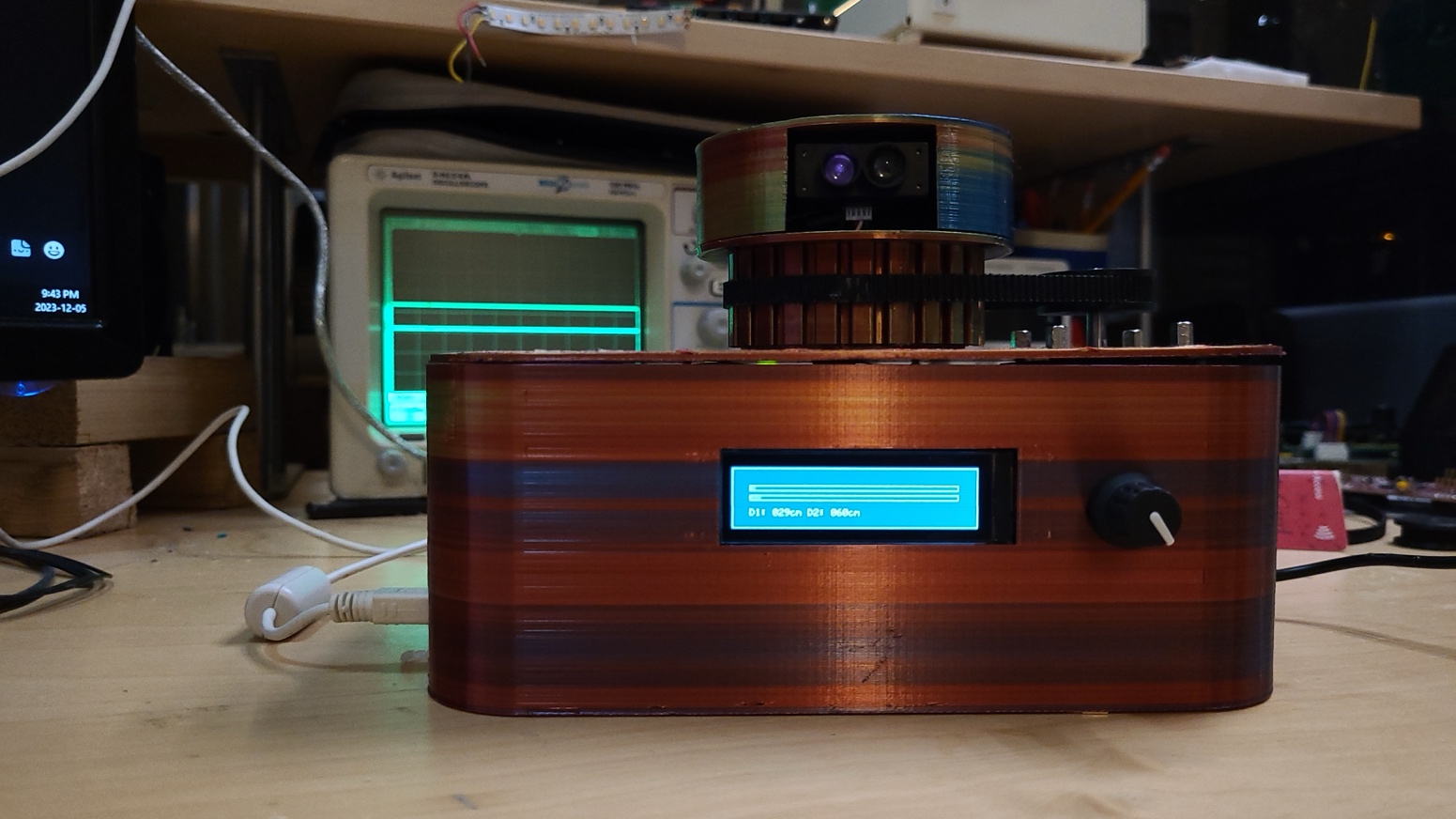
ELEX 7820: Real Time Embedded Systems  
 Lab Project:

2d LiDAR Mapping Device



Date: 12/05/2023

Prepared by:

|  |  |
| --- | --- |
| Student Name: | Student Number: |
| Myles Parfeniuk | A01022485 |
| Omar Kebedov | A01179213 |

Table of Contents

[1 Introduction 4](#_Toc152729480)

[2 Items 4](#_Toc152729481)

[3 Block Diagram of System 6](#_Toc152729482)

[4 Thread Priorities Table 7](#_Toc152729483)

[5 Semaphore and Mailbox Tables 8](#_Toc152729484)

