Lab 5: System Diagnostics

Taylor Okel

10/9/2018

Table of Contents

Table of Figures

Problem Statement

Description

The problem definition is defined in the Lab document. (Stakem and Crum n.d.)

Requirements

The following requirements were derived from the Lab 5 description (Stakem and Crum n.d.)

- The RTC should be utilized
- 3 LEDs should be wired – Pump, Vent, and Inhibitor
- Two State Machines should be created;
    - Normal – begins at the top of the minute, and rotates through 4 states over 20 seconds before waiting for the next minute
    - Diagnostic – Transitions between the previous states on button presses. Allows for RTC time updates.
- A user can control entering/exiting diagnostic mode through a long press of the button.

Assumptions

- All equipment functions as advertised on their respective datasheets
- The board has a high enough frequency to reasonably complete all tasks assigned to it
- The time constraints for mode change (1ms) start after a full long press. This way, the long press can be specified to be as long as the implementer desires
- No time constraint given for state changes, assume rough visual verification sufficient

Equipment

Hardware

- Olimexino-STM32 Development Board (henceforth “board”)
    - STM32F103RBT6 Arm Processor (Olimex 2016)
- USB programming cable
- Programming Workstation (henceforth “computer”)
    - Windows 10 operating system
    - Arduino IDE
- Pushbutton
- Two Red LEDs
- Yellow LED
- Various jumper cables
- Various resistors
- Breadboard

Architecture

Three LEDs and a pushbutton need to be connected to the board. Each LED will connect to the ground via resistors. The button will be pulled to ground using a 10KΩ resistor, with the button shorting the pin to 3.3V. This design is shown in Figure 1; the wired design is shown in Figure 8.

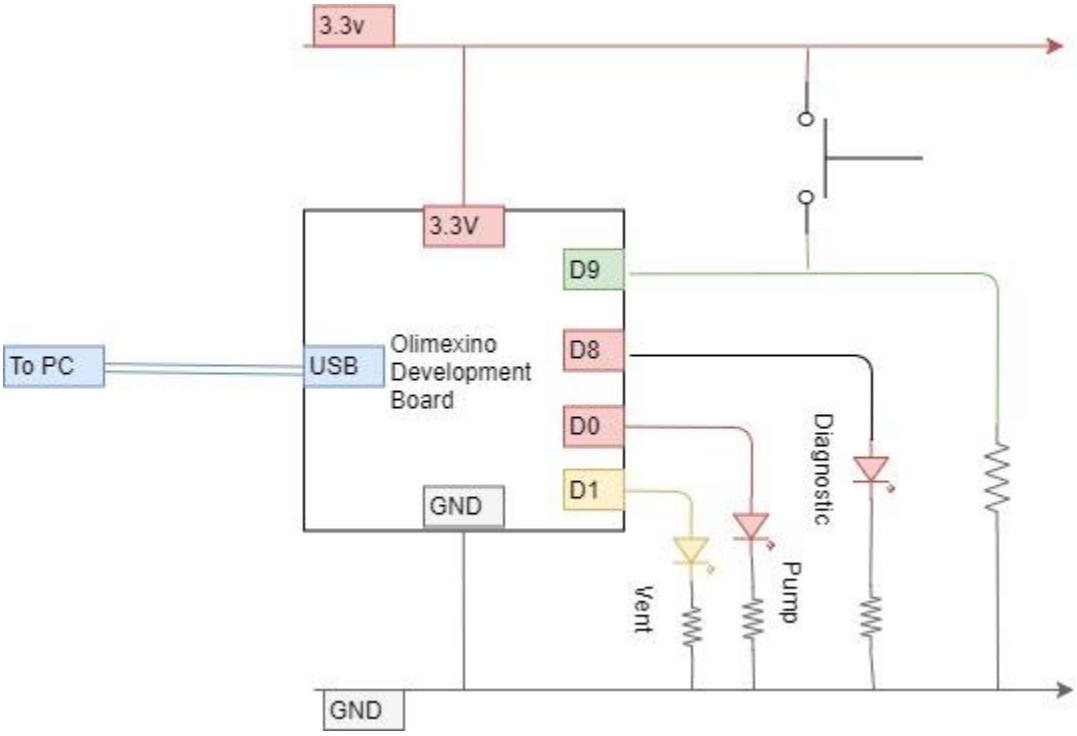

Figure 1: Wiring Diagram

Design

The design will consist of two major pieces: Normal Operation and Diagnostic Mode. Switching between the two will be controlled via switching between the two using button long presses, the diagnostic mode further by short presses for state changes and SerialMonitor input.

