ECE 3411 Final Project: Alarm Clock

Priyanshu Agrawal

5/7/2025

Description

In my project, I built a fully functional alarm clock with the AVR128DB48 Curiosity Nano board. The alarm clock uses a DS3231 real-time clock (RTC) to keep track of the time. It uses I2C to communicate with the RTC. The time and date are displayed on an LCD display, which also uses I2C communication. The displayed time shows hours, minutes and seconds. The displayed date shows the day of week, month, day and year. My alarm clock includes three buttons. These buttons are used by the user to snooze or turn off the alarm, in addition to navigating the settings menu, which is implemented with a state machine. The user is able to set one alarm which triggers at the same time each day. Snoozing the alarm delays it by 10 minutes. To make it easier for the user to set the time, date and alarm, the alarm clock has a potentiometer which is read with an ADC. This enables the user to turn the potentiometer dial to set these values within the corresponding menu screen. In order to play the alarm noise, I am using a buzzer with PWM.

Hardware Design

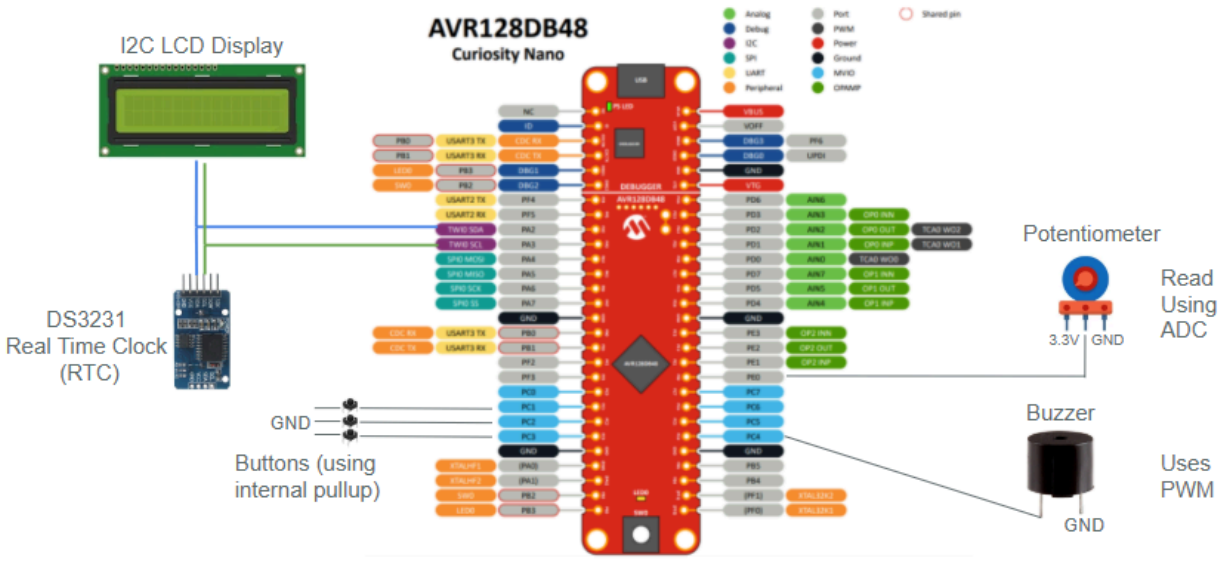

Figure 1. AVR128DB48 Pin Out and Wiring Diagram.

Part	Description
DFRobot LCD1602 Module	LCD display with I2C communication
DS3231 Real Time Clock	Real time clock with battery backup and I2C communication
Button x3	Buttons for user-input (read using GPIO)
Potentiometer	Dial for user-input (read using ADC)
Buzzer	Play alarm beeping noise (use PWM)

Table 1. List of Parts.

Software Design

To create my alarm clock, I used a task based design. At the start of the main function, the code performs all initial setup. This includes setting up the AVR clock, starting the timer, initializing UART (for debugging only), initializing I2C, initializing communication with DS3231, initializing communication with the LCD, configuring GPIO, configuring PWM, configuring ADC and enabling interrupts. My code uses a timer interrupt which triggers every 1 millisecond to increment four timer counters. In the main loop, there are four time-based tasks which are executed when the corresponding timer counter reaches the desired value. The tasks are as follows:

1. Real Time Clock Polling Task (every 200 ms)
    a. Reads the time from the RTC
    b. If the time has changed, update the displayed time and check if the alarm is triggered
2. Button Polling Task (every 20 ms)
    a. Reads the GPIO value for each of the three button pins
    b. For each button, progress the button debouncing state machine accordingly (see Figure 3)
    c. Based on the button presses, update the menu state or alarm state (see Table 2)
3. Potentiometer Polling Tasks (every 20 ms)
    a. Reads the ADC value of the potentiometer

b. Stores the value in a circular buffer of size 10
c. Computes the average value of all elements in the buffer
d. For menu states that involve setting part of the time and date, the potentiometer value will be scaled and the displayed time is updated
    i. Potentiometer value is scaled to 1-7 for day of the week, 1-12 for the month, 1-31 for day of the month (depending on the current month and leap year), 00-99 for year, 0-23 for hour, 0-59 for minute and 0-59 for second

