**Fish Sense - Low Power System**

Milestone Report - 237D

Rahul Polisetti, Mohana Seelan, Kyle Yang

### MVP, Milestones and Deliverables from the Project Specification Document:

* MVP: Our minimum viable product for FishSense would consist of benchmarking the power draw in different states and creating a low power mode.
* Deliverable 1: Find out the necessary power goals given the current FishSense system and researcher use cases - Week 5
  + **Milestone 1.1: Contact researches to find out how long and often FishSense needs to be active and collecting data from the RealSense Camera**
  + **Milestone 1.2: Benchmark the current power consumption in FishSense’s active mode**
  + **Milestone 1.3: Given the frequency of active mode, calculate the maximum power consumption that low power mode can have**
* Deliverable 2: Create a low power mode for the NVIDIA Jetson - Week 6 & 7
  + **Milestone 2.1: Turn the NVIDIA Jetson on and off at the researcher-given sampling frequency and measure the power consumption. In this case, “off” is the low power mode**
  + **Milestone 2.2: Create a low power mode on the NVIDIA Jetson by turning off cores slowing down clock speed, configuring different RealSense modes**
  + **Extra Milestone 2.3: Try to make active mode more low-power by changing RealSense settings**
* Deliverable 3: Assess how well low power mode does - Week 8
  + **Milestone 3.1: Measure low power mode consumption and state change power consumption**
* Extra Deliverable 4:
  + Connect the STM32 IO board to the Jetson and have that turn on and off the active mode on the Jetson
  + Measure power consumption after configuring STM IO Board

Progress on MVP, Milestones, and Deliverables:

* MVP: The first half of our MVP, the benchmarking of power consumption in different states, is complete. Further explanation and demonstration of the benchmarking is below. We are currently working on the second half of our MVP which is the low power mode for the FishSense system. After getting the benchmarking results from Milestone 1.2 and calculating the maximum power consumption needed in low power mode from Milestone 1.3, we have found that achieving a low power solution with the parameters set in Deliverable 2 to be nonviable. We replaced Deliverable 2 with our Extra Deliverable 4 from our Project Specification to be part of our final MVP.
* Deliverable 1 - Week 5:
  + **Milestone 1.1: Contact researches to find out how long and often FishSense needs to be active and collecting data from the RealSense Camera** - Based on our project mentor, Nathan’s, guidance, we postponed this milestone to the last week of the project. This is because if we were to contact researchers at the beginning of the project , they would be giving us a long list of either unreasonable or unviable requirements which we would not know how to handle before we implement the MVP. These requirements would be essentially useless for us at the moment. Our current goal is then to get the MVP up and running and present to the researchers what our system is capable of, and give them a list of parameters, such as wake up time, cycle time, etc. that we would be able to change for them.
  + **Milestone 1.2: Benchmark the current power consumption in FishSense’s active mode** - This milestone has been completed, and results have been presented in our oral project update.

The Excel sheet containing our data is below:

<https://ucsdcloud-my.sharepoint.com/:x:/g/personal/rpolisetti_ucsd_edu/ETujqVTSuNJGhAEdr-AKgywBJEAt5spWVZOftMMrlYUgVg?e=SQh4bb>

The cleaned and plotted power data is below:

<https://docs.google.com/spreadsheets/d/1WFcS6u2uPOQZOVs0XoIG-RhzFOwcmA-VoYtzn1EzCCU/edit#gid=1351120054>