[6 CPU Utilization Analysis 9](#_Toc152729485)

[6.1 Method 9](#_Toc152729486)

[6.2 Results 10](#_Toc152729487)

[7 Design Choices 11](#_Toc152729488)

[7.1 Hardware and Case 11](#_Toc152729489)

[7.2 Python Client 11](#_Toc152729490)

[7.3 PC and Controller Communications 12](#_Toc152729491)

[7.4 Firmware Structure 12](#_Toc152729492)

[8 Testing 13](#_Toc152729493)

[8.1 Lockup Testing/Debugging 13](#_Toc152729494)

[8.2 IR Sensor Testing 14](#_Toc152729495)

[8.3 Filter Response Testing 14](#_Toc152729496)

[8.4 Accuracy Testing 15](#_Toc152729497)

[8.5 Stepper Driver Testing 15](#_Toc152729498)

[9 Results. 16](#_Toc152729499)

[10 Conclusion. 17](#_Toc152729500)

[Table 1. Project Items 3](#_Toc152722290)

[Table 2. Threads. 6](#_Toc152722291)

[Table 3. Semaphores. 7](#_Toc152722292)

[Table 4. Mailboxes. 8](#_Toc152722293)

[Table 5. CPU Utilization. 9](#_Toc152722294)

[Figure 1. Block Diagram of system. 6](#_Toc152729471)

[Figure 2. Real Time Analysis Load Module Configuration. 9](#_Toc152729472)

[Figure 3. Load module API Calls. 9](#_Toc152729473)

[Figure 4. Step task CPU utilization data returned from load module API call. 10](#_Toc152729474)

[Figure 5. ROV classic viewer depicting tasks in our firmware during I2C lockup bug. 13](#_Toc152729475)

[Figure 6. IR detecting photo diode circuit utilized in test, and in final product. 14](#_Toc152729476)

[Figure 7. Main characteristics table of benewake TF-Mini-S LiDAR sensor, from its respective product manual. 15](#_Toc152729477)

[Figure 8. Basement suite map generated by our 2d LiDAR mapping device. 16](#_Toc152729478)

[Figure 9. Map generated by commercially available RP LiDAR system, retailing at $521.55 CAD 16](#_Toc152729479)

# Introduction

Our project is a 2-D Lidar mapping device, which transmits angle and distance values to PC over UART/COM. The distance values are measured by two I2C distance sensors, mounted on a podium 180 degrees apart, thus ensuring a faster scanning speed. The angle is determined by the number of steps to complete one full rotation of the podium. The number of steps resets once one full rotation has taken place by detecting a falling edge of an IR sensor, mounted inside the podium. Data transfer between sensors and microcontroller is facilitated via slip ring with six conductors, where I2C data pins are connected to a pair of wires each, thus ensuring a more reliable connection and less of a chance of interruption due to slip ring bounce. Complete description of the way of operation of this device will be explored in depth throughout this report. The device is fully operational, allowing its user to draw a shape of the room the device is placed into.

# Items

This section contains tables of both the mandatory and other items as outlined in the marking guideline.

Table 1. Project Items

|  |  |
| --- | --- |
| **Mandatory Items** | **Description of Implementation** |
| SYS/BIOS RTOS – meaningful use of all thread types: HWI, SWI, TSK, IDLE | HWI – Used for I2C, UART, SPI, transaction events, GPIO signals from IR photodiode and encoder.  SWI – Used for post processing of received UART data, I2C transactions, and encoder.  TSK – Used for synchronizing lidar sampling with stepper motor, as sharing distance and angle data between several different tsks responsible for updating the OLED, sending data to PC, and parsing received commands from PC.   IDLE – Used to blink a green LED (David’s favorite colour). |
| CPU Utilization | See relevant section (Sect. 6.0) |
| DSP | * Moving average for filtering number of steps per rotation of LiDAR podium. * Exponential moving average filter (EMA IIR filter) used for measured LiDAR distances. |
| Real Time Processing | Taking real time distance samples with 2 lidar sensors, matching them with the current angular position of the stepper shaft, and sending them to a PC to generate **a map of acceptable fidelity with minimal visible lag** to user. |