To run normal operation, as well as determine "long presses", the RTC Controller from the previous lab will be used. Several new functions, dealing with the alarm interrupts, were introduced.

RTC Controller

The implementation of the bulk of the API is defined in the previous lab writeup (Okel, Lab 4: Real Time Clock 2018). The following functions were added to the API:

- rtcSetAlarm – sets alarm registers to value
- rtcAttachInterrupt – enables alarm interrupt line and attaches handler function

- rtcDetachInterrupt – disables alarm interrupt line and detaches handler function

rtcSetAlarm

The RTC must be initialized prior to this function. The flow follows the previous set API function calls. The controller waits for the RTC clock to sync and any config operations to complete, if needed. Once this is done, the Alarm value is written to the alarm registers. The RTC is then taken out of config mode, allowed to complete the write, and control is returned to the program.

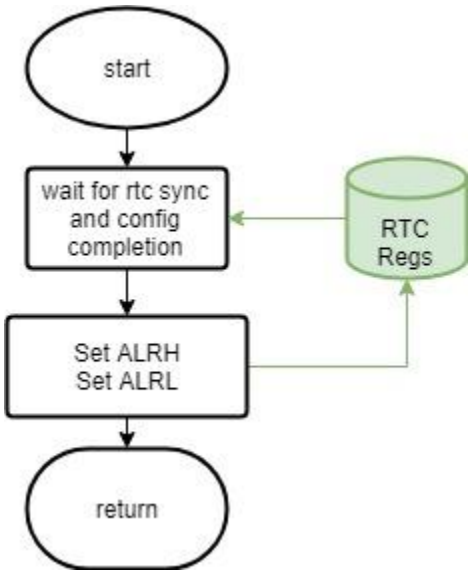

Figure 2: rtcSetAlarm flowchart

rtcAttachAlarmInterrupt

The RTC must be initialized prior to this function. The flow follows the previous attach interrupt API function calls. The controller stores the interrupt handler into a known location, and enables the NVIC register value for the RTC Global interrupt. On interrupt, the __irq_rtc routine handles the interrupt, calling the interrupt function, clearing the interrupt bit and returning control.

Note that the __irq_rtc routine is common for (global) Alarm and Second interrupts. While not shown in Figure 3 (for simplicity), the RTC structure and __irq_rtc were expanded to handle *both* calls by checking the interrupt flags. Both will be handled at once, if necessary.

