Peter M. VanNostrand

CSE 321 Real Time and Embedded Systems Final Project

DEMONSTRATION

My demonstration video can be found online on YouTube at
https://www.youtube.com/watch?v=B51cTrIvIwA

PROJECT TEAM

This project is an independent project. All aspects of this project were completed by the author of this report, Peter VanNostrand, utilizing open source libraries such as the Arduino standard libraries and hardware specific libraries from Adafruit. For detail on libraries used and their functions see the references section at the end of this report.

PROJECT GOALS

The goal of this project is to construct a Wi-Fi enabled smart lamp which would be capable of controlling RGB LED strips. The device should be able to take web requests from other internet connected devices and then switch between a variety of states including simple light on/off, color change, and dimming. The device should also be able to perform preprogrammed events such as sunrise simulation, with the lamp dimming on in the morning to help you wake up gradually. If time permits a buzzer will be added to make the device a fully functional dual purpose lamp and alarm clock. A backup battery system may also be implemented to ensure that a power outage does not prevent the alarm from ringing.

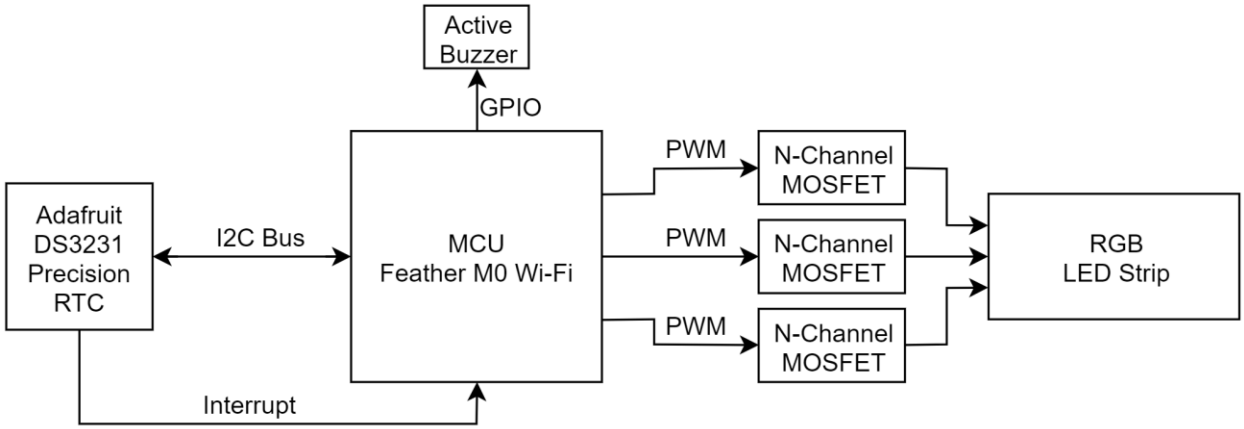
HARDWARE

This project using the following hardware components:

- Adafruit Feather M0 Wi-Fi Development board
- Adafruit DS3231 Precision Real Time Clock
- 3 N channel MOSFETs
- RGB LED Strip
- Active Buzzer
- 12V power supply and connector
- 12V to 5V Universal Battery Eliminator Circuit (buck converter)
- RTC backup battery
- 3.7V LiPo battery

BLOCK DIAGRAM

The hardware components are arranged as shown in the block diagram below.



The core of the system is a Feather M0 Wi-Fi development board. This acts as the system master device. When the board is powered on it connects to Wi-Fi and establishes a web server. This server receives HTTP requests from the user's device and returns an HTML page with embedded JQuery code. JQuery is an open source JavaScript library for event handling and animation. This page allows the user to interface with the system hardware by providing input values for LED brightness by color, alarm date time, and how long the sunrise animation should take. When the user clicks the "Update" button on this page a JQuery function builds these values into a URL and passes them to the web server via an HTTP GET request. This webserver acts as a centralized user input and is available to any device on the network making for easy control of the device.

