HYDROWATCH

Tyler Lovelace, Rebecca Hoehne, Mark Etzelmueller, Adam Burt, Tansy Wang

2. Table of Contents

3. Introduction

The world is full of wearable devices. Recent devices of interest, such as the FitBit and Apple Watch, provide the user with information regarding their heart rate and movement throughout the day. With these devices, users receive quantitative and qualitative information regarding their health. A key component of health is hydration. The prevention of dehydration has always been a vital part of survival, but a practical wearable device that is able to inform an individual of their relative hydration level does not currently exist. We propose the construction of a wearable device that, based on a baseline measurement, communicates to the user a relative hydration level and advises the user when to drink water to rectify low hydration levels.

The main market for our proposed device is the average health conscious consumer. People who wear a FitBit, Apple Watch, or similar device for the fitness monitoring features are the main target for the device, and with millions of users, this device would become popular among these people. Additional markets for this device would be high performance athletes and military. Professional athletes need to maintain peak performance in order to properly do their jobs. Our device would ensure they are constantly aware of their hydration level and never allowing it to drop, especially off the field when their bodies are recovering. The military also could use our device to ensure troops are staying properly hydrated during training and combat situations. Keeping people in the fight is crucial to military success, and hydration can be an enemy if not properly monitored. Heat exhaustion and heat stroke are real factors that can hinder a unit's ability to fight in desert, jungle, and mountainous environments. Our device could be used by the Army, Navy, Marine Corps, and Air Force to ensure members are always staying

conscious of their hydration level to ensure they do not become casualties due to lack of hydration. Overall, the goal of our device, HydroWatch, is to give users the ability to accurately track their hydration and receive reminders and alerts when their level of hydration is dropping too low.

We will determine relative hydration based on concentrations of water present in tissue on a person's forearm. We will do this by interrogating the tissue with specific wavelengths of light, emanating from surface mount LEDs, chosen based on the absorption spectrum of water. The light will be scattered and absorbed by the tissue, and some of the light will be detected by a photodetector around 1 cm away from the LEDs. Water has a very distinct absorption spectrum that increases drastically in the near infrared (NIR) region. In this NIR region, water is the most absorbing component of human tissue. Therefore, light in this region at 970nm and 1200nm will be used to interrogate the tissue. We expect that if there is a larger concentration of water present in the tissue, then the reading from the photodetector will be lowered due to fewer photons incident on the detector. Since water is the most absorbing component in tissue at the NIR wavelengths, the amount of light reaching the detector from these wavelengths will change only when the concentration of water changes.

Another significant wavelength of light that is used in our device is 450nm. This wavelength is used to determine skin contact and whether we should trust data received during a test run. Blue light is highly absorbed by the melanin in human skin, so if the the device and blue LED are in close contact with the skin, the photodetector does not pick up any signal. However, if the device is not in close contact with the skin, some of the blue light will reflect off of the skin and be detected at the photodiode. This information tells the system that the gathered data was

not accurate as the device came off of the user. This mechanism acts to throw out any inaccurate data that would potentially give an incorrect reading of relative hydration level.

When the board is powered on, the various wavelength LEDs begin flashing , 1.5 ms on 0.5ms off, to send light into the skin to be detected at the photodetector. We can then connect the Bluetooth interface via a dongle on a computer system and export the data from the photodetector wirelessly. We can then process this data and allow the user to observe it using a MATLAB GUI which determines if the board was in good contact with the tissue and then informs the user of their hydration level relative to the control measurements that were conducted in the testing portion of our design cycle.

Our final HydroWatch design met all of our original expectations, as were previously listed in our High Level Design documentation. Our main requirements for this project were to be able to interface with the two TI AFE4490 chips to set the flashing rate of the LEDs, receive photodetector readings via an SPI interface, have a stable Bluetooth interface to send data wirelessly from our board to an external processing machine, and have the ability to post process the data and notify the user of his or her hydration level. The HydroWatch program allows an impressive amount of control of the LED current values and photodetector gain amounts. Our program also provides a very stable Bluetooth interface to send data to the RSL10 Bluetooth Low Energy dongle, as well as allows us to establish a strong connection when the HydroWatch program is running.

Due to the complex biochemical nature of hydration levels, our goals were difficult to meet because hydration is not something that can be easily quantified for every person in the same manner. Initial sponge tests to roughly simulate hydrated human skin provided good results

that allowed us to decide upon the wavelengths of light that we could use to be the most numerically significant in terms of hydration levels. HydroWatch is capable of demonstrating relative hydration levels in relation to the control group measurements that we conducted to determine a baseline hydration level. While we did have aspirations of being able to give information to the user about him or her becoming dehydrated, we found that hydration levels do not drastically change in short periods of time, and as such, it is still useful to have a relative hydration level to be compared to with multiple measurements throughout the course of the day. The GUI for HydroWatch also allows the user to easily use the project and a medium to visually capture the complex phenomenon of human hydration levels. Overall, we are quite pleased with HydroWatch's ability to control the different wavelength LEDs and provide measurements and meaningful data surrounding the nature of hydration levels in the human body.

4 Detailed System Requirements

The basis of the system relies on the detection of absorbed light by water, deoxyhemoglobin, and oxyhemoglobin. We must design a system that is able to process multiple LEDS and capture real time data of the absorbency. This system ideally will be able to control at least four different wavelength LEDs at the same time and be able to accurately distinguish between the various wavelengths of light incident on the photodetectors while taking into account the ambient light that also may hit the photodetector at any given moment.