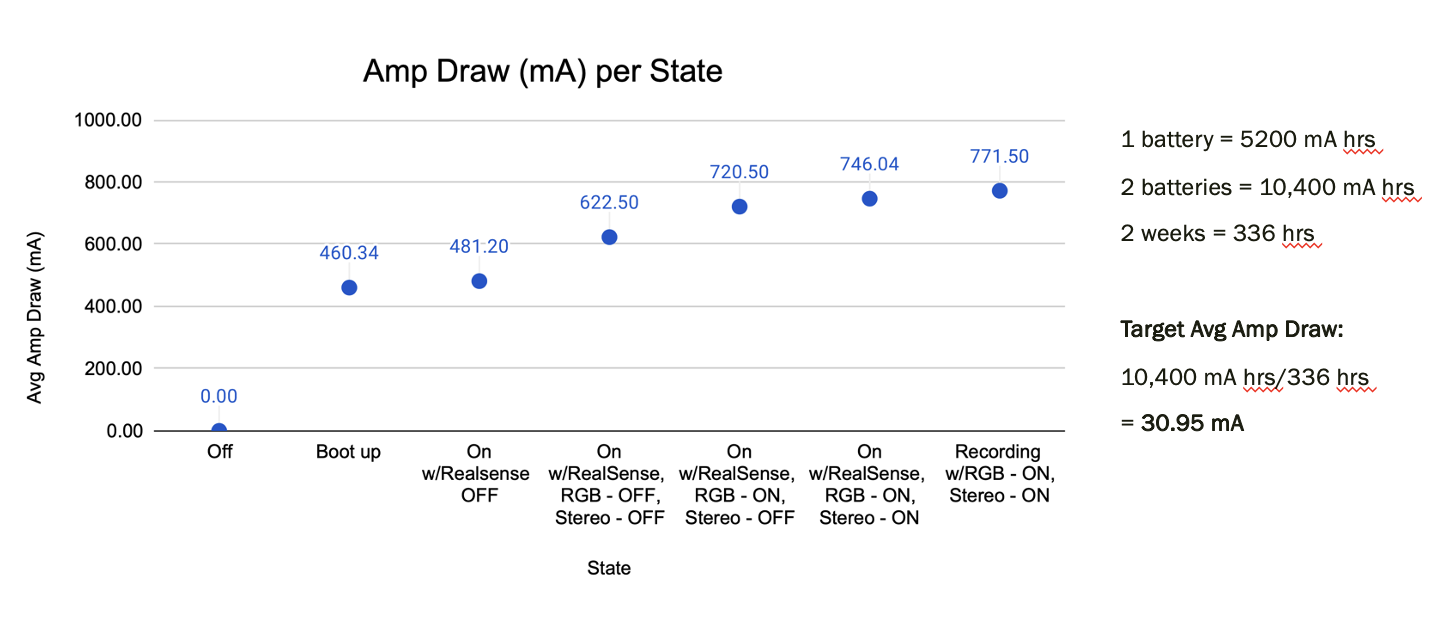
A video demo about our experimental setup is below:

<https://photos.app.goo.gl/FdP81VfHANU9sNabA>

* + **Milestone 1.3: Given the frequency of active mode, calculate the maximum power consumption that low power mode can have** - Based on the current FishSense setup and our power consumption figures from the previous Milestone, we have calculated the maximum power consumption that low power mode can have. Based on the results we have found that the parameters placed by us in Deliverable 2, that is to create a low power mode purely using the NVIDIA Jetson alone, is not going to work. The NVIDIA Jetson board cannot realistically be placed in a low enough power state to hit our targeted current numbers, and cannot be turned off on its own since it does not have an internal RTC to start itself up again.

The calculations we did were explained in our presentation for the oral project update, which is below for reference.

As can be seen in the slide below, the target average current draw if we want the system to last for 2 weeks permanently in a low power state should be around 31mA, whereas the current draw for the Jetson alone is 460mA in idle. Even without considering that we have to switch to a high power state, this setup is obviously not feasible without completely powering off the system.



* Deliverable 2: Create a low power mode for the NVIDIA Jetson - Week 6 & 7
  + **Milestone 2.1, 2.2**: As discussed above, the method of achieving low power mode in this deliverable is unfeasible. We have thus switched this Deliverable out with our Extra Deliverable 4 that we proposed in our Project Specification Document.

This method is to program directly onto an alternate STM32G0 board the capability of turning on and off the main system completely, which includes the RealSense, NVIDIA Jetson TX2, SSD, and USB hub. The board utilizes very lower-power to

run a timer, which can be interfaced with the system's existing power-IO board. Our power goal for this implementation is to draw less than 30 milliamps to power the STM board while the FishSense camera is in sleep mode. This would ensure enough power is available during the full duration of its deployment.