The next major component is the Adafruit DS32321 Precision Real Time Clock (RTC). This device is connected to the Feather board via an I^2C bus as a slave device. With its independent battery backup this RTC is capable of keeping time even when the system is powered down and automatically adjusts for temperature changes, months of differing length and even leap years. The device can also generate an active low interrupt signal when the current time matches a time stored in its alarm register. This interrupt is fed back into the MCU through a digital input pin with integrated pull-up resistor where it triggers an interrupt service routine. The device is used to provide the MCU with accurate time and trigger the sunrise sequence and alarm. When the MCU powers up it checks to see that the RTC has an accurate time. If the RTC somehow lost power (i.e. the backup battery died or was removed) then the MCU connects to a NIST (National Institute of Standards and Technology) time server and uses the Network Time Protocol (NTP) to get the current time. It then writes this value via I^2C to the RTC. When the MCU receives an HTTP GET request from the user to set an alarm it writes the alarm time to the RTC via I^2C an enables the alarm interrupt. Using the RTC in this manner allows the MCU to dedicate all its time to handling web requests and only be interrupted to perform alarm functions when necessary.

Also connected to the MCU are a set of three N-channel enhancement MOSFETs. These allow the Feather board which operates on 3.3V logic to control the LED strip which operates on 12V. The MOSFETs are connected between the LED strip and ground, with the MOSFETs sources all connected to ground, the drains connected to the ground pin for each channel, and the gates connected to three pins of the MCU. The MCU can then generate a PWM signal on these pins which will cause the MOSFETs to rapidly connect and disconnect the LED strip from

ground, effectively turning the strip on and off very quickly. Using persistence of vision this makes the strip appear as if is becoming dimmer or brighter depending on the duty cycle of the PWM wave, and by changing the duty cycle of the wave independently for each color channel we can control the color of the light emitted by the strip.

Lastly an active buzzer is connected to the MCU via a digital output pin. When the MCU drives this pin to a logical high of 3.3V the buzzer will turn on and start producing a loud beep using its internal oscillator circuit. The MCU can then write the pin low to turn the buzzer off. This device is used to generate a "beep-beep" pattern when the alarm rings.

Power for these devices is provided from a 12V power supply and 12V-5V buck converter. The LED strips and buck converter are connected directly to the supply voltage with the RTC and MCU connected to the 5V output of the buck converter.

IMPLEMENTATION TIMELINE

Please note that as this project is also a learning experience the project timeline follows a "bootstrap" approach. Features were implemented in order of simplicity, starting with the simplest and then moving to the more complex, iterating each time and adding new functionality only as it is needed. This allowed me to troubleshoot the system easily as I could test each feature independently and then slowly combine them with known working code.

For example, when building the web interface, I could have built the entire page with all the fields I needed plus styling before ever loading it onto the Feather board. This may have been faster, but if anything wasn't working as intended there would be quite a lot of code to troubleshoot. So instead I started with a simple "Hello World" page then added just the necessary inputs to control LED brightness. Later I went back and added styling and fields for alarm time and sunrise duration. This made sure that at every step the code contained just the bare minimum needed to run, ensuring that all troubleshooting was performed within a limited scope. The development process I followed is broken down below.

1. Hardware Competency – The first step was for me to learn how to setup and interact with the various hardware components needed for this project.
    a. Solder headers to Feather M0 Wi-Fi development board and use serial interface to check that it is functioning correctly
    b. LED Control
        i. Connect a single MOSFET to the Feather board and use it to blink a single channel on the LED strip on and off using digitalWrite()
        ii. Use analogWrite() to set the LED strip to a specific brightness
        iii. Read in brightness value from serial interface to set LED strip to any brightness
        iv. Connect second and third MOSFETs to remaining LED strip channels
        v. Expand code to allow control of per channel brightness using the serial input
    c. Real Time Clock
        i. Solder headers to RTC module and connect to MCU via I^2C bus
        ii. Use the Adafruit RTC library to read and write time from the module
    d. Internet connectivity
        i. Use the Arduino WiFi101 library to connect the Feather to the internet