As this project is intended to be wearable, the end goal for HydroWatch would be to be powered by a LiPo battery to allow for portability of the system. The user should also be able to

run the system off of a USB Mini-B cable, such that he or she can select to power the board off of either a USB or a LiPo battery. This requires a stable DC-DC converter be present on the board to take these different operating voltages and output the required 3.3V for the system. Seeing that the RSL10 draws around 30 mA when running its processor, the AFE4490 consumes 70 mA to run the flashing logic and read photodetector values, and the LED currents will be in the range of 0-100 mA, a rough estimate for current usage in our project would be 300 mA in a worst case scenario for usage. The user should be able to use the board for at least one full day taking measurements. When not in use, the board must be able to go into a low power, sleep-like mode. The AFE4490 chips can be powered down with the use of a low signal on a digital output pin, while the RSL10 itself can be placed into sleep mode. In these modes, the collective current draw for the entire system would be 15 μA and would likely be in this mode for nearly 23 hours of the day, while the 300 mA current draw would be the active measurements, totaling a maximum of 1 hour. Because of these calculations, it can be seen that the battery capacity must be at least 600 mA-h.

The AFE4490 requires that a stable serial peripheral interface (SPI) be created and executed such that 8-byte instructions can be safely sent and correctly received to and from the RSL10 microcontroller. The AFE4490 requires that the timing registers for when data samples are to be taken, LED flash rates, photodetector reading synchronization and the number of samples to take all must be correctly initialized for this project to work as expected. The AFE4490 must also be communicated to through SPI in order to set crucial design parameters, such as the intensity of the four wavelengths of light, the transimpedance amplifier gain to accurately represent the full dynamic operating range of the AFE analog to digital sample

conversions, the output reference voltage operating range, and the ability to both read and write commands to these chips is also dependent upon this stable serial peripheral interface. The AFE can handle SPI communications at up to 4 MHz, as the external crystal oscillator provides an 8 MHz clocking signal which is then divided down to adequately handle and clock all of the instructions into and out of the AFE. The HydroWatch system also is required to be capable of placing the system into a low power and/or low current consumption mode, such that battery capacity and life can be preserved.

HydroWatch needs to have Bluetooth capabilities that will allow for at least 8 bytes of information to be sent at a time while also providing the user with a functioning distance of a reasonable range from the receiver. In an effort to combat possible noise in a crowded RF space because of all the available Bluetooth devices to pair with, HydroWatch must be able to be programmed such that the power to the antenna to transmit the Bluetooth information can be altered to find the maximized performance metrics. We also must have a receiver of some sort on the processing side such that it can convert the Bluetooth signals into utilizable data. In the RSL10, there is a built-in BLE interface and antenna. In the choice to use the RSL10, it became clear that these requirements were realizable to an extent, but with some minor modifications and clarifications. The output power to the antenna must be kept greater than 0 dBm (to allow for forward propagation of signal), the RSL10 BLE dongle must be used to receive the BLE signal and convert it into usable data on the Bluetooth Low Energy Explorer program, and within BLE Explorer, we were required to increase the transmit/receive power of the dongle to its maximum of +6 dBm in an effort to increase the functional range of HydroWatch.

For this project to be successful in this early prototype design, HydroWatch needs to have a form of external data processing to view and analyze the data received through the Bluetooth interface. This post processing tool must be able to parse through log files from the Bluetooth Low Energy Explorer software to extract the pertinent 8 byte photodetector readings. This post processing tool must also provide the user with the ability to graphically view the results of the photodetector readings in comparison to a baseline hydration level, which would ideally be set by an initial daily measurement by the user at the beginning of the day, as well as view the readings from the three other LED wavelengths to view the biologically significant information provided by each. The post processing tool should be able to use the reading and corresponding value of the 450 nm (blue) light to determine if HydroWatch is in sufficient contact with the user's skin. If the user does not have the device in complete contact with his or her skin, the post processing tool must be able to determine this discrepancy and disregard the data recorded during that measurement while also notifying the user that they had poor contact between the device and their tissue. Similarly, this post processing tool must be able to take the photodetector reading from the 1200 nm LED, the most significant wavelength for determining water concentrations in the tissue through absorption, and notify the user that he or she is hydrated or dehydrated relative to his or her initial hydration level. The notification methods that should be realized by this post processing tool include sending text messages to the user with their hydration status, sending the user an email with the same information, and visually displaying the user's hydration status on the post processing user interface in a clear manner.

For this project to successfully determine the hydration status of the user, there must be a certain amount of testing put into action to ensure that the readings received from the

photodetectors are biologically significant. This testing must consist of testing multiple test subjects who are initially believed to be dehydrated. Then, after getting at least three baseline measurements of that test subject, the subject was instructed to drink what we deemed to be an almost excessive amount of water to completely differentiate the levels of hydration and dehydration of the user. After waiting upwards of 45 minutes to 60 minutes, the subject were tested again with the project and his or her relative hydration level had been affected in some noticeable manner. To accomodate for not all skin types being the same in terms of pigmentation or moistness, these tests should be conducted on many subjects of varying backgrounds beyond that of the HydroWatch group. To keep the testing as ubiquitous as possible, one should try to run these tests on the same area of each subject, namely the right forearm with the infrared photodetector closer to the elbow for the most certain contact and pressure applied to get viable readings. In running these tests, it must be ensured that the user has the system in contact with his or her skin completely, and if the 450 nm light does not register as a nearly zero measurement, the subject must retake the data in order for our testing to produce the most reliable results possible. To maintain consistent pressure, it is advised that one look at the reading from the 450nm LED. If these reading are roughly the same, approximately the same amount of light is being absorbed and thus somewhat consistent pressure is achieved. After conducting these tests, we need to be able to take these data points and be convinced that the photodetector measurement of the 1200 nm light actually corresponds to the test subject drinking water and rehydrating himself or herself.