* Extra Deliverable 4 (Replaced Deliverable 2): Use the RTC onboard the STM32 to turn off the FishSense System completely- Week 6 & 7
  + The milestones for this deliverable have been updated and have been split up into more detailed steps after we pivoted to using the STM32 board. These details can be seen in the next section on Updated Milestones and Deliverables.
  + **Milestone 4.1: Create a basic RTC timer using the STMCube IDE and flash to the STM32G0 for proof of concept testing -** We needed to start learning the STM32 development environment along with creating a proof of concept by flashing a LED light on the STM32 based on an RTC timer interrupt that we programmed. The video link below shows our program running on the STM board in debug mode. The program sets up the RTC on the STM32, switches to the LEC oscillator crystal, and sets up the onboard LED4 to light up after 25 seconds have passed. The program sets on program start, the RTC to be 12:00:00, and then starts counting.

Video of RTC timer proof of concept running on an STM32:

<https://drive.google.com/file/d/1suZ16OBKDzy11mhQbb_P4-Fl9UBGymdC/view?usp=sharing>

* Deliverable 3: Assess how well low power mode does - Week 9
  + **Milestone 3.1: Measure low power mode consumption and state change power consumption** - This is a week 9 deliverable and will be started after the STM32 board has been integrated into the Low Power System.

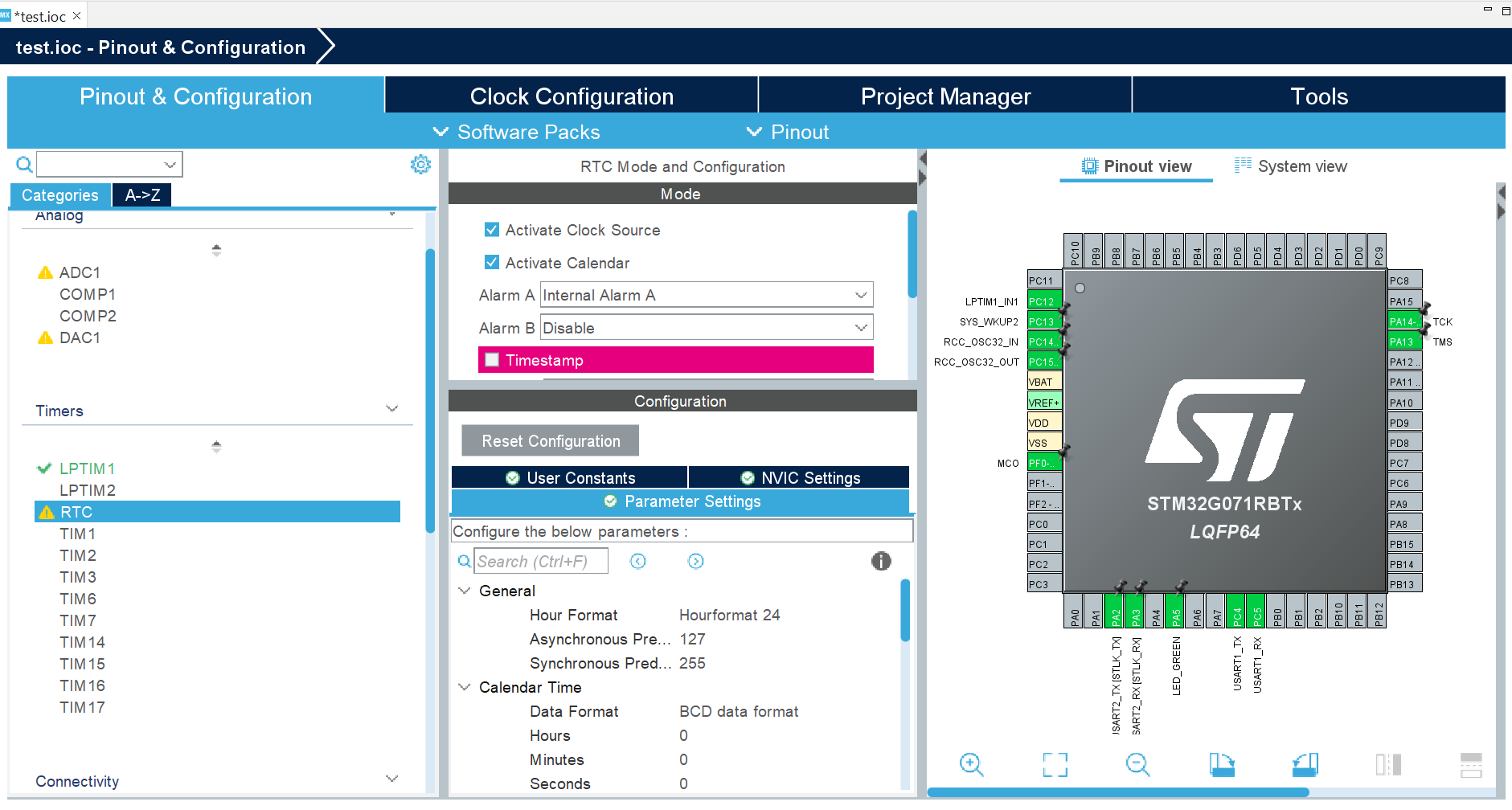
Updated Milestones and Deliverables:

* Deliverable 1: Program the STM32 to act as an external timer with an RTC, Priority 1
  + Scheduled Deadline: 5/24
  1. **Milestone 1.1: Read documentation on how to create an RTC time on the STM32 (In Progress)**
     1. [STM32G0 HAL Layer documentation](https://drive.google.com/file/d/1hcKkOyquUUFsUSfPGr3paL8llkbvs7I4/view?usp=sharing)
     2. [STM32G0 MCU Reference Manual](https://drive.google.com/file/d/1D-xi0UJZoFb36BAIgmKhAljadXhTq_ri/view?usp=sharing)
  2. **Milestone 1.2: Create a basic RTC timer using the STMCube IDE and flash to the STM32 to test that it works (DONE)**
  3. **Milestone 1.3: Create a low power RTC timer using the HAL library in a .ioc file.**
     1. Set the LSE oscillator on the STM
     2. Set the time correctly
     3. configure GPIO to supply power to carrier board on PD9
        1. custom carrier board already designed by Nathan (should arrive around Tuesday)
        2. carrier board will bypass the power specs for power io to cut and supply power to TX2
     4. Flash to the STM
* Deliverable 2: Test that the STM32 is supplying power during timed intervals, Priority 2
  + Scheduled Deadline: 5/28
  + **Milestone 2.1: Connect the STM32 to the custom power IO carrier board and time when power is on and off using power supply** 
    - Waiting to receive carrier board from Nathan on 5/24
* Deliverable 3: Set up communication between the TX2 and STM32, Priority 1
  + Scheduled Deadline: 5/24
  + **Milestone 3.1: Figure out how the TX2 can communicate with the STM32 and modify the current firmware code to do so**
    1. TX2 tells recording to stop, unmounts SSD
    2. send timing data to STM32, so it can resume timer
    3. STM32 or TX2 cuts power to TXT2
* Deliverable 4: Connect TX2, custom carrier board, and STM, Priority 2
  + Scheduled Deadline: 5/30
  1. **Milestone 4.1: Test that TX2 and STM32 are communicating at correct time intervals by observing when the TX2 turns on and off with the power supply**
  2. **Milestone 4.2: Test that system is also recording during on interval**

Team Member Responsibilities:

* Milestones completed: **Nvidia Jetson benchmarking:** Benchmarking was done on the power draw of the Nvidia Jetson, using a voltage meter and stopwatch. Calculations were done to understand the average power draw from the Jetson and the allowable power consumption from the batteries during the recording on and off states.

Thus far, STM integration is being done and we have begun research and experimentation with STM32 and programmed firmware in C through the GUI. A snapshot of the GUI can be seen below:



Figure[1]: Snapshot of GUI for STM32CubeIDE

* Milestone: **Implement C-code in STMCubeIDE (Kyle, Rahul)**. The board runs similarly to an Arduino and STM c-code commands are written into the infinite while (*while(1))* loop embedded in the code. A list of functions and commands is found in the documentation: “Description of STM32G0 HAL and low-layer drivers” [1]. Macros are included which will be used to specify specific interrupts (such as peripheral interrupts) and clock timers. Clocks mentioned in the HAL system peripheral description are controlled by two main functions: “HAL\_RCC\_OscConfig” and “HAL\_RCC\_ClockConfig”, the prior of which passes in the argument “RCC\_OscInityTypeDef \*RCC\_OSCInitStruct” and the latter passes in “RCC\_ClkInitTypeDef \*RCC\_ClkInitStrcut” and “uint32\_t FLatency”. GPIO’s are included which control the structure fields pin, mode, pull, and speed, each of which include possible values in the configuration. Mode includes submodes, labeled GPIO modes, external interrupt modes, and external event modes. We anticipate using the external interrupt mode in our low-power configuration since the interfacing with the NVIDIA Jetson will be an external interrupt. We will implement several other structure fields into the c-code such as HAL\_RCC\_GetSysClkFreq.
* Milestone: **Data fields (Rahul, Mohana).** Implementable data fields that could be changed to fit the functionality of the low-power mode include the RTC\_AlarmTypeDef, defined in the program stm32g0xx\_hal\_rtc.h file included with the IDE. Data fields that may have influence are RTC\_TimeTypeDef AlartmTime, uint32\_t AlarmMask, unit32\_t AlarmDateWeekDaySel, and uint8\_t\_AlarmDateWeekDay, and are also included which could be used alongside the code written in the infinite while to adjust parameters of the low-power system.

