**Before Week 1**

What I need ready

Get the old sensor system working to demo on week 0

* Clean working environment

Working Github environment that plays well with STMCubeIDE

* We had some issues last year with syncing and potentially bricking STMCubeIDE installs

Design Decisions

* Keep old main.c file or rewrite?

**Week 2** – Introduction to Firmware, STMCubeIDE, and Register-Level Programming

Overview of Firmware

* Demo old sensor system
  + Send commands and receive data over USB connection
  + Use oscilloscope to probe the communication pins between board and sensors

STMCubeIDE

* Downloading the IDE for windows
  + <https://www.st.com/en/development-tools/stm32cubeide.html#st-get-software>
  + You must make an account
  + Extract downloaded zip & run exe
  + Only download STLINK Drivers
    - STLINK USB driver is required to upload code to microcontroller board, make sure you have this
* Making a Project
  + Start NewSTM32 Project
    - Takes a bit, has to download some files
  + Target Selection
    - Commercial Part Number: STM32F439ZIT6
  + Project Setup

A screenshot of a computer program

AI-generated content may be incorrect.

* + - Click Next, not Finish
    - Make sure its “Copy only the necessary files”
    - Finish
* Downloading Necessary Software Packages
  + Help -> & Login

A screenshot of a computer

AI-generated content may be incorrect.

* + Click this button to generate code
    - This process downloads microcontroller drivers, sets up the project environment, creates code templates
    - This process is decently intensive on the computer

A computer screen shot of a computer

AI-generated content may be incorrect.

* IOC
  + Allows for easy configuration of board functions
    - pin functions : GPIO, UART, SPI, etc. & their properties
    - clock configuration
  + When a change is made and saved, it will re-generate code
* Dark Mode is in Window -> Preference -> Appearance
  + Makes comments hard-ish to read
* Code Generation
  + When the IDE generates code, it creates and updates all c files based on the settings in the IOC
  + Look at main.c
    - The file is split into multiple sections by comments, each section has a specific purpose you should follow. In some cases, code generation will mess with certain sections, so make sure you are writing in the correct section to not get your code overwritten
    - main function : will be ran on startup
    - while(1) loop : infinite loop
      * some embedded/firmware systems are written to include some kind of continuous loop
* Project 1: Blinky
  + Goal : make an LED blink on/off
  + Step 1: Enable the pin which controls the LED
    - Which pin? Check user manual
    - User Manual: <https://www.st.com/resource/en/user_manual/dm00244518.pdf>
      * This type of document specifies features for the specific board : NUCLEO-F439ZI
    - Green LED is connected to pin B0
      * A pin refers to an individual output
      * A port refers to a collection of pins
      * B0 is the 0 pin in port B
    - In IOC, set PB0 to GPIO\_Output
      * General Purpose Input/Output

A grey square with white text and numbers

AI-generated content may be incorrect.

* + - Save and Generate Code
  + Step 2: Code the functionality
    - Back in main.c

A screenshot of a computer program

AI-generated content may be incorrect.

* + - MX\_GPIO\_Init() is new code added by code generation. This function enables the pin in GPIO mode, details will be expanded on later.

A screen shot of a computer

AI-generated content may be incorrect.

* Step 3: Uploading to the Board
  + Detailed steps filled out later when I get access to a board
* How does this code work
  + We’ll look at this at a register-level
  + Explaining HAL\_GPIO\_TogglePin
    - HAL Documentation: <https://sourcevu.sysprogs.com/stm32/HAL/>
    - HAL\_GPIO\_TogglePin page: <https://sourcevu.sysprogs.com/stm32/HAL/symbols/HAL_GPIO_TogglePin>
    - Function Source Code:

A screenshot of a computer code

AI-generated content may be incorrect.

* + - First Argument: GPIO\_TypeDef\* GPIOx
      * \* means this is a pointer to a data type
      * GPIO\_TypeDef definition : <https://sourcevu.sysprogs.com/stm32/CMSIS/symbols/GPIO_TypeDef>
        + GPIO\_TypeDef is a struct that refer to the registers of a specific port.
      * This essentially points the function to the correct port
    - Second Argument: uint16\_t GPIO\_Pin
      * This is an unsigned 16 bit integer
      * This tells the function which pin of that port to toggle
    - -> means to modify a variable in a struct. In this case, we are updating the BSRR register since the struct members refer to registers
* GPIO Register
  + Reference Manual: <https://github.com/CalPoly-UROV/Embedded24-25/blob/main/Docs/referenceman-stm32407mcu.pdf>
    - For family of microcontrollers
    - Go to GPIO section, read it
  + Each GPIO port has a collection of registers that control how that port functions. It includes settings and the current state of the pins, whether they’re high/low, etc.
    - We care about the registers that allow you to toggle the pin, these are GPIOx\_ODR and GPIOx\_BSRR
    - **GPIOx\_ODR**: in output mode, this register directly sets the pin, 1 is high, 0 is low

A screenshot of a computer

AI-generated content may be incorrect.