|  |  |
| --- | --- |
| **Optional Items** |  |
| Add-on-Boards | * LiDAR sensors: Two LiDAR sensors interfaced to via I2C. * OLED display:  16 shades of gray (SH1122 driver interfaced via SPI). * TMC2209 stepper driver: Stepper driver IC used to control stepper motor. Interfaced via GPIO. |
| Ports | * I2C Used take and read back samples from LiDAR sensors. * SCI (UART) Used to send distance and angle data to PC, as well as receive stop and start scanning commands from PC. * SPI Used to communicate with OLED display. |
| Non-trivial use of: Semaphore(s) synchronizing code | Used to post tasks in several places:   * From hardware timer generated delays (see delay\_timer.\*[[1]](#footnote-2)) * From HWIs and SWIs associated with I2C, UART, SPI, GPIO pins to synchronize tasks with events. * From other tasks to synchronize each other (i.e., the step\_task being synced with the lidar\_sample\_task) |
| Non-trivial use of: Mailboxes, Queue(s), other | Mailboxes used in several locations to send data between tasks:  - sending distance data to pc\_data\_tx\_task and oled\_display\_task from lidar\_sample\_task - sending angle data pc\_data\_tx\_task from step\_task  - sending stop and resume conditions between task\_wait and lidar\_sample\_task and step\_task |
| Standalone operation | Devices takes samples and displays them to OLED when disconnected from PC. CPU watchdog timer disabled for standalone operation. |
| Other | * 3d printed enclosure for controller and lidar sensors * Python client: Draws a map from LIDAR data received over UART, custom script/program running on a PC. * 6 hand-soldered perf-boards for mounting connectors and components. |