For this project to be wearable, we must ensure that no dangerous voltages or currents are present, as these can present serious health concerns for humans. There should be no exposed

wires on the final board design in an effort to make sure that these types of contact are not possible and the user is as safe as possible. This project needs to be able to lie in contact with the skin, and as such, design considerations must be made so that the LEDs and photodetectors are able to lie on the back of the board to be in sufficient contact with the tissue to be interrogated. Similarly, the LEDs and photodetectors should be far enough way that no crosstalk between different sources is possible, as well as the wavelengths of lights used should not be dangerous or damaging to human skin. The device itself must be very lightweight and also be able to fit in some configuration on a human wrist or arm. The device needs to be secured onto the user through the use of some sort of athletic band or other movement-restricting manner.

5 Detailed project description

5.1 System theory of operation (how the whole thing works)

HydroWatch consists of one main board, which contains the RSL10 microcontroller, two AFE 4490 chips, four LEDs, two photodetectors, and selectable power inputs between Mini-USB and a LiPo battery. Once the main application code is downloaded correctly to the board using the J-Link by Segger, the RSL10 initializes the two AFE chips through SPI commands to have the correct timing registers and current settings for the flash rate and light intensity of the LEDs on the backside of the board. Then, the LEDs will flash at the predetermined rates and the user is expected to put the device on his or her forearm, such that the LEDs and photodetectors are flush against the skin. The photodetectors collect the various

wavelengths of the LEDs and convert the light into current, which is subsequently sent to the AFE chips to be discretized into voltage readings. These voltage readings are then sent via SPI communications back to the RSL10 microcontroller, which prepares the photodetector readings to be sent to the On Semiconductor Bluetooth Low Energy dongle via BLE.

Once the RSL10 is put into Bluetooth Low Energy advertising mode, it will continuously update the values of the four LED readings every 100 ms. These values can be captured with the Bluetooth Low Energy Explorer software, provided by On Semiconductor. When a satisfactory number of data points have been collected from the main board to the dongle, the user can save the data in a log file format. This log file is then parsed by the python files. Each line in the file is added to a list and then each string is search for the UUIDs of the LEDs (bec1, bec2, bec3, bec4). The 3 byte values from the LEDs are then added to another list. This second list is then used to write a comma separated value (csv) file based on LED identification number, the first byte of the reading, and the scaled 2 bytes value, the last 2 bytes of the 3 byte value. This csv file is then called by a MATLAB program and post processed to analyze the results. After enough testing of the HydroWatch board on human test subjects, we theorize that we will be able to decipher a baseline for typical hydration levels in average adults. Using this threshold value, our program will be able to look at the incoming data from the LEDs and compare against this value, and can light up an LED on the main board that will let the user know whether he or she is hydrated enough.

For the construction of the device, we set forth the following design rules:

AFE 4490 chips determine LED flash rates: Through SPI commands, the AFE 4490 chips will be able to set how fast the LEDs will flash. Four different wavelengths of LEDs were used (450 nm, 660nm, 970nm and 1200nm) to get different absorptions of light in the skin. The 450nm LED is used to sense skin contact. The 660nm and the 970nm LEDs are used to sense deoxygenated hemoglobin and oxygenated hemoglobin respectively. The 1200 nm LED will be used to detect water. Intensity should be enough to get past the epidermis and run through this deeper layer of tissue and be absorbed again at another point approximately 1 cm from the point of incidence.

Photodiode captures light: The photodiodes will absorb the light from the LEDs that was sent through the skin and convert this light into a current that is readable by the AFE chips. We require two photodiodes, one to capture the light for the the 440nm(blue) and 660nm (red) LEDs, one to capture infrared light from the 970nm and the 1200nm LEDs.

AFE 4490 chips receives signal from photodiode: The AFE will receive these current values and convert these currents into voltage values. The AFE will then provide the microcontroller with the voltage readings through SPI communications.

Bluetooth mechanism: After receiving the voltage measurements, the RSL10 Bluetooth SoC repackages the data so it follows UART (universal asynchronous receiver/transmitter) protocol. The required data is send from the RSL10 microcontroller and to an external computer interface, in this case the Bluetooth Low Energy Explorer developed by ON Semiconductor.

After the desired amount of data is collected a log file is created. This log file is then parsed through using a python code which extracts the relevant voltage values and the corresponding LED identification codes. This information is saved in a csv file which will be sent to the MATLAB software for processing and user updates.

Power Source: The device can either be powered by a Mini-USB or a LiPo battery. Power from the Mini-USB or LiPo battery is first passed through a DC-DC converter to ensure the 3.3V required to operate the device.

MATLAB Software: A MATLAB GUI will allow the user to input preferred method of notification (i.e. text or email). The "track" button on the GUI will call log file to be parsed and generate the csv file with all the relevant data. This data is then displayed on the MATLAB GUI on three different graphs, one for each different characteristic measured (i.e. one for skin contact, one for pulse/hemoglobin, and one for water concentration. Following MATLAB processing, the user will be emailed or texted a message with their hydration state and their hydration status will be displayed on the GUI.

5.2 System Block Diagram

In the below figure, the system operation at a high level can be seen.

