Software Formalization

Year: 2023 Semester: Spring Team: 10 Project: Parking Tracking System

Creation Date: February 14, 2023 Last Modified: February 18, 2023

Author: Jonathan Doorn Email: jdoorn@purdue.edu

Assignment Evaluation:

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

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

General Comments:

1.0 Utilization of Third Party Software

The team does not plan on using any Third Party software to carry out the logic required to operate our parking tracking system. However, the team has used snippets of code from Purdue’s ECE362 course to make use of modules featured in previous labs, such as the keypad and LCD display found on the Aggregator Module (AM) of the tracking system. Additionally, the team has used the STM32CubeMX tool to generate initialization code and provide low level libraries for interaction with microcontroller peripherals. The initialization code generated by this tool is licensed under the BSD 3-Clause license, and the HAL and LL libraries provided with the initialization code are licensed under the Apache 2.0 license. Both of these licenses [1] and [2] are royalty free and permissive, allowing the team to use code from these libraries in our project, so long as the disclaimers attached to the source code of these libraries are not removed or modified. These licenses prevent the team from holding the authors of the initialization code and peripheral libraries accountable if using the code results in damages to the team or the team’s customers.

2.0 Description of Software Components

The software of the parking tracking system can be broken into initialization code, interrupt handlers, module APIs, global state, and the main loop. All code has been written by the team unless specifically noted.

**2.1 Initialization Code (Figure 1)**

The initialization code has been produced for the most part by the STM32CubeMX tool to configure GPIO pins, SPI and USART controllers, and Timers to behave as expected for the module APIs. In addition to this generated code, there are functions from the Module APIs which can be considered as initialization code, such as code to turn on and clear the LCD display, and code to reset and broadcast a wireless access point from the ESP-01 Wifi module.

Initialization code for the LCD module and Keypad has been reused from Purdue’s ECE362 laboratory exercises.

2.2 Interrupt Handlers (Figure 2)

There are several interrupt handlers which serve to update global state variables. The USART interrupt handler pushes incoming data from USART receive data registers into a buffer for handling at a later time. The timer interrupt handler for the keypad cycles a logic high signal between keypad columns to detect which key is being pressed by the user. The timer interrupt handler for the car detection function on the car detector module (CDM) sets flags to send a flow message to the AM if a valid sequence of proximity measurements are received by the CDM and its slave module. The timer interrupt handler for the microsecond counter timer increments a global variable counting microseconds for use in determining the distance measured by the proximity sensor.

The interrupt handlers for the keypad operation have been reused from lab code in Purdue’s ECE362 course.

2.3 Module APIs (Figure 3)

There are exported APIs to interact with the modules connected to the AM and CDM to enable the functionalities guaranteed by the system. The Wifi API contains functions to create and connect to wireless access points through the ESP-01 module, and send or receive data once connected to an access point. The LCD Display API contains functions to write lines of text to the LCD display. The Keypad API contains functions to obtain a keypress value from the user. The Proximity Sensor API contains functions to measure the distance from a connected proximity sensor. The Seven Segment Display API contains functions to display a number across several large seven segment digits.

LCD and Keypad API functions have been reused from Purdue’s ECE362 course.

2.4 Global Datastructures (Figure 4)

The interrupt handlers and module APIs both rely on global datastructures to enable system functionality. The system has three high level datastructures, the first being an API message, which serializes and deserializes messages sent through the wifi network connecting all CDMs to the AM. This API message contains three fields: the module ID of the sender, the API action, and the API data body. The network data datastructure holds the last incoming message received by the Wifi module, and contains a count of the received data along with a buffer where the received data is stored. Finally, the global state datastructure holds flags associated with global state, such as proximity sensor trigger state, and parking lot capacity count.

2.5 Main Loop (Figure 5)

On the CDMs, the main loop will monitor global state to determine when to send a message to the AM if a car is detected. Additionally, the main loop will monitor the status of buffers filled by the Wifi module to receive proximity trigger state from a CDM’s slave module. On the AM, the main loop will monitor the status of buffers filled by the Wifi module to receive flow messages from the CDMs in order to update parking lot capacity count.

3.0 Testing Plan

3.1 Wifi Module

Priority 2

The wifi module API is used by both the AM and CDMs. The AM creates a network and the CDMs connect to the network created by the AM. Additionally, the slaves of each CDM must connect to the AM’s network. The usage of this module should be robust to disconnects, such that if the AM powers off and powers back on, all CDMs will still be able to function and automatically reconnect to the network once the AM is back in operation.