4. Buzzer Task (every 400ms or 1000ms)
    a. If the alarm is in the beeping state, turn on buzzer with 1500 Hz PWM at 0.9 duty ratio for 1 second and off for 400 milliseconds repeatedly

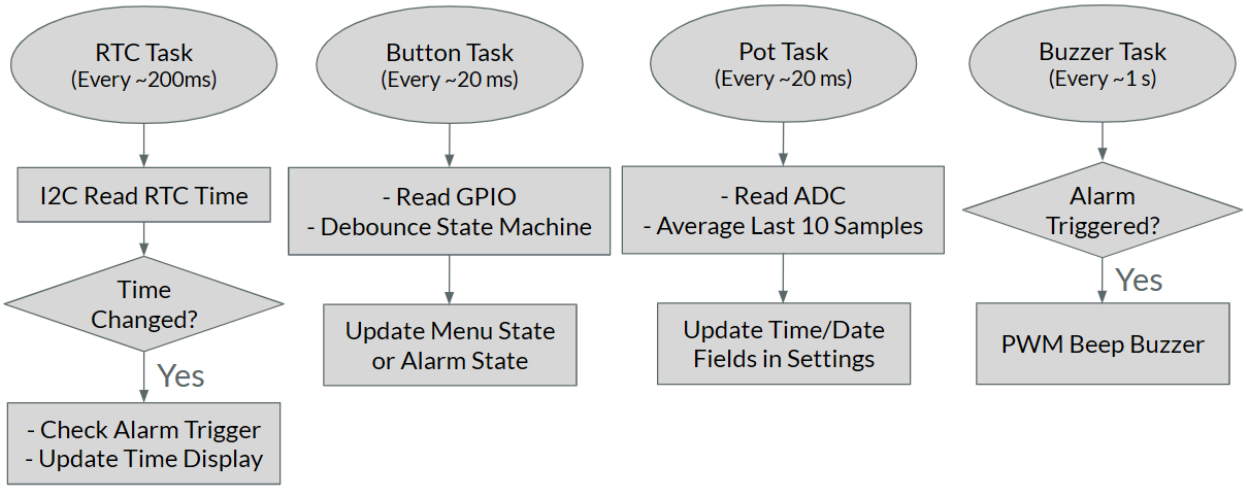

Figure 2. Block diagram showing all tasks.

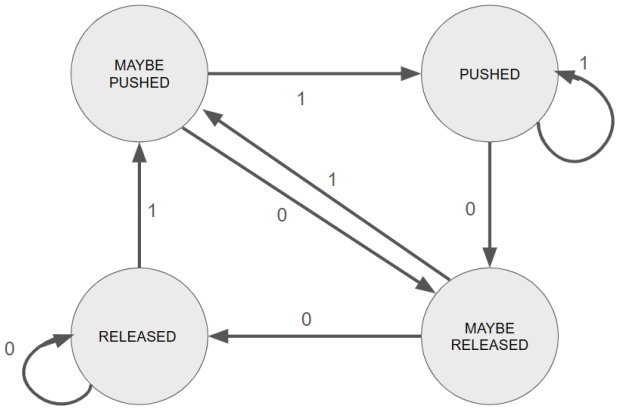

Figure 3. Button debouncing state machine. A 0 on an edge means that the GPIO indicates that the button is not pressed (GPIO pin high, since I use pull-ups). A 1 on an edge means that the GPIO indicates that the button is pressed (GPIO pin low).

Button	**Functionality**
1	• On the main screen, press to enter the settings menu. • In settings, use to navigate through options.
2	• On the main screen, hold view the alarm time. • When the alarm is beeping, press to turn off the alarm. • In the settings menu, use to navigate through options. • In the time/date/alarm setting menu, press to move to the next field.
3	• When the alarm is beeping, press to snooze the alarm for ten minutes. • In the settings, press to go back.

Table 2. Functionality of each button.

Figure 2 shows the high-level overview of all tasks. The main task with the most complex functionality is the button task. This task reads each button and debounces using the state machine in Figure 3. Then, it will advance the menu state based on the button that is pressed. Table 2 provides a description of the functionality of each button. Figure 4 shows all the menu screens.