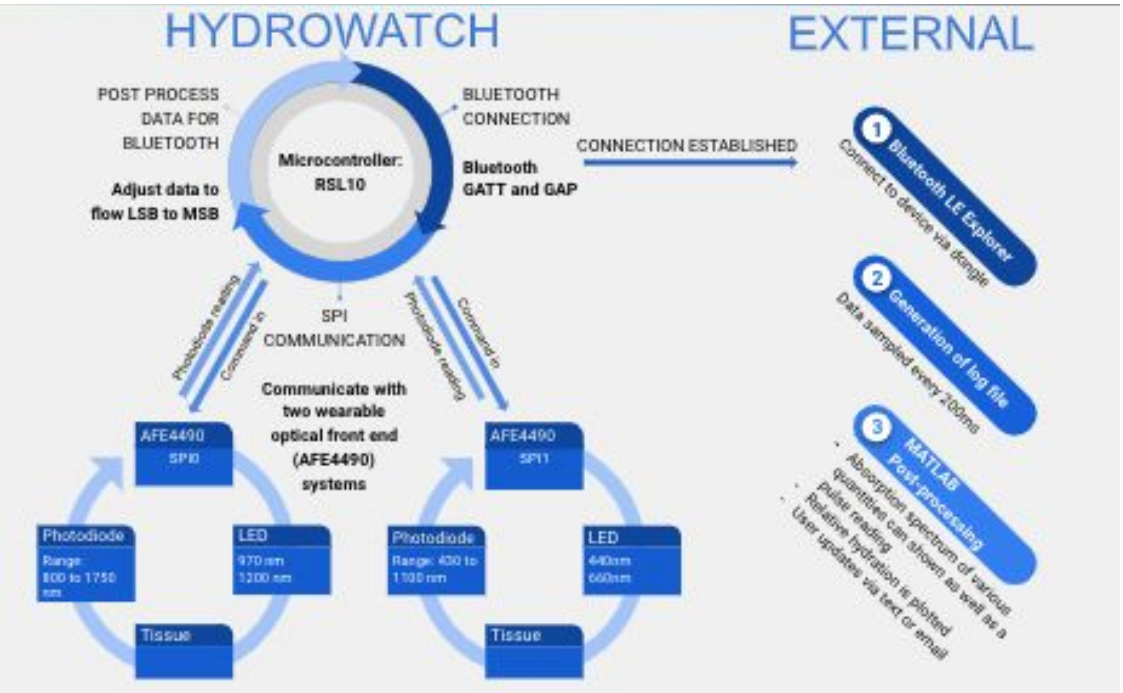


Figure 1. System Block Diagram

Subsystems List: LED/Photodetector System, RSL10/AFE SPI interface, RSL10 Bluetooth Protocol, MATLAB Post-processing and notifications

5.3 RSL10 Communication with AFE4490

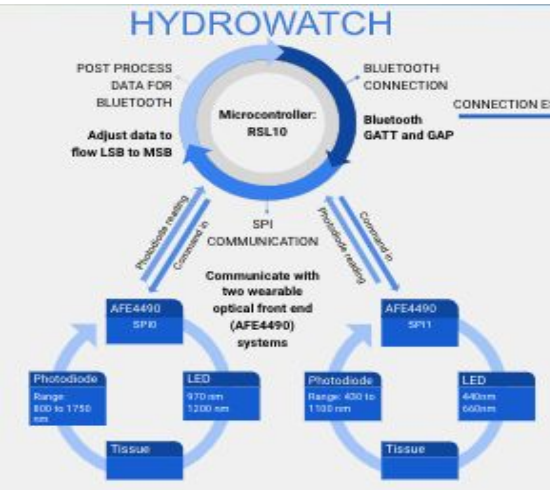


Figure 2: RSL10 Communication Diagram

Subsystem requirements: The RSL10 must be capable of having two separate SPI interfaces to communicate with two AFE4490 chips, setting appropriate timing registers, LED intensity and allowing photodetector readings for specific LED wavelengths to be received for future use in the Bluetooth data transfer.

The RSL10 was selected as our microcontroller on the basis that it had a built-in Bluetooth capability, alleviating the need for an external chip to be controlled by either I2C or SPI, saving digital input/output pins on our microcontroller, as well. Another positive feature of the RSL10 microcontroller is that it contains two SPI interfaces, and both interfaces are completely remappable to any digital I/O pins, which allowed us freedom in terms of layout of the final board as well as the ability to reduce our SPI traces down to only five traces total, as compared to what would typically require eight wires on another microcontroller with that has dedicated SPI pins, such as the PIC32, which was the other alternative microcontroller choice for this project. Other functions that led to our choice of the RSL10 were very low current sleep

modes, interrupt-driven wake-ups, powerful ARM-Cortex M3 processor for on-chip logic, C programming language compatibility and deceptively simple online documentation.

This online documentation provided much of the information we needed, such as suggested components for power delivery and decoupling capacitors as well as different operating modes for the RSL10. However, the SPI interface and commands were not very well documented at all for the RSL10. Another difficulty with developing programs for the RSL10 is that the coding environment used is Eclipse RSL10 SDK. The program hierarchy and set-up were tedious and challenging to get figured out, but included with Eclipse were sample programs for the RSL10 evaluation and development boards, of which we had two at our disposal. We attempted to use an *spi_master.c* program to figure out how a basic SPI interface would be set up on the RSL10, but when we tried to alter the information that was sent, we were only capable of sending one 8-byte word before the program would crash. This led to contacting the On Semiconductor customer service department, who swiftly redirected us to the CMSIS-Keil Driver overview pages. Essentially, we ended up using low-level system driver calls within the ARM-Cortex M3 processor in the RSL10 to create our SPI interfaces.