A procedure to test this module is as follows:

1. Power on the AM and perform initial configuration of lot capacity
2. Power on CDMs and CDM slaves
3. Ensure that CDMs trigger update in lot capacity for simulated inflows and outflows
4. Cycle the power on the AM
5. Ensure that after one minute of the power cycle, all CDMs are able to reconnect to the AM and trigger capacity updates

3.2 Proximity Module

Priority 3 (lowest)

For each CDM, there is an on board proximity sensor, and a slave associated with the CDM with another proximity sensor. To trigger a flow message, the proximity sensor of both the CDM and its slave must measure a distance lower than a specified threshold within a timeout period. During this timeout period, duplicate triggers should be ignored.

A procedure to test this module:

1. Test outflow
   1. Approach the proximity sensor of CDM’s until CDM registers a proximity trigger
   2. Approach the proximity sensor of CDM’s slave until Slave registers a proximity trigger
   3. Ensure that the AM capacity display reflects an outflow event
2. Test inflow
   1. Perform step 1b.
   2. Perform step 1a.
   3. Ensure AM capacity display reflects an inflow event
3. Test duplicate rejection
   1. Perform step 1a.
   2. Within timeout period, perform step 1a. again
   3. Perform step 1b.
   4. Ensure that the AM capacity display reflects a single outflow event
   5. Repeat from a. with steps 1a. and 1b. swapped, and ensure that capacity display reflects a single inflow event.
4. Test timeout
   1. Perform step 1a.
   2. Wait until timeout period is exceeded
   3. Perform step 1b.
   4. Ensure AM capacity display reflects no change
   5. Within timeout period, perform step 1a.
   6. Ensure that AM capacity display reflects an inflow

3.3 Aggregator Module

Priority 1 (highest)

The aggregator module will be responsible for maintaining the number of free spaces in a parking lot. This value is initialized upon AM bootup, and then modified with each inflow and outflow event. The valid range of free spaces should be in the range of [0, MAX\_UINT]. However, it will be impossible to display a count greater than 99 from the seven segment displays which we are using for the project. The internal count should stay within this range, and display 99 on the seven segment digits if the free space count is in the range [99, MAX\_UINT]. If there is an outflow when the internal count is zero, the internal count should remain at zero.

Testing procedure:

1. Turn on AM, ensure that the AM creates a Wifi Access Point
2. Initialize capacity to zero. Ensure that seven segment digits read 00.
3. Send an inflow message to the AM.
4. Ensure that digits still read 00.
5. Send an outflow message to the AM.
6. Ensure digits read 01.
7. Repeat step 5 until digits read 10. Ensure that one outflow message results in one increment.
8. Send an inflow message to the AM.
9. Ensure digits read 09.
10. Repeat step 8 until digits read 00. Ensure that one outflow message results in one decrement.
11. Power cycle the AM
12. Initialize capacity to 95.
13. Send outflow messages until capacity reaches 99, and ensure the display reflects one increment per outflow message.
14. Send two additional outflow messages. Internal capacity should reflect 101, however the display should still read 99.
15. Send three inflow messages. Internal capacity should reflect 98, and the external display should also show a capacity of 98.

4.0 Sources Cited:

[1] Apache 2.0 License. Open Source Initiative. Accessed: Feb. 18, 2023. [Online]. Available: <https://opensource.org/license/apache-2-0/>

[2] BSD 3-Clause License. Open Source Initiative. Accessed: Feb. 18, 2023. [Online]. Available: <https://opensource.org/license/bsd-3-clause/>