# Block Diagram of System

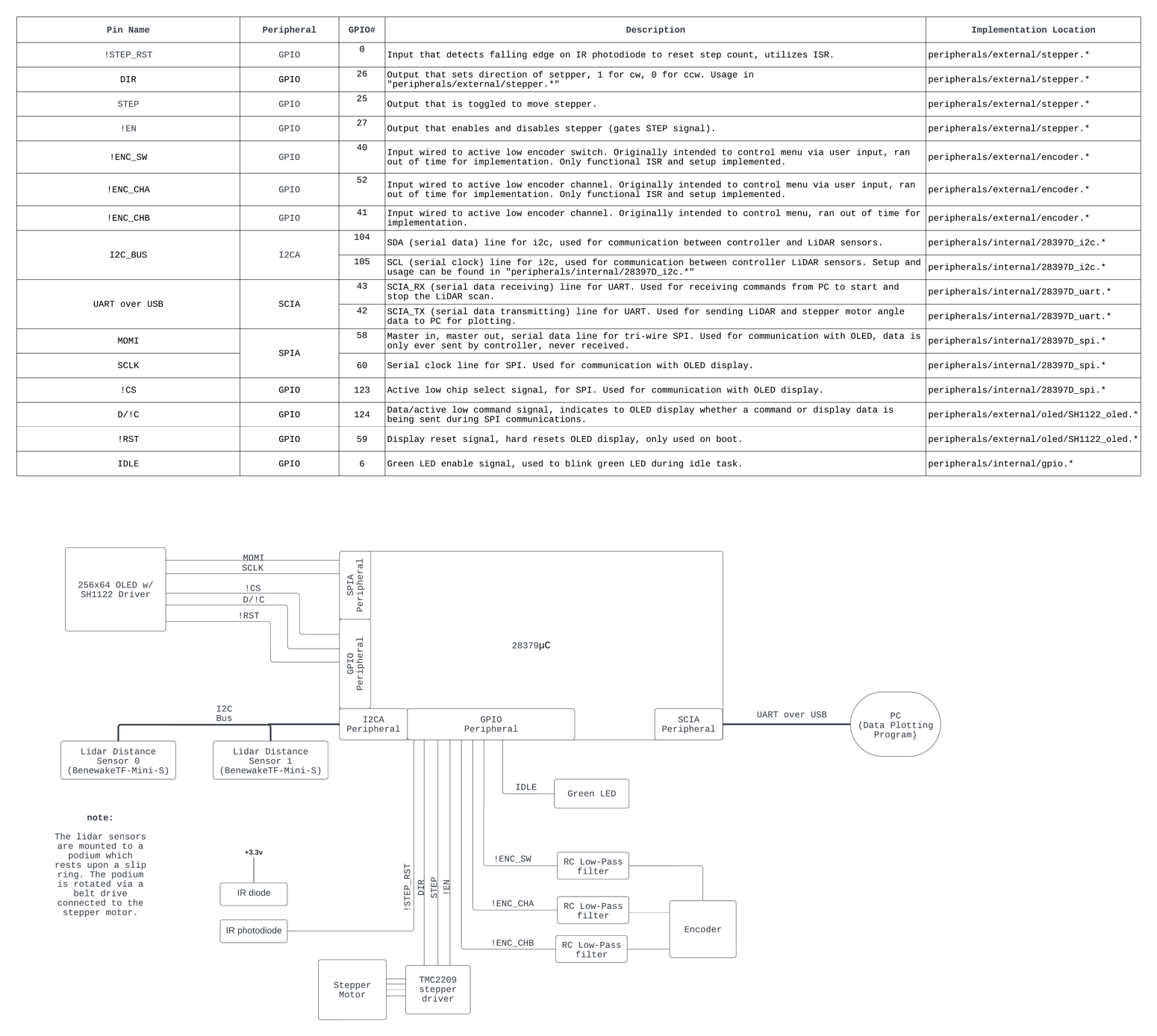


Figure 1. Block Diagram of system.

# Thread Priorities Table

The table below shows the Hwi/Swi/Tasks involved in the device operation, their respective priority and description.

Table 2. Threads.

|  |  |  |  |
| --- | --- | --- | --- |
| Thread/Function | Type | Priority | Description |
| ir\_sensor\_ISR | Hwi | N/A[[2]](#footnote-3) | ISR that triggers once IR sensor goes low (falling edge). Resets the number of steps to 0, indicating a complete rotation. |
| i2c\_handler\_ISR | Hwi | N/A1 | Checks the I2C mode (RX/TX) and posts an appropriate SWI. |
| SPI\_handler\_ISR | Hwi | N/A1 | Reads on SPI (dummy read) to clear SPIRXINT flag, and posts spi\_tx\_semaphore, releasing the task that pended it by calling spi\_transmit. |
| uart\_tx\_handler\_ISR | Hwi | N/A1 | Loads next byte into FIFO and posts uart\_tx\_sem to release which ever task pended it when calling UART transmitting function. |
| uart\_rx\_handler\_ISR | Hwi | N/A1 | Posts uart\_rx\_SWI. |
| encoder\_ch\_a\_ISR | Hwi | N/A1 | Triggers on falliong edge of encoder channel A. Posts encoder\_SWI. |
| encoder\_switch\_ISR | Hwi | N/A1 | Triggers on falliong edge of encoder switch. Posts encoder\_SWI. |
| i2c\_rx\_handler\_SWI | Swi | 10 | Services I2C receiving events, posted by i2c\_handler\_ISR HWI. |
| i2c\_tx\_handler\_SWI | Swi | 9 | Services I2C transmitting events, posted by i2c\_handler\_ISR HWI. |
| uart\_rx\_SWI | Swi | 8 | Checks any char sent by PC, posts task\_wait task to run via semaphore if char received corresponds to wait or stop command. |
| encoder\_SWI | Swi | 7 | Empty shell that re-enables interrupts that were disabled by switch and ch\_a ISRs. The planned implementation here was to send events to the oled\_display\_task to create a menu controlled by the encoder, but it was never completed. |
| step\_task | Tsk | 8 | Zeros the stepper on boot, takes 10 steps, mails motor angle to pc\_data\_tx\_task, and posts semaphore for lidar\_sample\_task to run when its respective semaphore is posted by lidar\_sample\_task. |
| pc\_data\_tx\_task | Tsk | 7 | Waits to be mailed angle and distance data, transmits the data to the PC over UART after receiving. |
| wait\_task | Tsk | 5 | Stops all scanning tasks and waits on further command from PC, resumes scanning tasks if start command is sent. |
| lidar\_sample\_task | Tsk | 4 | Takes a sample from each LiDAR sensor, filters the sample, mails it to tx\_data\_task, posts semaphore for step\_task to run, and pends semaphore pended by step\_task. |
| oled\_display\_task | Tsk | 3 | Sends the data to display to OLED. |

# Semaphore and Mailbox Tables

The table below lists all semaphores used for the device operation and their description.

Table 3. Semaphores.