Various HAL drivers are explained in the documentation and a usage model is included that outlines the functions of each .c code embedded in the program. The schematic is shown as follows:

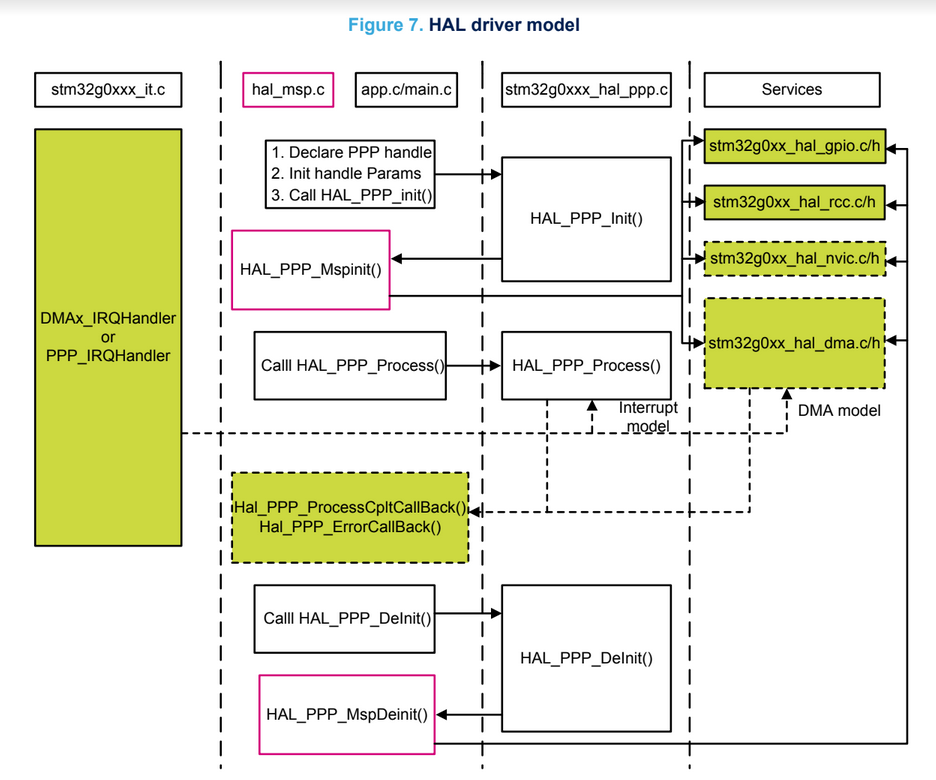


Figure [2]: Hal driver schematic

Mainly the main.c file will be written into, but other c-code files are included.

* Milestone: **Nvidia and STM32 interface (Mohana)**: Another milestone to complete is to test the RealSense with the interfaced STM board to observe the interrupts from the board. This will be done to see whether the STM interfacing works. An important task would be to benchmark the STM32 for its power consumption in comparison with the Nvidia. The comparison between the two is an important distinction as it illustrates the benefits of an additional board. Assigning interrupt tasks to a separate component and observing its benefits highlights key progress in the design iteration. Another milestone would be to include an STM32 with the FishSense module to test interrupts in real-time during real underwater deployments, to capture video footage at greater lengths of time.

**References:**

1. [STM32G0 HAL Layer documentation](https://drive.google.com/file/d/1hcKkOyquUUFsUSfPGr3paL8llkbvs7I4/view?usp=sharing)
2. [STM32G0 MCU Reference Manual](https://drive.google.com/file/d/1D-xi0UJZoFb36BAIgmKhAljadXhTq_ri/view?usp=sharing)