There was even more example code for this CMSIS driver technology, with the main program used to figure out the SPI interface being *spi_cmsis_driver.c.* This program makes low-level system calls to create the SPI interface. The program begins like most RSL10 programs do, which is with an *Initialize* function initializing and waiting for the 48 MHz crystal to start up for the system to clock and register commands correctly. Then, the system clock is divided and prescaled for us in the program before digital I/O pins get defined as being either inputs or outputs and whether there needs to be any filtering or pull-ups on the pins. For each SPI

interface, there needs to be an IRQ, which is the Eclipse analog of an interrupt service routine. However, for our SPI interface, the IRQs exist only to avoid compilation errors and warnings, as the code that we have running on HydroWatch continuously runs without being interrupted by these routines. The main code of the sample program addresses the SPI interface with a *spi#= &Driver_SPI#, spi#->Initialize(SPI#_Master_CallBack),* and *spi# -> Power Control (ARM_POWER_FULL)*. These commands allow the RSL10 to prepare SPI pin interfaces, with the pin definitions being completed in the included RTE_Device.h file, and then powering up those different pin configurations from the ARM processor.

A typical SPI interface has four main signals, which are MISO (microcontroller input, slave output), MOSI (microcontroller output, slave input), CLK (clock), and SS (slave select). Typical functionality for an SPI interface is that the SS signal goes low, clock signals are sent out for a certain amount of data bits, which are sent over the MOSI line to the slave. To read data back from the slave, a similar stream of events take place, wherein data usually must be sent in order to receive information from the slave.

The TI AFE4490 chips were selected due to their previous use in other senior design projects and the availability of the AFE4490 evaluation and development board, which included a pulse oximetry clip. Each AFE4490 can control the flash rates of two LEDs and is also capable of discretizing the current readings from one photodetector into a voltage reading between -1.2V and +1.2V. The typical application of an AFE4490 chip is in pulse oximetry, which essentially is just flashing red and near IR LEDs and trying to detect the pulsatile differences in blood flow in a fingertip. The AFE4490 can also control the light intensity of the LEDs, how many samples are taken per LED cycle, and taking ambient light values to be subtracted out later to allow us to

determine light contributions from single wavelengths. For our purposes, the AFE4490 presented an opportunity to select four different wavelengths of light that would contribute the most to being able to provide biological information about the hydration level of a human test subject. The exact wavelengths of light chosen and the rationale behind those decisions is discussed in the LED and photodetector subsystem explanation.

There are two primary system calls that occur that allow the RSL10 to communicate via SPI to the AFE4490 chips. The two types of commands that can be used are “write” and “transfer”. Write simply sends the desired command to a specific register in the AFE, while transfer should send the address of the register to be read and consequently can receive the contents of that register when the AFE is in read mode. To create the “write” function, the SPI interface must be put into master mode, the chip select must be lowered, and the number of bits to send as well as the speed in bits per second must be specified. The speed that we selected for our data transfers was 100 kbps, which was fast enough for the 200 ms updates of the LED readings on the photodetector. Then, an unsigned character buffer has to be initialized to hold the different values that are to be sent, which are then sent using a simple *spi#->Send(buffer, sizeof(buffer))* command followed by waiting for the SPI event to be completed before raising the chip select again, causing the SPI interface to go into inactive mode and no longer act as the master on the MISO, CLK, and SS lines. The required timing and typical waveforms for a write command are demonstrated below in Figure 3, from the AFE4490 datasheet.

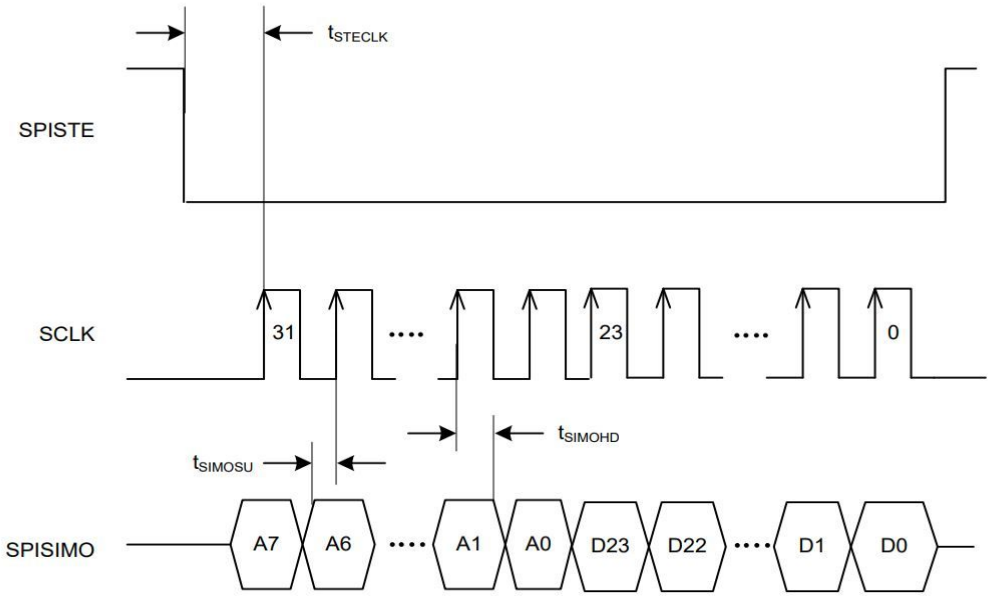


Figure 2. Serial Interface Timing Diagram, Write Operation

Figure 3. AFE SPI Timing Diagram

In a similar fashion, the "transfer" function consists of setting the RSL10 pins into active master mode by lowering the slave select, specifying number of bits to be sent and the speed to send them, and initializing a buffer for the data to be sent. The difference between "write" and "transfer" is that a buffer is also initialized for the received information from the AFE, which allows us to actually transfer the data using a *spi#->Transfer(buffer, received, sizeof(buffer))* command. After waiting for the transfer to be completed, the SPI interface then sets the slave select high and takes the system out of master mode and puts the interface into inactive mode. The required timing and shifting of data is outlined below in Figure 4, directly from the AFE4490 datasheet. This diagram shows that in order to receive information back from the AFE chips, you must send 16 bits of useless information, which is more necessary simply because the

clock signal is only sent when the chip enable line is lowered, and those are the only times when data is actually transferred via SPI.