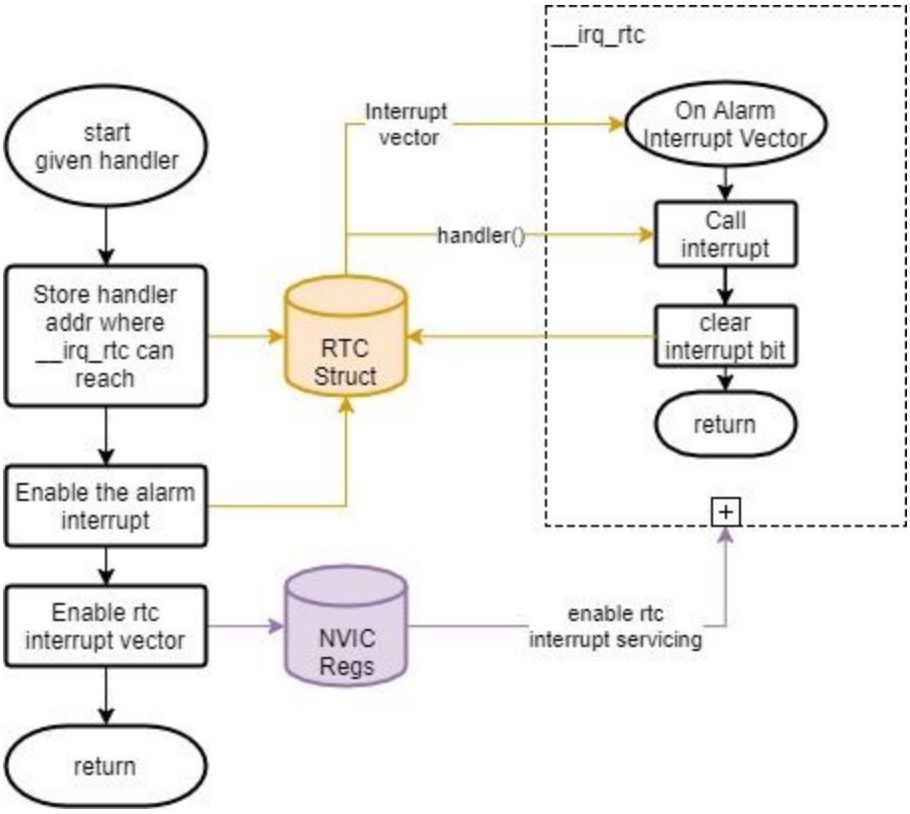

Figure 3: rtcAttachAlarmInt Flowchart

rtcDetachInterrupt

The RTC must be initialized prior to this function. The interrupt vector must be disabled in the RTC and NVIC, and the handler function set to NULL. Prevents the interrupt from firing, masks the interrupt if it does anyways, and will cause a program crash if the previous two fail as it attempts to call a NULL handler. This final effect is desired, as if both failsafes fail, we do not want to go into an unknown state. This design is shown in Figure 4.

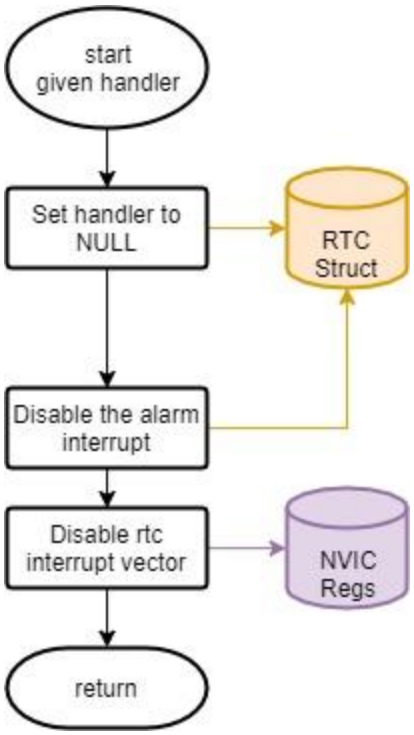

Figure 4: rtcDetachAlarmInt Flowchart

Program design

At the top-level, the state machine will be abstracted into a header file, defined in the following sections. The top-level program is responsible for setting up IO, the RTC, and state machine in the setup routine. Once the machine is running, the controller polls the button and serial lines. If the button is held for 5 seconds (configurable via a macro at the top of the file), the design switches between Normal and Diagnostic Mode. If the design is in Diagnostic Mode, the design also will poll the serial line for RTC updates. This design is shown in Figure 5.