* + - Writing 1 to ODR0 will set pin 0 high, writing 0 to ODR0 will set pin 0 low
    - **GPIOx\_BSRR** : in output mode, to set a pin high or low, you just write a 1 to the correct location

A screenshot of a computer

AI-generated content may be incorrect.

* + - For example, writing 1 to BS0 will set pin 0 high, and writing 1 to BR0 will set pin 0 low
  + **Issue**: In the code, you can’t directly access each individual bit in the register with something like BSRR -> BS0 = 1, you have to assign a value to the entirety of BSRR. This is what the HAL\_GPIO\_TogglePin code is doing: bit manipulation.

Next project: Control an LED using a button

* Use User Manual to find out what pin the button is connected to
* Set up the pin as a GPIO input
* Toggle the LED based on pin input

Documentation Form

**Week 3** –Practice with register-level programming and getting more familiar with C

Goal: Use the microcontroller’s timer module to control the brightness of the board LED

* By turning the LED on and off quick enough, we can control its apparent brightness
  + This is called Pulse-Width Modulation “PWM”
* At short delays/high frequencies, HAL\_Delay is too inaccurate, so we need to write our own delay function
  + Hal\_Delay has a minimum 1ms delay which gets inaccurate at the single-millisecond level
    - Demonstrate this using the oscilloscope
    - To visually dim the LED, we don’t actually need that level of precision, but future projects will require a precise timer
* We can use a feature of the microcontroller, a timer
* Timer module
  + Reference Manual: <https://github.com/CalPoly-UROV/Embedded24-25/blob/main/Docs/referenceman-stm32407mcu.pdf>
  + The timer module works by incrementing a counter based on an input frequency
    - The input frequency can come from the on board clock, or it can be an external input
    - The counter can function in many ways
      * Count up/down
      * When the counter reaches a certain value, do something
        + Reset the counter, stop the counter, fire an interrupt, etc.
  + Configuration
    - Prescaler
      * Divides the incoming clock signal further so it takes multiple inputs to increment the counter by 1
    - One-Pulse Mode
  + Auto-Reload Register
    - Set value at which the timer will reset, or trigger and interrupt, or etc.
* Delay Function #1
  + Continuously poll the timer counter register in a while loop, when the timer count is past a certain value, reset timer
  + This more accurately simulates HAL\_Delay
  + Issues
    - Pauses all processes, we can’t run any other code while waiting
    - Latency caused by polling
      * Miniscule, few cpu cycles at most
* Write a LED brightness controller using delay function #1
  + Control the brightness of LED using PWM
    - PWM generated by loop with delay
* Delay Function #2
  + Set timer module registers to automatically reset. On reset, trigger an interrupt
    - Explain how to use interrupts
      * Set up and enable timer interrupts
        + NVIC\_EnableIRQ
        + \_\_enable\_irq()
        + TIM2.dier
        + Interrupt Flags in TIM2.SR
      * Handler Function
        + Reset interrupt flags
  + Pros
    - We can run other code while the timer is running
    - Isn’t polling, so its more accurate
  + Cons
    - Doesn’t emulate HAL\_Delay functionality
    - Complexity
* Update LED brightness controller to handle controlling 2 LEDs
  + 1 LED controlled by polling
  + 1 LED controlled by interrupt

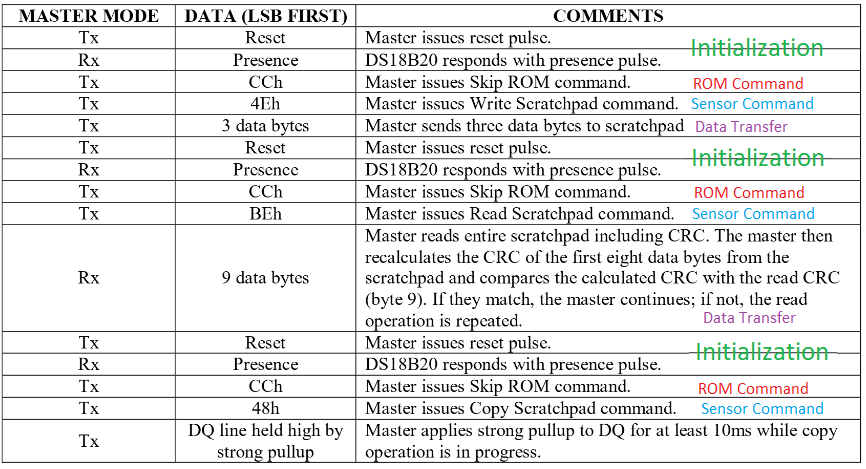
Relevant Topics

* C Functions
  + We’ll go into more C language topics
    - Functions
    - Data Types
* Debugging
  + IDE Debugger
    - Breakpoints
    - Shows current states of internal registers
  + Oscilloscope Probing
  + Print to Terminal
    - I will provide the code to print response to a terminal
    - UART, teraterm, stuff will be explained later