        ii. Program to the Feather to take HTTP requests and respond with a simple "HelloWorld" HTML page
2. Internet Integration – Now that I knew how to interact with the project hardware and connect to the internet I combined these to make hardware changes based on data from the internet.
    a. Setting the RTC to the current time
        i. Use the Arduino WiFi101 library to connect to a NTP server and fetch the current time
        ii. On boot check if the RTC lost power, and if so write the current time to the RTC
    b. Controlling the LED strip over the internet
        i. Write a static HTML page with several input fields for the user to enter the brightness for each channel of the LED strip
        ii. Add JQuery code that takes the inputs and adds them to the URL as query strings
        iii. Load this page onto the Feather board and have it returned when someone sends and HTTP request by opening the boards IP address in a web browser
        iv. Have the Feather parse each HTTP request for query strings and extract the brightness values for each channel
        v. Use these values and analogWrite() to set the brightness of each channel to the value sent by the user through the HTTP request
        vi. Add CSS to the user input page to allow for appropriate styling and formatting
3. Advanced Functionality – Once I had all the hardware working and could control the peripherals over the internet it was time to implement the higher level functions such as setting an alarm and triggering a sunrise sequence
    a. Setting an alarm – As the Adafruit RTC library does not support setting an alarm this must be done manually
        i. Connect the buzzer to a GPIO pin and set the pin as a digital output
        ii. Connect the RTC interrupt pin to the Feather and use the Arduino standard library to write an interrupt handler that set the buzzer pin high
        iii. Consult the RTC datasheet to determine alarm register configuration and learn to split the alarm datetime appropriately
        iv. Use the Arduino Wire library to write the datetime to the RTC alarm registers and enable the alarm interrupt
        v. Add a field to the HTML page to allow the user to enter an alarm time and use this value to set the alarm
    b. Sunrise animation
        i. Add a field in the HTML page to allow the user to enter a sunrise duration
        ii. Modify the code so that after reading in the alarm datetime it is saved instead of setting an alarm directly. When the sunrise duration is read from HTML page save it and then set an alarm for (alarm_time – sunrise_minutes)
        iii. Modify the interrupt handler so that it fires twice. The first time it fires is for the start of the sunrise sequence, at this point it sets a boolean flag to true to indicate that the sunrise has started and calculates how much brighter the

lights should get with each loop. It then sets another alarm for the original alarm_time. The second time it fires is for the alarm itself, here it sets a boolean flag to true to indicate that the alarm should starting ringing before resetting all the alarm fields to get ready for another alarm to be set

iv. Modify the loop to check for the sunrise boolean flag, if true increment the LED brightness by the calculated value each time the loop runs. When the brightness reaches the maximum value the boolean flag is set back to false and the increment value is reset to 0

v. Modify the loop to also check for the alarm boolean flag, if true beep the alarm twice and increment a beep counter. When the beep counter reaches a specified value the boolean flag is set to false to stop the system from beeping on the next loop and the beep counter is reset.

4. Debugging and cleanup

CHALLENGES

By far the biggest challenge in this project was managing datetimes to set the alarm. By convention most times are stored as a Unix time, which counts the number of seconds since January 1st 1970 and aligns with UTC (Universal Coordinated Time). However, the popular NTP (Network Time Protocol) counts seconds since January 1st 1900. In this system I've chosen to use UTC for all time measurements, so the NTP time from the time server needs to be converted to UTC. Thankfully we can convert NTP to UTC by subtracting 2208988800 seconds, the number of seconds in 70 years.

On top of this the user will almost definitely want to set the alarm date in time in their local time zone. To start we must first determine what time zone the user is currently in, this isn't as simple as determine where the user is because many places make use of daylight savings time. To do this I used the JavaScript function "getTimezoneOffset()" which gets the system time and determines the difference between the user's local time and UTC. This must then be multiplied to convert it to seconds and added to the user's time, finally resulting in an integer representing the UTC time of the alarm. Once this value was determined, and passed via query string to the MCU, it was relatively easy to save as the Adafruit RTC library has a datatype called DateTime which can be constructed from UTC seconds.

Despite this my time troubles were not over. The Adafruit RTC library provides methods for getting and setting the current time on the RTC, but does not provide functionality to set the alarm. In order to do I would have to manually use the Arduino Wire library to write a value into the RTC's registers. However, the RTC does not use large single registers to store time values, instead it uses part of a series of registers which all represent. For example this is the RTC register configuration for the alarm time.

