TODO

Sept 10 – Wrote drivers? And test cases to prove functionality of FRAM module for STM32 using HAL library and datasheet to send necessary opcodes and memory addresses(8 bit bytes) VIA SPI

Sept 12 – Developed custom robot framework keyword(function) to compare signal values within a given threshold of expected value

Sept 13 – Setup Mypy and pylint to detect case errors and type hinting and checking

Sept 16/18 – Setup python libraries and functionality for NI4322 DAQ card, added in custom data checking/comparison functions, added in reset functionality for all cards to fix problem for unknown state/unresponsive when turned on

Sept 19th- Setup python script to parse excel file into JSON file config

Sept 23rd **Meeting –**

Work has been fun with the ATE, great mentorship and team. I am open to working more so with that, honestly great help available.

My internship at the end of the day is an Embedded SWE in which up until now I have almost not touched, coding in C for microcontrollers. I want to improve my skills in this side of things as this is where I want to pursue my career.

Understand that it is a concern of company resources, but I believe there should be a balance.

If you believe I do not have the necessary skillset then I can work on skills outside of work if you can tell what it is that needs to be improved.

Oct 2nd – Worked on Implementing a CRC check on the hardware for the STM32H7 series. Profiled different programs for peak runtime optimization using an Occilscope. Had to profile to ensure that the program ran during time constraint setup section.

Performed optimzations

Oct 5th – Developed python script to develop CRC algorithm to calculate CRC value individually of the hardware module to ensure image is valid.

**Did this because the goal was to compare a known CRC value for a known section of the FLASH memory (.bin file contents) and then take that CRC value and store it in the FLASH using a post build command.**

Oct 6th – Tried to troubleshoot edge cases for the CRC word engine by rewriting manual HEX values, reading and writing HEX Values in the .bin file.

Performed program optimization to handle edge cases of when the flash memory contents were !% 4, resulting in a ~15 ms decrease, 36ms to 20 ms by allowing the use of modifying CRC engine to run at words(4 bytes) computations instead of 1 byte computations. By handling edge cases by taking known data and pad with known buffer instead of having the Hardware handle edge case and fill with unknown padded value to complete a full 4 byte word which did not allow us to recreate the program on python.

Had to handle dynamically changing FLASH content endpoint. Start by using linker file of known sections with LMA addresses existing in FLASH and then using size of other sections in FLASH (.data) , these sections although existing in FLASH to begin, at runtime they use VMA Address because they are loaded into the RAM? And utilize that address instead and was unsure of what sections would be deleted.

So modified the linker file and created a new section right after the last section used in flash(.data) and kept new section completely in FLASH allowing the LMA and VMA to be in flash representing the end point of the FLAHS memory area, then made that variable EXTERN to allow it be used in other files to allow it to represent a dynamically changing endpoint of the of .bin file in the FLASH memory.

Oct 7th – reprofiled program and worked through full implementation and roadmap of CRC implementation in setup program. Looked into post build operations to append 4 byte value onto the end of the .bin file that would be for the clean CRC value over a successful .bin image.

Oct 11th- ATE Sprint

Developed Camera API library architecture and code for aerospace ATE– Utilized multithreaded programming and OpenCV2 library using python Threading module to introduce new **threads** based on user commands and **thread lock mechanisms and events** to ensure safe resource allocation and to run in background and keep active watch on states in a **non-blocking manner**. Allowing multiple tasks to be ran in the background while giving user full recording and picture functionality

Developed library to allow for maximum compatibility with other camera through the use of OOP and parent classes, that allow device specific files to inherit from general camera API functions (**What do we call this structure of programming)**

Utilized thread events to keep timed tasks in a less CPU intensive state, decreasing CPU usage by 74% as opposed to running an endless while loop

Nov1st – Looked at SPI speed profiling, configured different clock confugrations in accordance with hardware specifications to achieve maximum data transfer rate while ensuring accurate checks using CRC for data validity. Used datasheet to write low-level SPI commands for Display functionality.

Improved data transfer speed by X% (6.91ms to 3.91ms) using clock calculations and different clock configurations on the stm32 , for writing a 8Kb block of data to a display. Did this by calculating optimal clock configurations, running PLL as sysClk and running to max rate to ensure that computations happening at fastest rate possible then configuring SPI2 specific timer, APB2 to max rate, 45 Mhz (SPI2 prescaler =2) , 22.5 Mhz. Adjusted slew speed (maximum output speed) for pins which was causing data instability , slow to very fast. Allowing a successful valid LCD screen write of 8Kb in 3.91 ms

Dec 1st- Looked at implementing further ATE functionality , along side with NV3052 video engine chip experimentation. Looking into OTP(one time programming capability) . For ATE looking at the sprint and developing setup and teardown of the suite after execution. Graceful execution and error logging using robot framework, handled all device closures

Dec 6th – Configured 9 bit spi transmission, requires user to **use a uint16\_t buffer in which only the first 9 bits are considered**, and then from there must be in MSB or LSB order. Learned about bit-banging in which you have to write functions that force gpio pins to reenact spi communication. For example SCK will be toggled at a rate to simulate CLK signal, MOSI pin will be GPIO output and will take in data through code and will toggle the pin as an output according to the data (ex. 0xFF in MSB means holding the pin high the whole transmission), MISO pin will be the GPIO input and will store the signals coming in on each clock cycle in a buffer and then will be read as data.