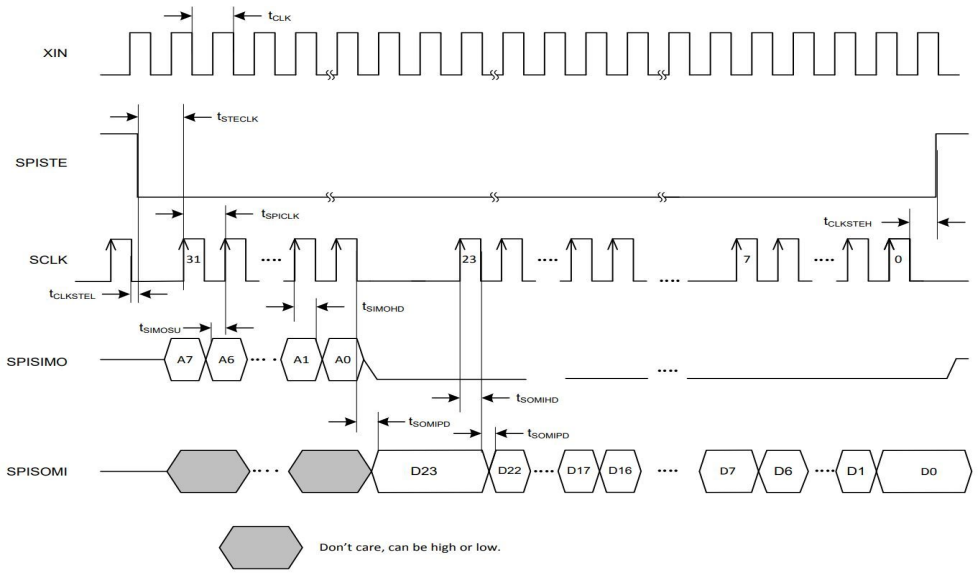


Figure 4. AFE4490 Full Serial Timing Interface

But, the AFE4490 accepts 24-bit commands, with the first 8 bits representing the register address and the last 16 bits containing the data to be written to the register. To accommodate these command types, special functions were created, namely *AFEwrite(), AFEwrite2(), AFEtransfer(),* and *AFEtransfer2()*. The number two corresponds to the second AFE chip that is included in the interface, while the unnumbered functions correspond to the first AFE chip.

The *AFEwrite()* functions take as inputs a two byte hex address of the register to be written to and also take six bytes of hex data as an unsigned integer to be the data to be written to the register address. The *AFEtransfer()* functions also take as an input the two byte hex address of the register to be written to as well as six bytes of "trash" hex data (0x000000), to be sent as

throwaway bytes simply to facilitate transfers from the AFE when it is put into read mode. The *AFEtransfer()* functions also must take a pointer to the receiving buffer for the data from the AFE.

These two basic functions were then used to develop the more complex functions that help initialize the LED flash rates, the LED intensities, and whether the AFEs were in read or write modes. The *AFEtimerinit()* function sets up the various timers and conversion steps and ticks for the AFE to include when waiting to make measurements. The *configAFE()* functions controls important aspects of the AFE functionality, such as current gain, reference voltage settings, LED intensities, and transimpedance amplifier gains for the photodetector readings. The *AFEreadmode()* functions put the AFE into read mode by writing 0x000001 to register address 0x00, which is the Control1 register of the AFE.

The register address 0x22, which is the LEDCNTRL register for the AFE, sets the LED current settings, subject to what the reference voltage selected is. For our purposes, we used a TXREF voltage setting of 0.75V, and we selected different current settings according to the following equations. For each of the LEDs, we decided to have full-scale currents as large as possible, so for each LED1 and LED2, the corresponding maximized current value was around 150mA through the LEDs. This decision was made based upon data collected in the lab set-up, which demonstrated that as the current through the LEDs increased, the differences between heavy water and normal water were much more noticeable at these higher current values. A full discussion of the LED wavelengths and selection of those wavelengths is provided in that

subsystem explanation section.

Table 6. Full-Scale LED Current across Tx Reference Voltage Settings(1)

LED_RANGE[1:0]	0.75 V (TX_REF[1:0] = 00)		0.5 V (TX_REF[1:0] = 01)		1.0 V (TX_REF[1:0] = 10)	
	I_{MAX}	V_{HR}	I_{MAX}	V_{HR}	I_{MAX}	V_{HR}
00 (default after reset)	150 mA	1.4 V	100 mA	1.1 V	200 mA	1.7 V
01	75 mA	1.3 V	50 mA	1.0 V	100 mA	1.6 V
10	150 mA	1.4 V	100 mA	1.1 V	200 mA	1.7 V
11	Tx is off	—	Tx is off	—	Tx is off	—

(1) For a 3-V to 3.6-V supply, use TX_REF = 0.5 V. For a 4.75-V to 5.25-V supply, use TX_REF = 0.75 V or 1.0 V.