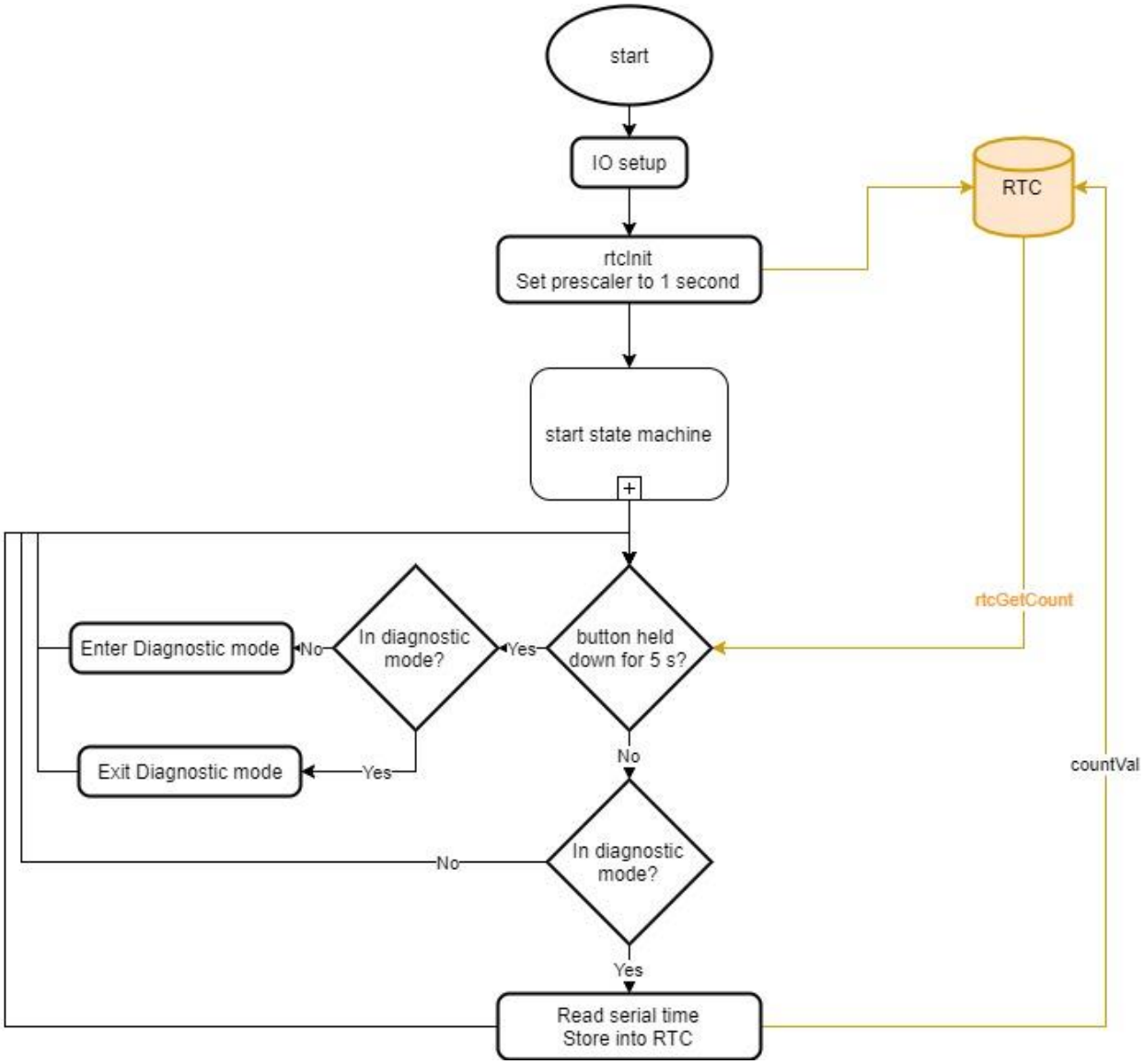

Figure 5: Main Flowchart

State Machine

The state machine consists of 8 function calls; pumpOn, pumpOff, ventOff, and ventOn, and diagnostic calls for each. The initialization of the state machine sets a separate function pointer to pumpOn, with each subsequent call changing this pointer to the next in the state flow, as described in the lab document (Stakem and Crum n.d.).

Normal Mode

To enter normal mode, the design calls initStateMachine. This function sets the pointer described above to the first function call, pumpOn, sets the alarm value to the top of the minute, detaches any interrupt from the button, and attaches the alarm interrupt. The interrupt handler routine is the function pointer, which is set to the next state on each interrupt (as described below).

The following states are then looped through:

pumpOn

- Turn on the Pump LED
- Get the RTC count
- Set the alarm to the current time plus 5 seconds
- Set the next state to ventOn

ventOn

- Turn on the Vent LED
- Get the RTC count
- Set the alarm to the current time plus 5 seconds
- Set the next state to pumpOff

pumpOff

- Turn off the Pump LED
- Get the RTC count
- Set the alarm to the current time plus 5 seconds
- Set the next state to ventOff

ventOff

- Turn off the Vent LED
- Get the RTC count
- Set the alarm to the current time plus 5 seconds
- Set the next state to pumpOn