Dec 7th – Looked into voltage matching options,

Buck boost converter -> used to convert lower DC-DC voltages at one constant level, will take 3.3V and boost it to constant 5 V, but it will be of lower current to abide by power rules (power= I x V)

Voltage level shifter IC -> Used to shift bidirectional voltages to different levels, for example if a device uses SPI in which 0 is low and high is 5V but the microcontroller operates within in a (0-3.3V) range then the Voltage level shifter IC , shifts the voltage to take into account the level shift

LOCK INTO CAN system and INVERSE KINEMATICS AND SIGNAL PROCESSING FOR ROBOT ARM

**CAN Proj**

* Prototypes the code through block diagrams and system design requirements

**Define Objectives**: Develop a modular system using the CAN bus interface that allows multiple sensors to be monitored and logged onto an SD card and displayed to a LCD screen. Must be able to start/stop process via button.

**Constraints**: Needs to log every message within 50ms period, must update the LCD Screen every 200ms, must use less than 12 Volts of power, needs sensor addition to be very moduar

**End users and their needs:** Engineers on the BAJA team to use the data in a visual way that allows them to make informed decisions about how their work affects the vehicles performance

Identify:

1. Primary Tasks of the system
2. Delegate/Create the task-specific requirements (timing, when to sleep/wake, what it actually does, intertask communication mechanisms, event or periodic driven)
3. Inputs, throughputs/processing, Outputs
4. Interrupts and what they do
5. Worst cases for memory, timing(response time and deadlines) and latency

Create:

1. Functional Block Diagram; Visual representation of system showing hardware and task interations
2. Timing Diagram; A visual depiction of execution, interrupts, communication and program flow
3. Task Dependency graph; for each task show what it depends on and what it activates

**Requirements:**

1. **Primary Tasks**
   1. **The system shall receive and sort every CAN message as it comes into the buffer using an Interrupt to signal message availability**
   2. **The system shall record all CAN Messages within 1 second period in a Mail Queue of CAN message struct (Find max mail queue size based off of message intervals) and post to SD card**
   3. **The System shall update the LCD screen with the appropriate information every 250 ms**
2. **Task-Specific requirements**

Work so far

**Events**

* Events are a 32 bit binary number in which you can change bits and have that act as a flag, ex. From 0 to 1 or 1 to 0. Of the first bit. In addition to being a flag, it can be used to block threads and hold them in an indefinite blocked/unready state until the specific bit is set. Meaning the scheduler knows it is always blocked until the bit is set and wont ever give it the CPU until then. In addition to that the event is thread safe meaning it can be accessed by many threads. After the function runs once it has the option to reset the flag and then go bing blocked while waiting for it to be blocked again.
* I used the event group as part of an ISR routine (This itself required lots of modifications to the FreeRTOSConfig.h file to allow the ISR version of xEventGroupSetBits to work properly, had to enable timers and static allocation and specific files, GPT for the way) the ISR was used to set bits in the event object when ever the RXB0 of the MCP2515 recevied a message. A separate thread of priority above normal used the xEventGroupWaitBits function to wait for the ISR to change the bits, then would actually run to process the message and add to the mail queue and go back to the blocked state by resetting the event group bits (param of the fxn) until the ISR set the bits again meaning msg available.

**Struct**

* The struct is structure used by C to basically create an “object” that allows you to have on “object” that has a name, and you can modify the parameters/attributes of the object by defining what they should be in the file. Very clean way to keep information modular and clean to track. Ex for a CAN message object is below
* A screen shot of a computer code

  Description automatically generated
* This gets a little messy if you start storing pointers in the struct, as if the struct is being used in other threads, than the one it is created in, the pointer needs to point in the heap memory, as pointer to a local variable in one thread, will NOT transfer to another thread.
* So for this case, because the struct is being used in between different threads, all parameters will be passed by copy, not pointer/reference
* This allows raw data to be moved around between threads, making it thread safe BUT Greatly increases memory usage now as you have to pass around copies not references
* Storing numbers is easy, but when working across threads or outside the scope of the functions call stack, have to see

**Mail Queue**

* **Depending on type, typically by-copy queue is used, sometimes a pointer queue is used**
* If pointer queue is used, A queue where only pointers are stored (memory address to the struct) the pointers needs to point to memory in heap, as local pointers will not exist for the same memory value in a different thread, which is the whole point of a queue (inter-thread communication). Much less memory than storing entire structs by copy.
* Remember that whenever you allocate memory via the malloc function, you must free the same pointer. Couple of options when creating a struct.
* For this case, I am using a typical by-copy queue, why? Because I don’t really know how to measure and allocate heap memory for every instance of the CAN message? IG I could use a pointer queue
* By-copy queue is much nice and easier to work with , by-copy means you simply add your object/struct to the queue and then a copy of it is stored there (don’t know the memory theory behind this one) and then can be accessed and read by any thread , but there is a max queue size
* Ok so using mail queue or just a regular from FREE rtos and storing pointers as done below; we send the address of the pointer (which is the data we want to send) as the pointer holds the address of the malloc structure
* A computer screen shot of a program

  Description automatically generated

**Debugging**

* Had this problem where system would always throw a hard fault and redirect to hard fault handler, don’t know exact cause but was able to **troubleshoot by** spamming print statements to see where it stopped, and then commenting out random parts of code until it worked, then go smaller and smaller commenting out until it works then diagnose the problematic piece of code.
* **Hard Fault:** SprintF has to be done using malloc , NOT using a local variable other will cause hard fault