|  |  |
| --- | --- |
| Semaphore | Description |
| lidar\_sample\_delay\_sem | Used to generate delay between samples, posted by delay\_timer\_ISR after calling delay\_task\_ticks (see delay\_timer.\*) |
| step\_to\_sample\_sem | Wait for step to complete before sampling. |
| sample\_to\_step\_sem | Wait for sample to complete before stepping. |
| wait\_to\_lidar\_sem | Freeze lidar task after posted by wait\_task. |
| wait\_to\_step\_sem | Freeze step task after posted by wait\_task. |
| uart\_tx\_sem | Waiting for TX semaphore posted from HWI OR for 5 ms in case of failure (UART times out). |
| i2c\_rx\_sem | Pended by task that called i2c receiving function, posted by i2c\_rx\_handler\_SWI after transaction completed. |
| i2c\_tx\_sem | Pended by task that called i2c transmitting function, posted by i2c\_tx\_handler\_SWI after transaction completed. |
| spi\_tx\_sem | Synchronizing SPI transmitting events, pended by task calling spi transmit function, posted by SPI\_handler\_ISR HWI after byte has been sent. |
| step\_delay\_sem | Generate delay between stepper steps, posted by delay\_timer\_ISR after calling delay\_task\_ticks (see delay\_timer.\*) |
| task\_resume\_sem | Resume all the scanning tasks. |
| task\_wait\_sem | Freeze all the scanning tasks. |
| zero\_sem | Pended by oled display task during stepper zeroing phase to ensure zeroing screen is displayed until samples being taken. |

The table below lists all mailboxes used for the device operation and their description.

Table 4. Mailboxes.

|  |  |
| --- | --- |
| Mailbox | Description |
| distance\_to\_oled\_mailbox | Passes the distance to oled\_dispaly\_task from lidar\_sample\_task. |
| distance\_to\_pc\_mailbox | Passes the distance to pc\_sending\_task from lidar\_sample\_task. |
| angle\_to\_pc\_mailbox | Passes the angle to pc\_sending\_task from step\_task. |
| stop\_lidar\_task\_mailbox | Passes Stop argument to lidar task from wait\_task. |
| stop\_step\_task\_mailbox | Passes Stop argument to step task from wait\_task. |

# CPU Utilization Analysis

## Method

To calculate the CPU utilization, we opted to use sys/bios’ provided tools as our tasks involve multiple delays relying on semaphores, and branching dependent on external events, because of this, using a GPIO pin would be a near Sisyphean task.

The TI sys/bios load module was configured via the XGCONF GUI as follows:

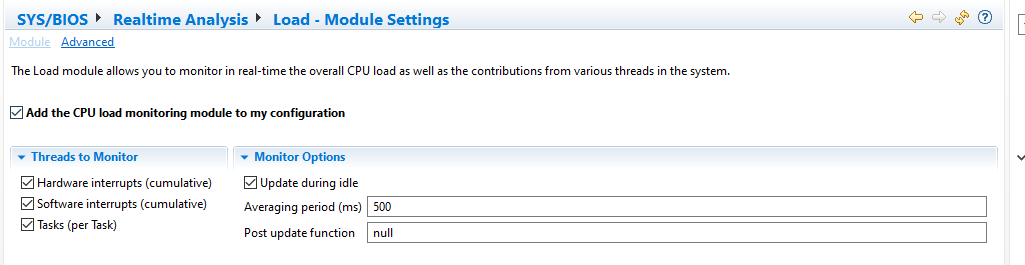


Figure 2. Real Time Analysis Load Module Configuration.

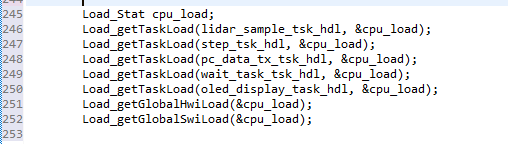
Next APIs were called from the ti/sysbios/utils/Load.h module to determine the execution time of the individual tasks, as well as all the HWIs combined, and all the SWIs combined:  
  


Figure 3. Load module API Calls.

Through the debugger, the return values within the Load\_Stat struct were examined to build the chart in 6.2. An example calculation for the step\_task and its relevant data is shown below:

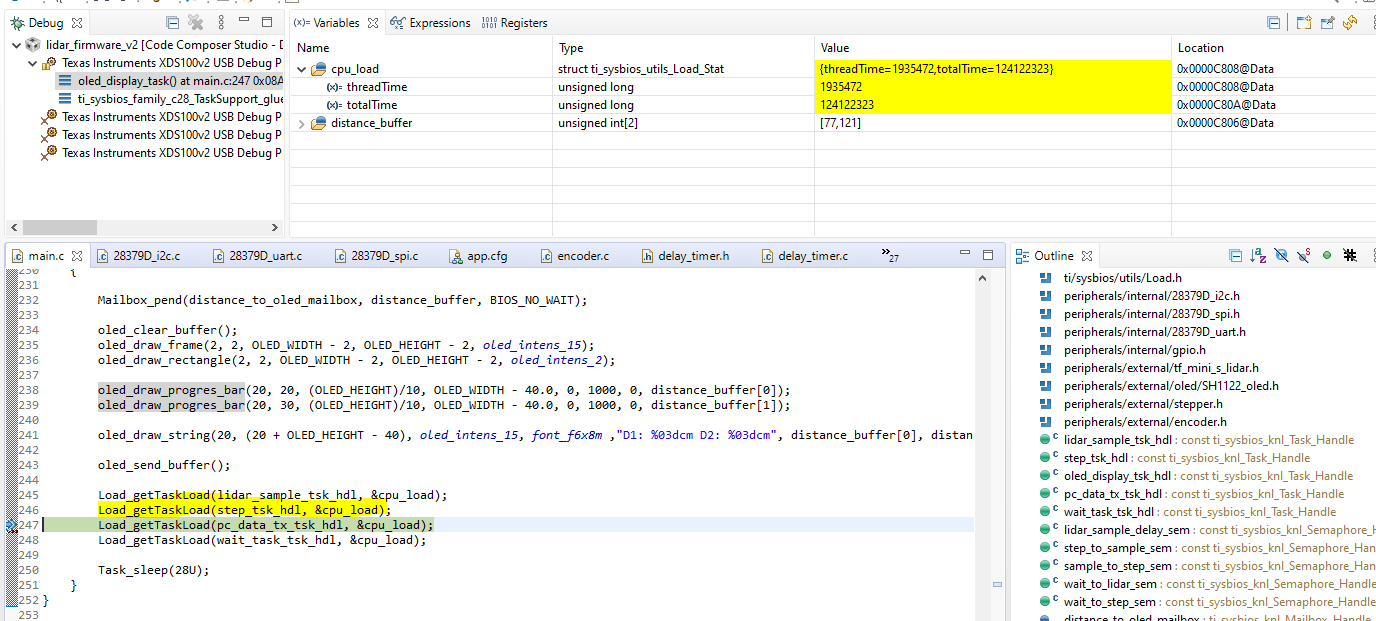


Figure 4. Step task CPU utilization data returned from load module API call.

From the API documentation located at [module ti.sysbios.utils.Load,](https://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sysbios/6_53_02_00/exports/bios_6_53_02_00/docs/cdoc/ti/sysbios/utils/Load.html#get.C.P.U.Load) these functions return a time duration over which the measurement was done, and the time spent in the respective task in system clock ticks. Knowing that our system clock is 200MHz:

## Results

Table 5. CPU Utilization.

|  |  |  |
| --- | --- | --- |
| Task | Execution time (ms) | % utilization (time) |
| step\_task | 9.68 | 1.56 |
| pc\_data\_tx\_task | 12.29 | 1.98 |
| wait\_task | 0.00[[3]](#footnote-4) | 0.003 |
| lidar\_sample\_task | 3.36 | 0.54 |
| oled\_display\_task | 309.24 | 48.61 |
| Global Hwi[[4]](#footnote-5) | 236.23 | 37.14 |
| Global Swi4 | 3.29 | 0.52 |

# Design Choices

## Hardware and Case

All internals are put in a designed from scratch enclosure, thus the user will not be able to see or touch them. This ensures the system durability and integrity. The enclosure is composed of a base containing most of the components, and a rotating podium holding the two lidar sensors.

Stepper gear has been printed to facilitate physical link between the stepper and the podium. The gear drives a belt which is wrapped around the podium.

A 12V NEMA-17 stepper motor with a 0.9deg per step spec was chosen as it provided a large amount of torque to ensure minimal slippage, and a finer resolution than the more commonly available 1.8deg models.

To keep track of when the stepper motor has been zeroed, an IR photodiode encased in a straw and tube of heat shrink is mounted within the enclosure underneath the rotating podium. The rotating podium has an IR emitting diode which causes the IR photodiode circuit output to fall low for a split second as it passes above.

Since a wired connection between the rotating podium and stationary base is required for communication between the controller and LiDAR sensors, a slip ring was used. Without the slip ring the wires would become tangled.

On the front of the device, an OLED screen serves a purpose of the real-time display of values (distances) as well as a zeroing state indicator. Additional functionality was planned, but not implemented due to lack of time.

An encoder to the right from the OLED was supposed to serve as a physical interaction between the user and the device, however due to time constrains, its implementation was not completed. Despite that, the respective GPIO pins are configured and some associated ISR routines are fully functional.

Lastly, to maintain solid connections and ensure stability of signals, a total of 6 custom soldered perf-boards boards with JST connectors was added to facilitate connections between the controller and external peripheral. These perf boards also served as a place to mount pullup & pulldown resistors where necessary, encoder filtering, the photo diode circuit, and the stepper driver.