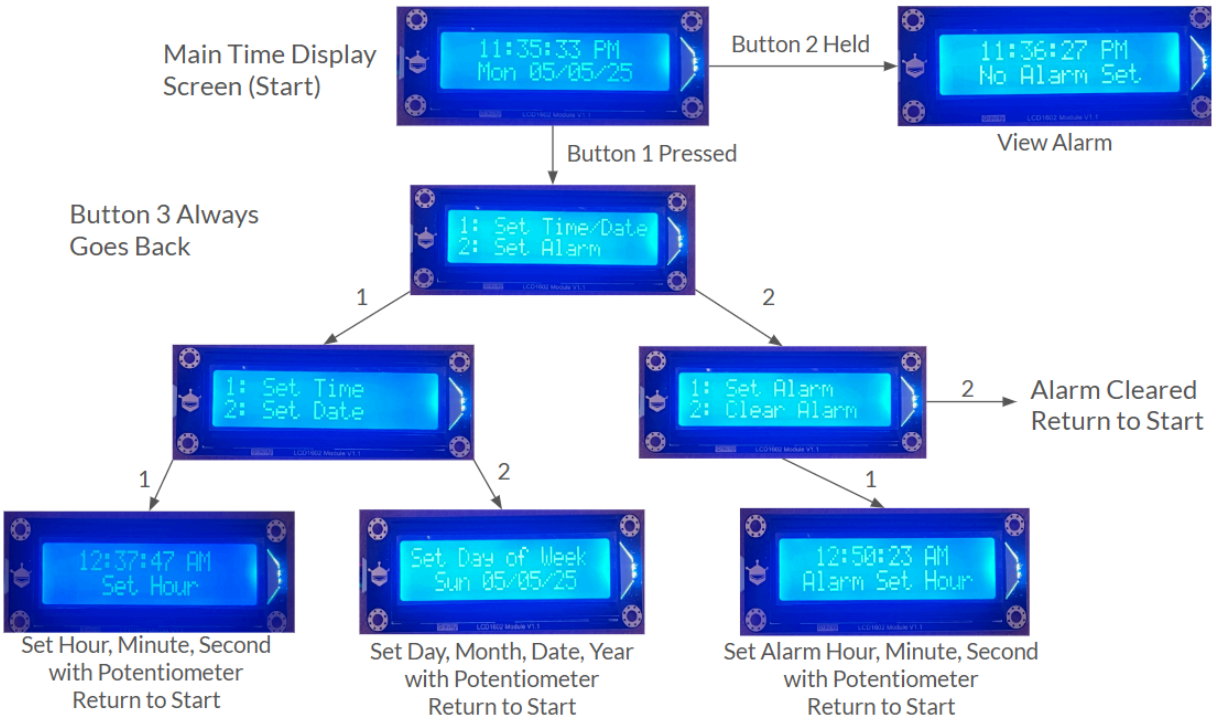

Figure 4. Menu state machine. 1 on an edge corresponds to button 1 being pressed. 2 on an edge corresponds to button 2 being pressed.

The bottom row of menu states each contain another state machine to keep track of the current field being set. For instance, the time setting state has sub-states for the hour, minute, second and confirmation screens. Within this screen, button 2 advances to the next state, while button 3 returns to the previous state. The potentiometer adjusts the value of the field corresponding to the current state.

Lessons Learned and Challenges

There were several major challenges faced throughout the project. One of the biggest challenges was accurate timekeeping. Initially, I implemented the project by using the AVR clock to track time. I developed the code to increment the seconds and manually rollover the seconds to minutes, minutes to hours and hours to days. However, after testing the full system, I realized that the AVR clock was not accurate. I left it running for 24 hours and it was several minutes off. To fix this, I learned about real-time clocks and added one to my design. The DS3231 RTC is

more accurate than the AVR clock because it has a temperature sensor and automatically compensates for variation in clock frequency due to the ambient temperature. After adding this to my design, the time appears to be tracked accurately down to the second. Furthermore, since the DS3231 has a coin-cell battery, it can track time even when not powered.

Another major challenge in my project was designing the user interface. For the alarm clock, all menu information has to be displayed on the 16 x 2 LCD. Since the LCD is quite small, I had to design a multi-level menu with many sub-menus to fit all the desired functionality. Furthermore, I had to make the user-input mechanisms extremely reliable. For instance, to prevent multiple bounces of buttons, I had to use a state machine. To prevent the potentiometer from having "jitter", I had to average the most recent values.

Finally, the last challenge was implementing the communication for both the LCD 1602 and DS3231 over I2C. This required reading the datasheet and thoroughly understanding both devices. To make it easier to interface with the DS3231, I found a C library online that implemented communication with the DS3231. However, I still had to make several changes to it to integrate it with the I2C library that we have already developed in class. Furthermore, I had to debug several issues related to the required delays between I2C messages with the device.