$$\frac{LED1[7:0]}{256} \times \text{Full-Scale Current} \quad (6)$$

$$\frac{LED2[7:0]}{256} \times \text{Full-Scale Current} \quad (7)$$

Figure 5. AFE4490 LED Current Selection

Another important register that gets set periodically throughout the program when one AFE is being communicated with is register address 0x23, which is the Control2 register, and the specific bit that needs to be set is bit 10, which enables a tristate mode for the AFE that is communicated with. This creates a situation where when an AFE is not being communicated with it should not control any of the shared communication lines while the other AFE is being communicated with. Once the AFEs are correctly initialized and the LEDs are flashing at the proper rates, both AFEs can be put into read mode so that the values that the photodetector registers can be read and transferred back to the RSL10. The registers that are being read in the main HydroWatch program are register addresses 0x2E and 0x2F, which correspond to LED2-ALED2VAL and LED1-ALED1VAL, respectively. The reading generated and stored in these registers are the photodetector readings of the LED values measured at the detector and subtracting away the ambient level of the LED from that reading. In this way, we can figure out exactly how much of a voltage is generated by each specific LED considering ambient light

readings, and we can thus figure out direct contributions from each wavelength. This allows us to pick specific wavelengths of light for the purposes of watching absorbing masses in the human body, such as water concentration, skin contact, oxygenated hemoglobin, and deoxygenated hemoglobin.

The photodetector current values get converted through an ADC into voltage values, according to the figure below Figure 6. In this way, the photodetector readings that we can retrieve from the AFE are discretized and can be calculated and graphed when these values are sent back to the RSL10 through the transfer functions to the RSL10.

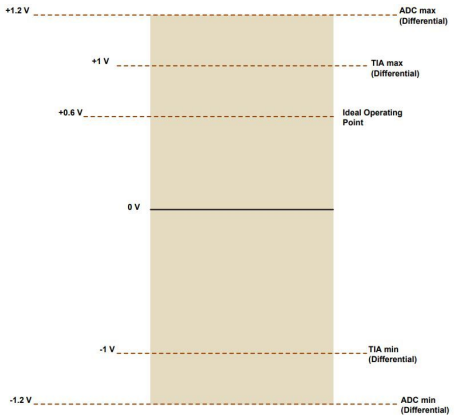


Figure 135. AGC Loop

The ADC output is a 22-bit code that is obtained by discarding the two MSBs of the 24-bit registers (for example the register with address 2Ah).

D23	D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12
Ignore		22-Bit ADC Code, MSB to LSB									
R/W-0h		R/W-0h									
D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
22-Bit ADC Code, MSB to LSB											
R/W-0h (TBD register correct?)											

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 7 shows the mapping of the input voltage to the ADC output code.

Table 7. Input Voltage Mapping

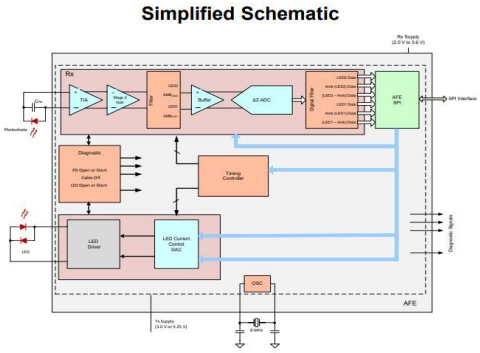
DIFFERENTIAL INPUT VOLTAGE AT ADC INPUT	22-BIT ADC OUTPUT CODE
–1.2 V	1000000000000000000000
$(-1.2 / 2^{21})$ V	1111111111111111111111
0	0000000000000000000000
$(1.2 / 2^{21})$ V	0000000000000000000001
1.2 V	0111111111111111111111

Figure 6. AFE ADC Discretization Scheme

Hardware:

The AFE 4490 has its own 8 MHz crystal oscillator, which helps facilitate all of the AFE timing and command executions. The AFE itself has minimal external decoupling capacitors

required, but the most important pins that we had to consider were the RESET, SPISTE, SPISIMO, SPISOMI, SCLK, and AFEPDN pins for basic functionality of the chips. The RESET and AFEPDN pins were active low, meaning that when the pins were grounded, the chip would either be reset or put into power down mode, which would not be ideal for continuous measurements being taken, so pull up resistors were required to keep these signals high until a digital I/O pulls the signal low to reset or power down the chips. The SPISTE, SPISIMO, SPISOMI, SCLK are the four pins that the SPI interface consists of. The other important pins of the AFE4490 are the TXN/TXP pins and the INN/INP pins. The TXN/TXP pins connect to the LEDs and are where the pulsed or flashing logic set in the registers of the AFE gets realized. The INN/INP pins are connected to the photodetector and are where the incoming current gets sent before the internal ADC of the AFE converts that current reading into a discretized voltage reading. These different systems and all the interfacing can be demonstrated and seen below in the simplified schematic, provided by the TI AFE4490 datasheet.



Also included here is the actual schematic from the HydroWatch board in Figure 7. As can be seen, there are eight decoupling capacitors on various pins of the AFE4490 as well as a very important crystal oscillator with balanced capacitance to ground. Other features of the

hardware schematic that are important are the orientations of the two LEDs, which can be seen in the bottom right of the figure. The way that these LEDs are pulsed is pictured in one of the two operating modes of the AFE4490. We selected this orientation for the LEDs because this mode of operation was what the chips default to when reset. Thus, when the timing registers are properly set up, each of the LEDs takes a turn lighting up, but to the bare eye, it seems that the LEDs are constantly flashing on. This is facilitated through positive pulses being sent out, alternating between the TXN and TXP pins, which effectively selects which LED from the H-bridge configuration to be turned on at a specific time. In a similar way, the photodetector is connected to the AFE4490 by the INN and INP pins, with the INN side corresponding to the anode of the detector and the INP corresponding to the cathode of the photodetector.