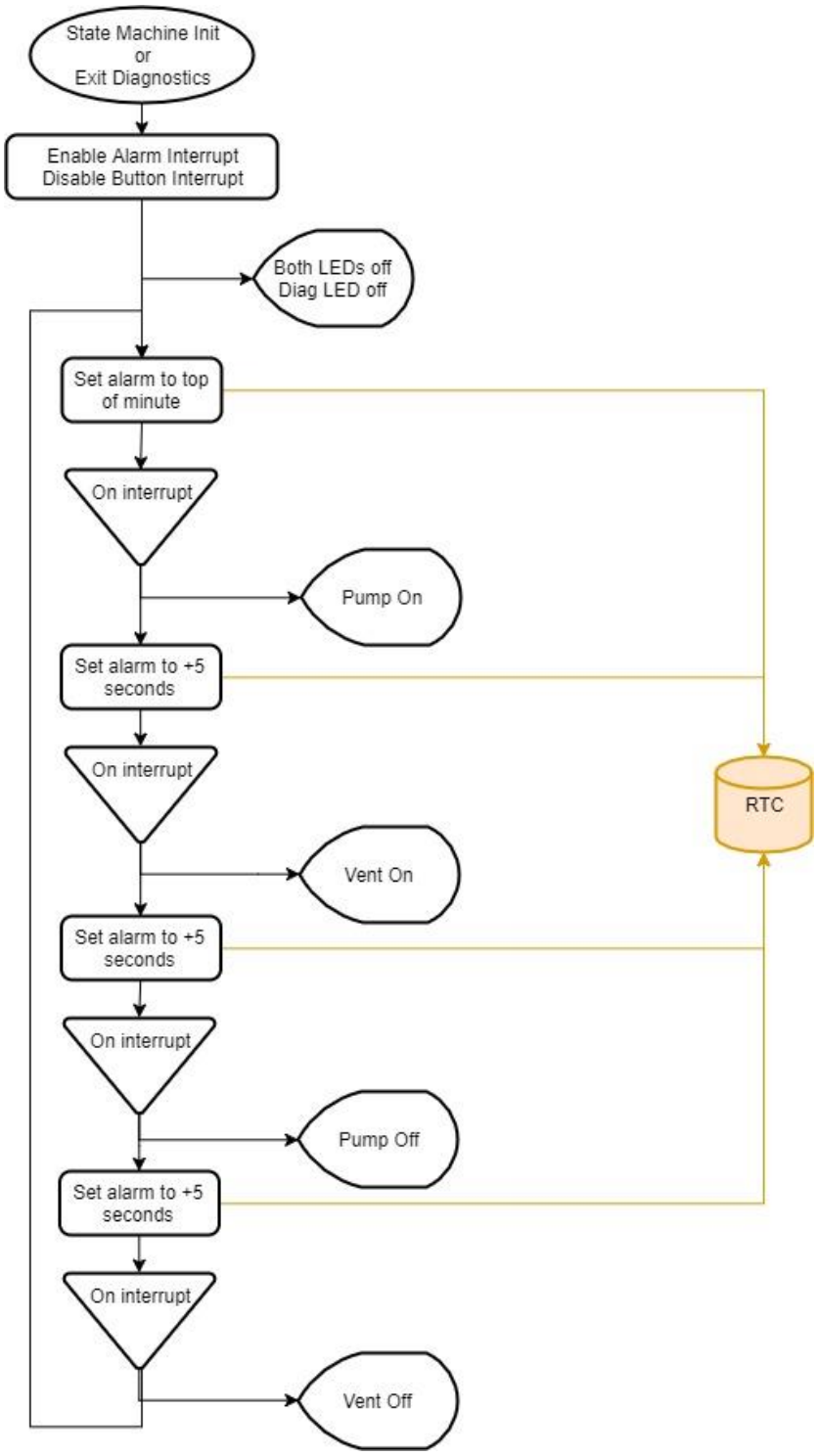

Figure 6: State Machine in Normal Mode

Diagnostic Mode

The basic idea for this mode is the same as Normal, except the transitions happen on button press instead of alarm values.

To enter Diagnostic mode, the enterDiagnostics function is called. This function flags diagnostic mode for the main function, detaches the alarm interrupt, attaches the button interrupt, drives both Pump and Vent LEDs low, and drives the Inhibit LED high.

To exit Diagnostic mode, the exitDiagnostic function drives the Inhibit LED low and calls the stateMachineInit function, described in Normal Mode.

The states then flow using the handler function pointer like Normal mode, transitioning on button press. The button presses need to be software debounced; this is accomplished by wrapping the handler function pointer in a condition that prevents the function from calling more than once per 200 milliseconds. By this point, the button is sufficiently stable and can be trusted as another input.