Appendix 1: Software Component Diagram

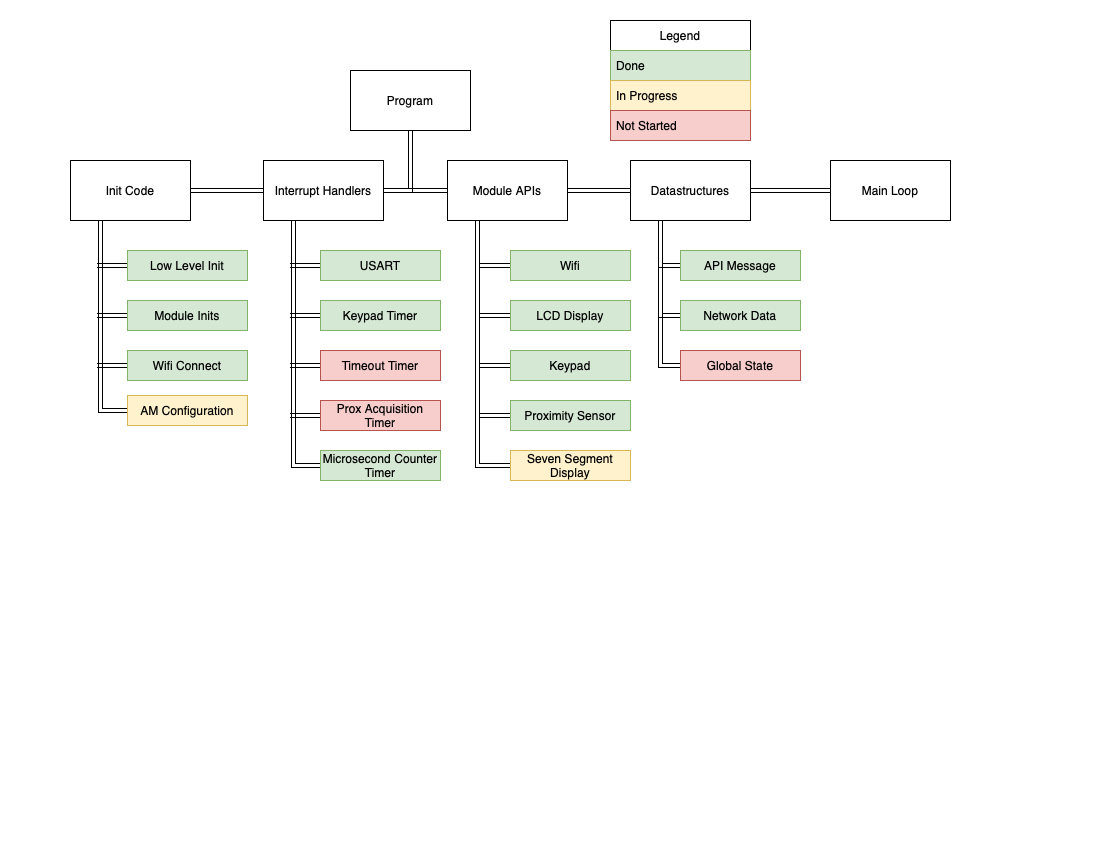
**

Figure 1: Program Modular Block Diagram

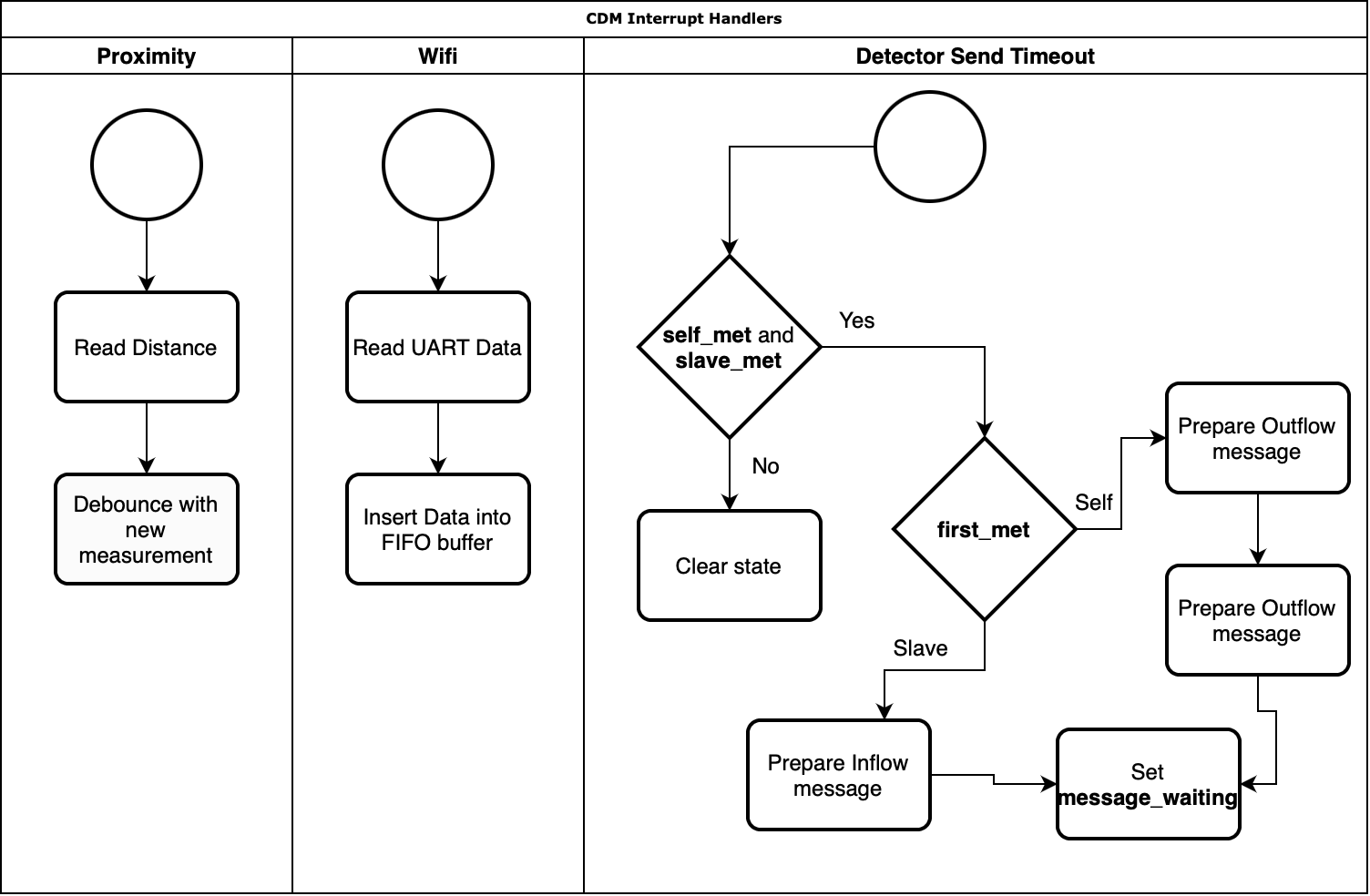
**

Figure 2: Interrupt Handler flowcharts for CDM

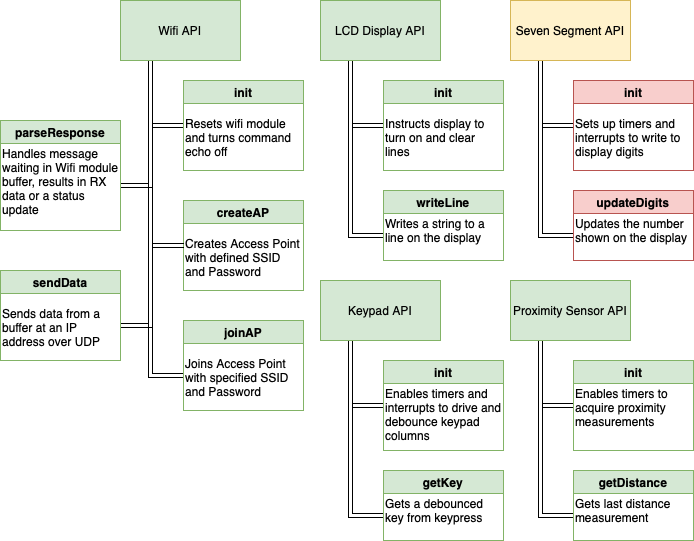