**Week 4**

Lets try to write a driver for the temperature probe

* Microcontroller Datasheet & Registers
  + What to look for in a datasheet
    - Communication Protocols
    - What commands to send
    - How the data is formatted
  + Datasheet: <https://github.com/CalPoly-UROV/Embedded24-25/blob/main/Docs/stm32f40xx-datasheet.pdf>
    - STM32F439ZI
    - For specific microcontroller
* Temperature Probe Datasheet: <https://github.com/CalPoly-UROV/Embedded24-25/blob/main/Docs/DS18B20.pdf>
  + Page 1:
    - Uses 3 wires: high, low, and communication
      * Unique because most sensors usually use at least 2 wires for communication
    - Allows for multiple sensors to be connected to a single microcontroller
      * This is common for sensors to be able to be chained like this
      * Each sensor has a unique rom code/ID which allows for the microcontroller to differentiate and select a specific sensor
  + Page 10: Transaction Sequence
    - This tells us what steps to take to talk will the sensor. Most of our firmware coding revolves around finding out how to talk with a sensor
    - Initialization, ROM Command, ROM Command Data Transfer, Sensor Command, Sensor Command Data Transfer
  + Page 18: Example Transaction Sequence



* + - So we know what to do, how do we do each step?
  + Initialization/Reset Pulse

A diagram of a line type pulse

AI-generated content may be incorrect.

* + - Goal: reproduce the reset signal and read when the probe’s presence signal
    - Open drain wire protocol
      * Explain how the open drain and push/pull functionality works for communication
      * How to set up open drain config on gpio
        + Set a gpio reg configuration or use IOC

Gpio reg config is preferred because we can change the pin easier

* + - Use delay-based timer function
  + Write/Read Slots
    - How the protocol handles data transfer for commands and data transfers

A diagram of a bus

AI-generated content may be incorrect.

* + - Goal: write C functions that can handle reading & writing bits to the wire
    - Here we’ll talk more about C code
      * Macros
      * Header Files
  + Put it all together
    - Put the code together to be able to read temperature

**Week 5**

Communication Protocols

* I2C, UART, SPI
  + <https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/>
* UART Terminal

A close-up of a white background

AI-generated content may be incorrect.

* + Use a buffer to store the output text
  + Teraterm: <https://sourceforge.net/projects/tera-term/>
  + Small Project: Update LED code to spam the terminal
* PHT Sensor
  + Goal: Get it running for pressure and temperature. Ignore humidity
  + Datasheet: <https://github.com/CalPoly-UROV/Embedded24-25/blob/main/Docs/ENG_DS_MS8607-02BA01__C3.pdf>
    - Page 5:
      * Uses I2C
      * 5 commands for working with pressure and temperature
        + Each command is 1 byte long
    - Page 6 :
      * Contains a Table of all the commands and their variations
    - Page 8:
      * Contains flowchart for how to read temperature and pressure
      * 1: Start
      * 2: Read PROM
      * 3: Read Temp & Pressure Data
      * 4/5: Calculations using read data
    - Page 11:
      * Info about how to set up I2C
    - Page 12-14:
      * Shows what commands to send over I2C and their responses
  + Setting up I2C
    - Using IOC

A screenshot of a computer

AI-generated content may be incorrect.

* + - Enabling I2C in the IOC will set up and allocate pins
    - Save to Generate Code
      * In main.c a new function will be created : MX\_I2C1\_Init()
      * Also generates code in stm32f4xx\_hal\_msp.c
        + HAL\_I2CMspInit

This sets up the pins to I2C mode

* + Using I2C
    - HAL\_I2C\_Master\_Transmit()
    - HAL\_I2C\_Master\_Receive()
      * Blocking, so it waits for a full response before continuing
      * HAL\_I2C\_Master\_Receive\_DMA() is non-blocking

**Week 1 – 5**

Begin reviewing firmware system (communication with topside, commands)

Begin reviewing thruster PWM driver

**Week 6**

Catch onboarded members up to speed on current systems

Onboarded members begin new imu driver

**Week 7**

**Week 8**

Finish Firmware backbone, thruster driver, old imu driver

**Week 9**

Begin investigating competition task related sensors

Begin servo driver

**Week 10**

**Winter**

**Week 5**

Finish servo driver, imu driver, competition task-related sensors

**Week 6**

Stretch goal investigation