The state machine design:

pumpOn

- Turn on the Pump LED
- Set the next state to ventOn

ventOn

- Turn on the Vent LED
- Set the next state to pumpOff

pumpOff

- Turn off the Pump LED
- Set the next state to ventOff

ventOff

- Turn off the Vent LED
- Set the next state to pumpOn

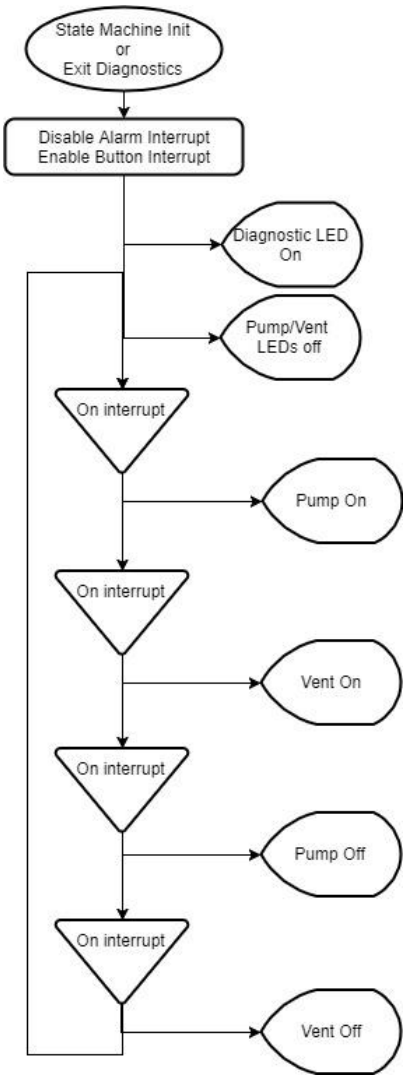

Figure 7: State Machine Diagnostics Mode

In addition to the state machine changes, the main function also allows for setting the RTC values.

Implementation

The schematic depicted in Figure 1 was wired onto the board with the help of the given breadboard. The USB cable was connected to the Host PC.

