Software Formalization

Year: 2021 Semester: Fall Team: 8 Project: Sink or Be Sunk

Creation Date: October 5, 2021 Last Modified: October 13, 2021

Author: Joe Mislansky Email: jmislans@purdue.edu

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** | 5 | x2 |  |  |
| **Description of Components** | 5 | X3 |  |  |
| **Testing Plan** | 5 | x3 |  |  |
| **Software Component Diagram** | 5 | x4 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** | 5 | x2 |  |  |
| **Formatting and Citations** | 5 | x1 |  |  |
| **Figures and Graphs** | 5 | x2 |  |  |
| **Technical Writing Style** | 5 | x3 |  |  |
| **Total Score** | 100 | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

*Great descriptions with details on the software components!*

1.0 Utilization of Third Party Software

Most of the software on the MCU uses header and C files provided by Espressif. Some code has been modeled off the examples given in the Espressif documentation as well. The only third-party software we are using is cJSON which “aims to be the dumbest possible parser that you can get your job done with.” [1] This is licensed under the MIT License so that we can use it for free however we want as long as we include the original copyright. This software enables the use of a cJSON struct data type that allows the MCU to parse data into and out of JSON which is what WebSocket uses to send data back and forth between the ESP32 and the server.

2.0 Description of Software Components

Our main software/firmware components of our design are boat detection, basic I/O, LED array, and WebSocket server.

The boat detection is responsible for determining where the user places the boats. This is done an array of locations that will be shorted by the boat being plugged in. These will be muxed into the ESP32 and the software will essentially scan for the first short in the array. Once it finds that it will search in the rest of that row and column to determine where the other pin is for the boat. Once that is completed, it will be determined how far apart the pins are which will tell the server which boat was placed, and which coordinates it is located in. Then, those pins will be removed from the scanning. This will be repeated for all four boats. All this code is being developed by the team.

The basic input/output is responsible for user input and output during the game. This specifically does not include the before game boat detection as this component is considered separate from the in-game interactions. The keypad will be used for user input, mainly the attack location. It will also be used for any sort of user confirmation of actions or selecting game modes. The audio and rumble are outputs to increase the user experience. These are very simple vibrations controlled by a GPIO that will be flipped on for a short duration and simple audio files controlled through the DAC. Lastly, the LCD is controlled through SPI2 and will put out simple gameplay messages such as confirmation of attack location and game select.

The LED is another basic output that is controlled by one data input line. The software for turning on and off the LEDs will be modeled as a function in which we turn on an LED based on the server sending data as a hit or a miss. The data will be kept on the server about which LEDs are on or off at any given time and will send data packets to the ESP32. Once it receives the data it will send a data signal through a single GPIO pin to turn on or off the correct LEDs with the correct colors.

The WebSocket server is responsible for communication between ESP32 and the Sink or Be Sunk website as well as board to board gameplay. This will control all the game flow as well. The two main functions are wifi\_send and websocket\_event\_handler. Wifi\_send exists to send data from the esp32 to the server. This will be used for all input communications from the user such as where the boats are being placed and where the user is trying to attack. Websocket\_event\_handler on the other hand will handle the data being sent and received by the server itself. It will send all communication to the boards such as whether user inputs are hits or misses, if a user has sunk a battleship, if the game has ended, and what moves the opponent makes. The server will also keep track of whose turn it is, if they’ve entered valid coordinates, etc. Keeping most of the game logic on the server allows us to reduce the number of computations on the ESP32 which will in turn lower the power necessary to run everything. The server is also responsible for initializing the game. The ESP32 using BLE will connect to a device such as a smartphone to provide the correct Wi-Fi information to the board. Once provisioned successfully, the board will break the connection with the phone and communicate solely with the server over Wi-Fi.

3.0 Testing Plan

The priority order of our testing is boat detection, sending data to the server, receiving data from the server, LED usage, keypad usage, LCD display, and audio/tactile outputs.

The testing for the boat logic will be very straightforward. Testing for the prototypes has already been underway by the team in which we short random pins in a small-scale practice grid and check to see if the correct location is being noted on the LCD screen. This testing will be expanded once the rest of the boat detection code is written by placing two pins at once and making sure the system identifies the boat and all covered pins correctly. The next step in testing will be making sure that two boats can be placed back-to-back and that the system will still be able to scan successfully. The boat logic is the highest priority because it is the most integral part of the game.

The testing for the basic input and output will be a lot simpler. Testing the keypad just involves pushing the input to the LCD to show the output and this has already been completed. We already have proof of this working. The rumble motor and speaker will be tested easily by sending a signal from the ESP32 and checking to make sure the motor turns on and off quickly. The audio testing will be done by sending a basic signal at first and then by testing with a short audio file. The audio will be short phrases such as a splash or dialogue such as “you sunk my battleship.” The audio and rumble motor are lower priorities.

The testing for the LED will be a little bit more complicated. We already know that we can turn on a singular LED with different colors. Next, we will make sure we can turn on individual LEDs in a short series for testing. Once this is successful, we will transition to putting different colors on different LEDs. We will also need to do load testing in which we turn on many LEDs at the same time to make sure the MCU can handle sending the data quickly through 128 LEDs.

The game flow testing will require sending messages back and forth from the board to the server and vice versa. To test board to server, we will link dummy JSON packets to different keys on the keypad. Hitting a button will send the data over to the server and then we will check the database to see that the data has been successfully added. To test server to board, we will create more dummy JSON packets to send other that tell the board if attacks are hits or misses. We will confirm this through the LCD display simply by printing hit or miss text. By testing with the keypad and LCD display as opposed to the boat pins and LEDs, we eliminate any possibility of compounding errors. We have enough background knowledge of the keypad and LCD that we can be confident that those are working on their own and would not be the reason that tests could be failing.

4.0 Sources Cited:

[1] Gamble, Dave. “cJSON,” *GitHub.* [Online] Avaliable: https://github.com/DaveGamble/cJSON [Accessed Oct 2021]

Appendix 1: Software Component Diagram

![A screenshot of a computer

Description automatically generated with medium confidence]()

*Figure 1: Overall Code Structure*

This is the overall breakdown structure of the software for the project. The two major components are the software that exists on the WebSocket server and the firmware that exists on the ESP32.

*A picture containing timeline

Description automatically generated*

*Figure 2: WebSocket Code Structure*

*A screenshot of a computer

Description automatically generated with low confidence*

*Figure 3: ESP32 Code Structure*

*Timeline

Description automatically generated with low confidence*

*Figure 4: Basic I/O Code Structure*