## Python Client

For user interaction, a python-based script to plot the received sensor data as a map, as well as a simple interface to start and stop mapping was implemented. Once the device is connected to the PC over a USB cable, and the script is launched. The mapping process will begin automatically by default. User has an option to stop the sampling at any point of time.  
  
In order to ensure the map was drawn without lag, the python client is multi-threaded, using one thread to read data from the comport, another to plot the data, and another to detect GUI button presses.

## PC and Controller Communications

The communication between the PC and the device was implemented by using UART protocol. The baud rate was chosen to be 1MHz, the frame with no parity bit and one stop bit, and 8 data bits.   
The communication is full duplex, the controller and PC and both send to each-other at the same time.  
  
For receiving, some thought was put into writing firmware for a command parser that starts and stops scanning after receiving a command from the python client. To simplify things and avoid the use of string comparison functions within the firmware, we decided to make each command a single character delimited by the new line key ‘\n’.

On the sending side of things, the firmware has been written such that all 16 bytes of the RXFIFO can potentially be utilized, increasing efficiency by offloading processing to the SCIA peripheral. Each packet of data sent from the controller to the PC contains 3 numbers, a four-digit number representing the current motor shaft angle in radians\*1000, followed by two three-digit numbers representing the filtered measurement from each lidar sensor corresponding to the motor shaft angle.

## Firmware Structure

In order to keep the firmware clean, it was decided to divide all the peripheral functionality into separate .c and .h modules, located in the peripherals directory of the project. The main.c file serves solely as a place to launch the main tasks used within the firmware and hold their definitions.   
  
Mailboxes were decided to be used over global variables to transfer data between tasks. This was done for multiple reasons. The task sending the data to the PC must ensure it has received a new sample before sending it, mailboxes can be pended which eases this process; furthermore, excessive use of global variables in a large codebase is generally bad practice. It can often lead to race conditions and corruption.  
  
Lastly, to create a re-usable and efficient blocking delay without the use of Task\_sleep, a delay function was developed based off a hardware timer and a passed semaphore, the implementation of this can be seen in delay\_timer.\*.

# Testing

Multiple tests and hours of debugging occurred, but this section will only cover some of the highlights.

## Lockup Testing/Debugging

In early revisions of the firmware, it was noted that the device would occasionally stop taking samples and lock up. To find the source of this issue the ROV classic viewer was used to view the state of all the tasks. It was noted that all tasks appeared to be blocked, only the OLED task was executing, which was confirmed with the debugger.

A screenshot of a computer program

Description automatically generated

Figure 5. ROV classic viewer depicting tasks in our firmware during I2C lockup bug.

Further investigation determined the tasks would only enter this state occasionally after I2C transactions in the sampling task. Using the register viewer it was determined that the status flags were not being cleared in some cases, which caused the related HWI to never fire, meaning the sample task never had its semaphore posted to continue.   
  
The solution was clear the relevant flags before re-enabling interrupts in the i2c transmitting and receiving functions, by writing to the IRS bit of the I2CMDR register.   
  
To confirm the issue was resolved, the device was left to run for periods of 1hr on three separate occasions, the lockup bug failed to be replicated, indicating its source has most likely been found and eliminated.

In summary, on this issue, and many others throughout this project, the ROV classic viewer, register viewer, variables window, and expressions window were all used effectively in debugging.

## IR Sensor Testing

A diagram of a voltage

Description automatically generatedAfter soldering up a perf-board containing the IR photo diode circuit depicted below:

Figure 6. IR detecting photo diode circuit utilized in test, and in final product.

The Vout signal was measured on a scope as an IR emitting diode was waved above it. It was quickly noticed that some form of shielding would be required. Since it was desired to trigger an interrupt based off the falling edge of this signal, a sharp edge was necessary. Instead, the voltage on the scope gradually dropped and raised as the emitter passed the photodiode.   
  
To fix this issue the photodiode was encased at a depth of a few millimetres within a straw and wrapped in heat. Doing this limited its visibility of the emitter until it was directly over-head, creating a much sharper edge.  
  
This was confirmed on the scope. After being mounted onto the enclosure and tested with the IR emitter on the rotating podium, it was found that its fall time was less than 10ms, satisfactory for our needs.

## Filter Response Testing

To adjust the alpha constant and averaging window size of the filter being used for the measure distances, both were tweaked until a static image could be represented with satisfactory fidelity.

This was done by placing the device into a rectangular bin with rounded corners and running a scan. The plotted image was compared, and the constants adjusted until it visually matched the shape of the bin.   
  
