Summary:

The purpose of the Hardlight firmware is to:

1. Perform basic suit startup activities
2. Take “Instruction Packets” from the computer, and translate them into suit actions.
3. Store suit “state” such as what motors are on, what color the front LED is, etc.
4. Perform routine tasks, such as returning tracking data and routinely pinging the computer to verify the suit is operational.

On a technical level, the Hardlight suit is essentially a huge collection of peripheral devices on two I2C buses, coordinated through a central MCU. The two core functionalities are:

- Motor commands, which are written in series to 16 individual DRV2605 motor drivers, mediated in groups of eight using TCA9548a I2C splitters.
- Tracking data, which is collected on a polling basis from 3 different BNO055 9 axis MPU trackers in DMP sensor-fusion mode, all mediated by a single TCA9548a.

(Note that the TCA chips are required because both the DRV and MPU chips only support 1 and 2 possible I2C addresses, respectively.)

In previous iterations of the suit, both the motor drivers and tracking system were on the same I2C bus. This presented significant timing problems - the motor system is inherently dynamic, with traffic depending on the demands of the user’s game, but the sensor system is inherently synchronous, with a large amount of extremely time-critical information delivered at a set transmission rate. When the two systems shared a bus, these two systems would conflict - too much haptic instruction would cause the tracking to lag, and too many trackers active in the system at once caused significant haptic delay.

The current goals of the firmware are:

- Support for motor commands via the main I2C network
- Support for BNO055 tracking data receipt over the secondary I2C network
    - Stretch goal: do this entirely with non blocking transactions
- Command of other various elements of the system, such as the TCA network splitters, direct PWM control over the LED chip, and management of power, if necessary.
- Reliable connection over UART serial to the computer.

Firmware Structure:

Classes outside the basic hardware control classes are documented in Doxygen, and have accompanying documentation.

- Main:
    - Defines the Serial class, which is passed by reference to many other classes due to the need for a shared buffer
    - Contains the “Task” and “Instruction” classes
    - Performs one-time startup activities
    - Main loop interprets incoming UART instruction with a case statement based on the contents of the incoming UART packet. [See attached “Packets” documentation.]
    - Calls periodic UART and maintenance activity, such as pings and the return of tracking data.
    - Currently also stores some state information, which should all be transitioned to the “Task” class
- Instructions
    - A wrapper class that manages user-sent instructions to the suit.
    - Responsible for translating the raw data in a packet into “Task” class functions, and correctly assigning packet data to the Task function’s parameters.
    - Does not store any suit state information
- Task:
    - Performs every task that is periodically required of the suit
    - Declares all hardware management classes, and the Packet class
    - Most of the “Instruction” functions are one to one with “Task”, but Task contains additional functions that cannot be called by the computer, such as periodic pings and tracking activity.
    - Also stores suit state information, such as tracking, motor and LED status.
- Packet:
    - Manages how serial packets are sent back to the computer.
    - Stores “default” packets that can be edited with new parameters
    - Also contains basic functions for “clearing” or “resetting” packets.
    - Basic examples of use include the “Ping” task.
- BNO055, DRV2605, TCA9548a
    - Device libraries that directly manage the hardware via I2C
    - Called by Task.
    - Store device information, such as registers, but not state.

Packets:

[see attached]

Arduino Code:

For reference, we also have old arduino code available from the MKII version of the Hardlight suit. This code uses the Arduino language, and is much simpler, but contains most of the same instructions. It can be useful for outlining the minimum viable product for the MKIII firmware, as well as the base acceptable Hardlight instruction set.

Major differences:

- **Limited use of classes:** almost all functions are simply declared in included .ino files within the scope of Main. Instructions are in the instructions.ino file, periodic activity such as startup functions and tracking is in the routines.ino file.
- **No buffers in UART Serial:** the restriction of the Arduino serial buffer to 64 bytes caused major issues when sending large volumes of data, which is avoided in the MKIII.
- **Only one I2C bus:** Since the Ardunio Micro only has a single I2C bus, all I2C transactions are blocking, meaning that large volumes of haptic activity can disrupt how quickly tracking data is sent back, and vice versa.
- **No multi-color LED control:** the MKII only has a single white LED on the chest, which can be blinked on and off.