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE

07h	A1M1	10 Seconds			Seconds	Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes	Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour	Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/DT	10 Date		Day	Alarm 1 Day	1–7
					Date	Alarm 1 Date	1–31

Here we can see that the seconds, minutes, hours, and date are divided across four separate registers with varying portions of each register representing the values. To set the alarm I had to split each time division (seconds, minutes, hours, date) into their respective 10s values and 1s values using modulus and divide. Then these values needed to be shifted by the appropriate amount and combined with each other using OR. Then I had to locate the appropriate bit flags for the left half of each register and OR these into the value for each register. After all this the values can be written via I^2C to the appropriate register.

Despite all this our alarm is still not ready to ring. In order to allow the RTC to trigger an alarm we must also go set an alarm_1_enable bit and an interrupt_enable bit in the RTC's control register. To do this without corrupting the rest of the control register I had to first locate and read the control register, then stop the I^2C communication, OR the control value with the enable flags and then write this value back into the control register.

Finally I wrapped all this in a function called set_alarm(DateTime dt) and used it in several places to set the appropriate alarms.

INTERESTING FUNCTIONAL COMPONENT

My favorite part of this project is the sunrise animation effect. I can easily set an alarm on my phone, and even control RGB LEDs with a cheap commercially available IR remote control, but without buying expensive home automation equipment and Phillips Hues bulbs there is no easy way to obtain this functionality with commercial equipment.

The sunrise animation works by setting a set of two alarms. When the user clicks "Update" on the user interface page a JQuery function executes that takes the time they entered and converts it into UTC seconds, then appends it to the URL as a query string. The same happens with the sunrise_minutes input from the user. These query strings are then passed to the server on the Feather board through an HTTP GET request, and the server parses this GET request until it finds the query strings. First it takes the alarm time and parses the string to an integer representing the UTC seconds calculated earlier, this value is stored as a DateTime variable called alarm_time. Then the server parses the sunrise_minutes string to an integer and stores it. Once this happens it checks to see if a value for alarm_time has already been provided, if so it converts the sunrise minutes into seconds and subtracts it from the alarm time. This new value represents the time at which the sunrise will start, and an alarm is set for this time using the set_alarm(DateTime dt) function described in the "Challenges" section of this report.

Later, when this alarm triggers it causes an interrupt service routine called alarm_interrupt() to fire. This function checks that the boolean is_first_interrupt is true and that a sunrise_minutes value has been received. If these condition hold it computes how many main loop executions will occur between now and the alarm time. Using this value, it calculates the value light_step which represents how much brighter the lights must get each time the loop runs such that the lights start at 0 brightness and end at their maximum brightness of 255 just as the alarm rings.

Finally, it sets a new alarm for the time stored in alarm_time, marks is_first_interrupt as false and the boolean value start_sunrise_sequence as true before exiting the interrupt. These changes tell the interrupt handler that the next time it fires it should cause the buzzer to activate rather than start a sunrise.

On the next loop execution, the conditional statement "if(start_sunrise_sequence)" evaluates to true and the code enters a new block. This block is responsible for gradually increasing the brightness every time the main loop runs. First it computes the new brightness value for the LED strip as (light_step*sunrise_loop_counter) and then sets the lights to this new value. If the new brightness was greater than 255 we know the sunrise_animation must be complete so start_sunrise_sequence is set to false and sunrise_loop_counter is reset to 0. Otherwise the sunrise_loop_counter is incremented and the loop restarts.

By handling the sunrise animation in this fashion we can ensure that the sunrise alarm is captured by the interrupt, even if the loop is busy. It also leaves the brightness changes to the loop so that they can occur gradually and ensures that the interrupt takes the minimum possible time.

REFERENCES

The Adafruit RTClib library was used to read and write time to the real time clock as well as to store datetimes with its DateTime class.

https://github.com/adafruit/RTClib

The National Institute of Technology (NIST) time servers were used to fetch the current date and time. Their reference can be found online at

https://tf.nist.gov/tf-cgi/servers.cgi

The Arduino standard Wire and WiFi101 libraries were used for I^2C and WiFi functionality respectively and can be found at

https://www.arduino.cc/en/Reference/Libraries