To test filter responsiveness, objects were placed in front the scanning path at distance much closer than background objects. This allowed us to observe how the filter reacted to sudden changes. It was then determined that alpha needed to be reduced, as too much smoothing was being applied, leading to the close object being drawn incorrectly as an arc.

## Accuracy Testing

A table with text and symbols

Description automatically generatedTo test accuracy, the device was again placed into the same rectangular bin, points were measured with a ruler from the LiDAR sensors to the edges of the bin in multiple locations, then compared to the map drawn by the python client. It was determined that at close ranges, the device had no deviations greater than 2cm, which is well within spec according to the TF-Mini-S LiDAR sensor [data sheet](https://cdn.sparkfun.com/assets/8/a/f/a/c/16977-TFMini-S_-_Micro_LiDAR_Module-Product_Manual.pdf):

Figure 7. Main characteristics table of benewake TF-Mini-S LiDAR sensor, from its respective product manual.

Next, to test the device at greater distances, a tower of cardboard boxes was constructed so the sensor could ping the walls of a room without being impeded by objects like furniture. Again, the same process was repeated, except using a measuring tape at a scale of several meters. It was determined the sensors were again within spec, with no deviations greater than 20cm at distance between 6 to 8m.   
  
Admittedly, our measurement methods were crude, and could be improved by comparing against a professional LiDAR product, or a floorplan of the room scanning was performed in, if we had access.

## Stepper Driver Testing

To facilitate the movement of the stepper, we need to correctly configure the interaction between the stepper driver and controller. The TMC2209 was selected as a driver due to its simplicity and ease of operation. It comes with two control interfaces, one via GPIO pins, another via a UART bus. For simplicity, we opted to interface via GPIO pins. The control is performed via an active low enable pin, a direction pin, and a step pin, that moves the motor a single step for every time it is toggled.   
  
After writing the firmware to toggle the step pin according to a blocking delay, the signal was observed on scope to ensure the high and low period were as expected. The direction pin was observed to be set correctly as high, and the enable as low.   
  
Despite verifying all the signals, we embarrassingly managed to fry FOUR stepper drivers due to errors made during soldering, forgetting a bypass cap, and accidentally shorting the 12v rail while probing with a scope (twice).

This was a lesson learned for both of us, look three times, solder once. After the paranoia had set in, for the 5th and final attempt at creating a perf board, we individually tested every contact for shorts between soldering traces. This was done using the continuity test on a multi-meter. We also mounted female headers instead of soldering the driver directly to the board in case more embarrassing stepper incidents occurred, such that the driver could easily be replaced.

# Results.

A graph with a green outline on it

Description automatically generatedThe final product turned out to be a user-friendly albeit much less refined version of other commercially available LiDAR systems such as the “RP LiDAR”. A comparison can be seen below, the first screen shot being a map generated of a basement suite by our device, followed by a screen shot from a YouTube video [here](https://www.youtube.com/watch?v=Qrtz0a7HaQ4) of the commercially available RP LiDAR performing a scan of somebody’s office. Note that the RP LiDAR is being used for comparison, as it retails for $521, it most likely has a similar BOM cost of approximately $100.

A screenshot of a computer

Description automatically generatedFigure 8. Basement suite map generated by our 2d LiDAR mapping device.

Figure 9. Map generated by commercially available RP LiDAR system, retailing at $521.55 CAD

While we cannot comment on an accuracy comparison, at least visually, it yields similar results to the *real deal*.

Considering its limited functionality, there is a lot of room for improvement, thus adding more features, such as an object counter, surface length calculator and more would be desirable. The firmware could also be improved to be much less resource intensive.

Nonetheless, it achieves the key milestones of this project and is able to process the data in real time with sufficient accuracy and reliability.

# Conclusion.

1. A period followed by an asterisk appearing after a name like so “name.\*” anywhere in this document can be interpreted as a combination of a header and source file, i.e. name.c and name.h. [↑](#footnote-ref-2)
2. Please note, that priority selection for HWI is hardwired, and not selected by the user and thus has been omitted. The priorities of the interrupts used can be referenced Table 3-3. of the TMS320F2837xD t.ref manual if desired. [↑](#footnote-ref-3)
3. Wait task was measured as zero as the stop/start scan commands were not utilized during testing, thus, it never ran. [↑](#footnote-ref-4)
4. SYS/BIOS’ Load APIs do not include a method of measuring SWIs or HWIs individually, only the total summation of their execution times. [↑](#footnote-ref-5)