**

Figure 3: Module API Functional Block Diagram

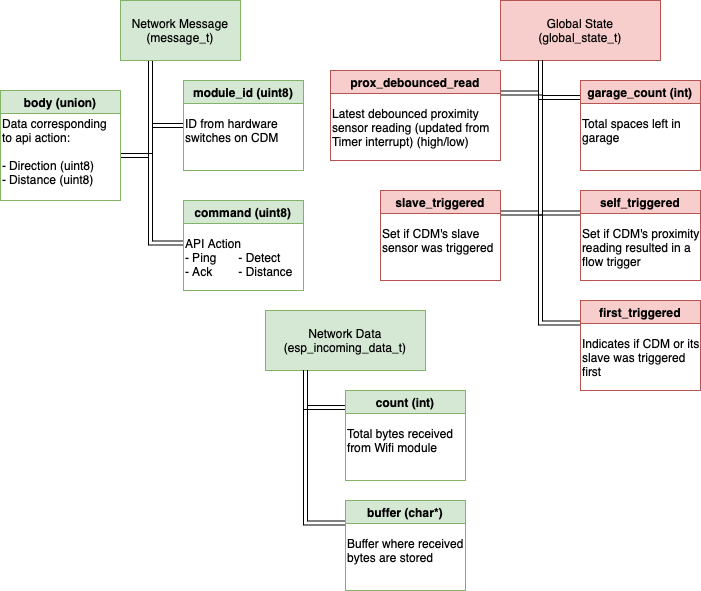
**

Figure 4: Global Data Structure Block Diagram

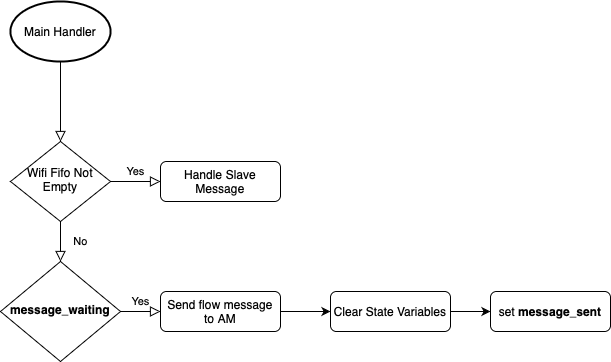
**

Figure 5: CDM Main Loop Flowchart