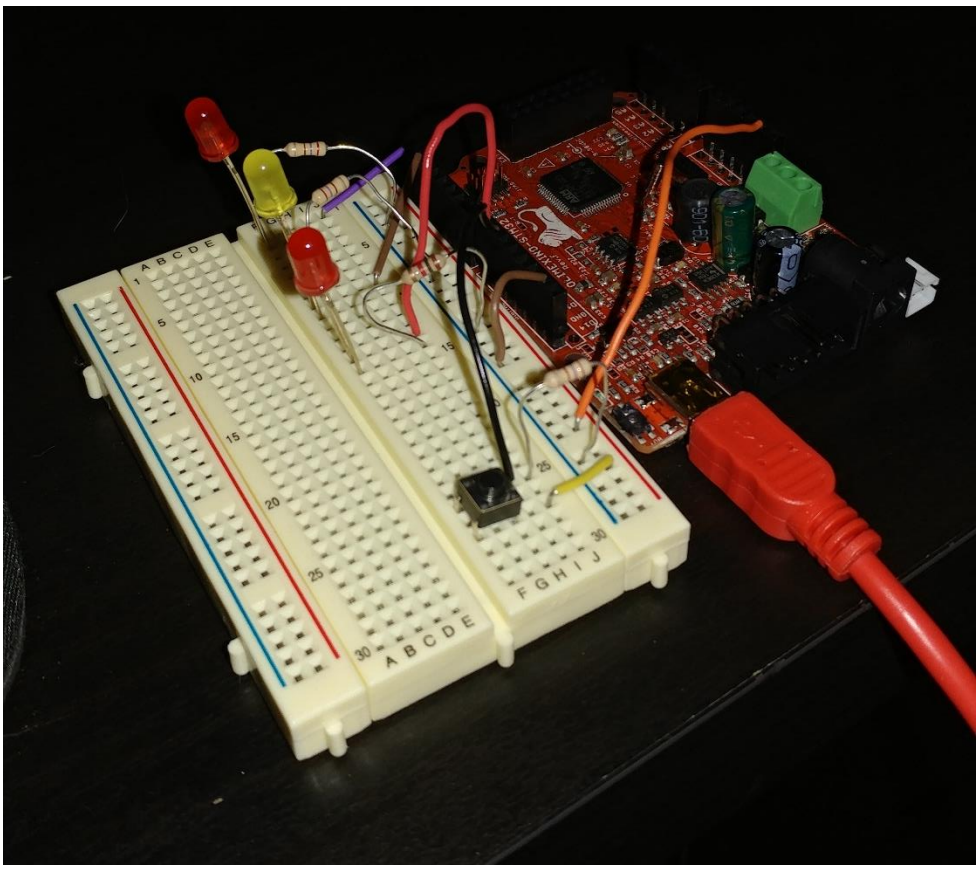

Figure 8: Schematic On-board

The main code is loaded through Lab5.ino. The RTC API is provided through RTCController.h, and the state machine is abstracted in StateMachine.h. Arduino libraries are included by default in the Arduino IDE. After the code is compiled by the Arduino IDE, the IDE then uploads the binary to the board, which begins running after a short delay. The user then interacts by entering values into the Arduino Serial Monitor or pressing the pushbutton on the breadboard.

Arduino Functions Used

- attachInterrupt – attaches the button interrupt in diagnostic mode
- detachInterrupt – detaches the button interrupt for exiting diagnostic mode
- Serial.read – read values from the serial line for setting the RTC value in diagnostic mode
- Serial.print[ln] – debug statements written to the tester
- digitalWrite – LED drivers
- digitalRead – implements the long button press switching mechanism
- millis – used in software debounce
- pinMode – digital IO setup

Test Plan

1) Wire the schematic
2) Compile the test Payload.
3) Plug the USB into the computer and board.
4) Upload the test payload
5) Test normal operation by letting it run for two cycles.
    a) Measure the time between transitions
6) Test ability to switch into diagnostic mode by long button press (5s)
    a) Confirm with Inhibit LED
    b) Switch when Pump/Vent LEDs lit to confirm diagnostic mode drives low
    c) Both should happen <1ms after the 5s long press
7) Test diagnostic operation by running through two cycles
    a) Confirm that software debounce is sufficient by using moderately quick (≥ 200ms frequency) button presses
    b) Only one transition per press should occur
8) Test diagnostic operation by writing into RTC value
    a) Use print statements for quick verification
9) Test ability to switch back to normal operation
    a) Inhibit LED turns off
    b) Pump/Vent LEDs turn off
    c) Both should happen <1ms after the 5s long press
10) Confirm that normal operation continues

Test Results

Table 1: Lab 5 Test Results, the successful test matrix, was obtained after some trial and error to get the code in working order.

Table 1: Lab 5 Test Results

Test Number	Pass/Fail	Comments
1	Pass	Visual Inspection
2	Pass	Compilation
3	Pass	
4	Pass	Upload successful
5	Pass	Transitions not perfectly on 5 second boundary; resolution issues with alarm, overhead due to interrupt
6	Pass	
7	Pass	
8	Pass	
9	Pass	
10	Pass	

A subset of these tests is demonstrated in the following YouTube video:

https://youtu.be/EGV8bUaGFnI

Suggestions

Future Improvements to this Implementation

- Better time input – currently, only an integer value representing time since epoch is accepted. Some method to accept multiple input formats could be implemented.

Code

The code can be viewed on the author's class GitHub page.

https://github.com/525-615-81-FA18/labs-okeltw

The code was tagged to provide snapshots into the development process:

- Lab5_Rev1 – Bulk of Lab working
- Lab5_Rev2 – Ability to write RTC during diagnostic; additional documentation and drawings
- Lab5_Rev3 – Move digitalWrites on enterDiagnostic for better timing, deleted unnecessary debug statements, more documentation

References

Okel, Taylor. 2018. "Lab 4: Real Time Clock." Lab Report.

Olimex. 2016. "Olimexino-STM32 development board User's manual." Olimex Ltd, September.

Stakem, Pat, and Gary Crum. n.d. "Lab 5 – Interrupt Capture, State Machines and Debounce." *Blackboard.* Accessed 10 2, 2018. https://blackboard.jhu.edu/webapps/blackboard/content/listContent.jsp?course_id=_171040_1&content_id=_5545672_1.