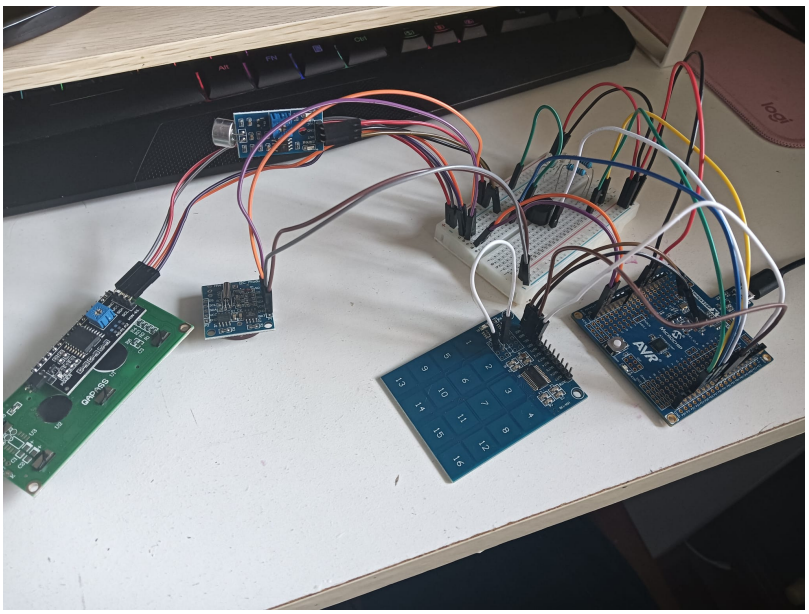


Image of the complete circuit



Software Design

I implemented this project in C, using platformIO.

The clock is implemented as a finite state machine, having the following states with distinct operating loops:

- normal state - the state in which it displays time, state in which it will be most of the time
- menu state - the state in which the time on the screen us replaced with the menu and the user can interract with the settings using the keypad

- alarm state - the state in which the buzzer is activated. In this state, the LCD displays questions and the noise sesor checks for yelling

Normal state

Printing the time. Once every second the microcontroller reads the current time from the RTC module in the format (hour, minute, second) and then prints the hour in the format (hour:minute).

With every reading the code also compares the current time with the time set for the alarm to ring (if an alarm time is set) and if the times equal, the clock goes into the alarm state.

In this state the device also listens for user input, specifically checking if the user has pressed the key 1 which is the button for enetering the menu state.

Menu state

In this state, the clock waits for user inputs and displays/changes the settings accordlingly.

On first entering the menu the top line of the LCD displays the first setting “set alarm” on the top row and “1back 2ok 3next” on the bottom row.

- if the user presses 1, the clock will go back into normal state
- if user presses 3, the top row will change to the next setting, “ragequit mode”. With this button, the user can cycle to all settings to find what they want to change
- if the user presses 2 the user will be able to change the setting displayed on the top row.

There are 2 types of settings in terms of the text displayed on the bottom row and the way the user inputs the desired value. For the settings with a limited number of possible values (such as “ragequit mode” where the only options are “on” and “off”), the bottom row will display key mappings similar to the previous ones and the user will input one of the displayed keys. For other settings (such as “set alarm” where the possible values are any valid combinations of minutes and hours), the user will have to manually input the value and the bottom row will display keys they have entered so far.

On changing a setting the clock goes back in normal mode.

Alarm state On entering this state, the device generates a question along with its answer and displays the question on the LCD. Then, it listens for user inputs and if they have entered the correct number with the keypad, the number of consecutive correct answers will be incremeted. When this number reaches three, the clock goes back into normal state.

The questions can be a calculation (using any of: +, -, *, /), an ecuation or a sequence that the user has to continue. The type of question as well as the exact values are chosen randomly every time a question is generated. In order to simulate randomness, when the clock is activated, it generates a random seed, based on the exact time it was activated.

Rezultate Obţinute

Care au fost rezultatele obţinute în urma realizării proiectului vostru.

Concluzii

Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.

Bibliografie/Resurse

- https://github.com/optimusdigital/ttp229-arduino/tree/master
- https://github.com/optimusdigital/DS1307

Export to PDF