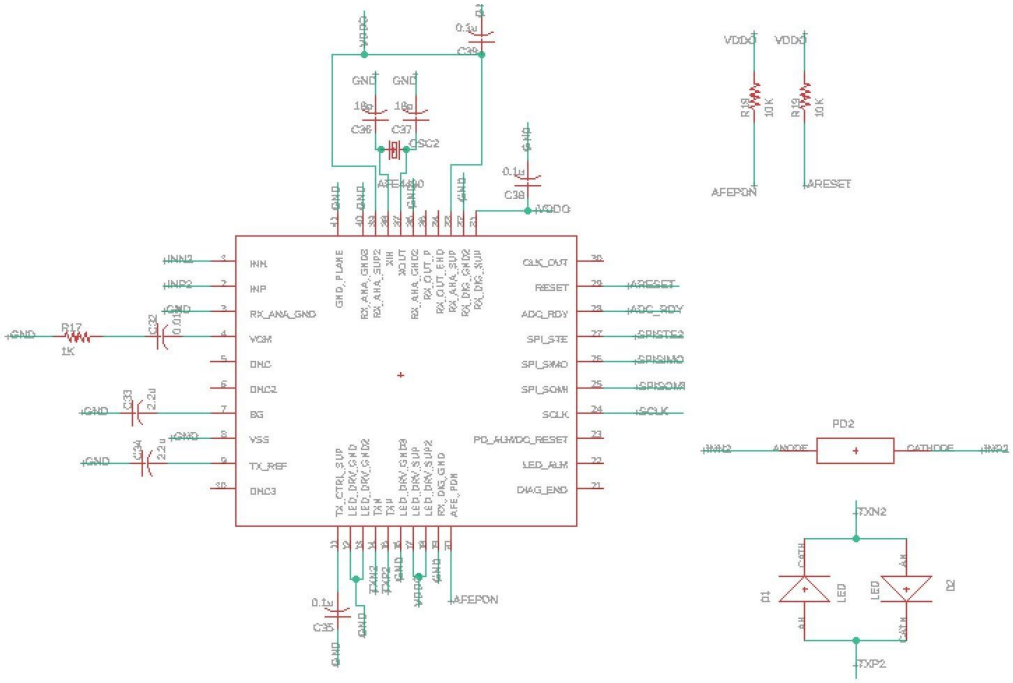


Figure 7. AFE Hardware Schematic

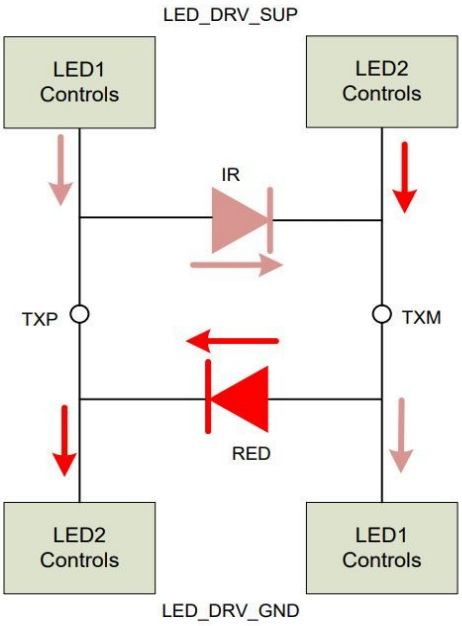


Figure 133. LEDs in H-Bridge Configuration

Figure 8. AFE LED Configuration

Subsystem testing:

To gain initial understanding of the command formatting and interfacing with the AFE4490 from a very basic perspective, we used a PIC32 Kitboard from the Senior Design room to develop a simple SPI interface. This SPI interface used the same writing and reading type data transfers that we later developed for the RSL10, but we were able to hook up the PIC32 board to two separate AFE4490 breakout boards and see a pulse oximetry clip lighting up with the correct flash rate that was dictated by our register values. SPI interface functionality testing was aided by the extensive use of the Saleae Logic Analyzer, which allowed us to decode SPI transactions and figure out the required timing for our commands. Then, in order to demonstrate that register values could be read in real time from an AFE, we connected an LCD screen to the Kitboard,

which allowed us to see the register values in real time. We then read off the LED values from the pulse oximetry clip and were able to see the pulsatile nature of oxygenated hemoglobin, as measured by a near infrared light.

After figuring out the entire interface and the correct commands to send from the PIC32, we had to develop the RSL10 interface. This interface was tested using the RSL10 development boards connected to a single breakout board that we created for the AFE4490. These breakout boards allowed us to set registers correctly and connect to the pulse oximetry clip that was included with the AFE4490 development board. To check proper SPI functionality, we extensively used the Saleae Logic Analyzer to decode the various SPI transactions and determined that one SPI interface was functioning well enough. Then, we developed a second SPI interface that shared the same pins for MOSI, MISO, and CLK, but had a different slave select line. We then hooked up the development board to two of the breakout boards for the AFE4490 and put the logic analyzer on the different signals and observed stable SPI communications on the RSL10 development board. By requesting register readings from both breakout boards at once and displaying them in the command window of Eclipse, we were able to successfully demonstrate two functioning SPI interfaces returning the expected register contents from the RSL10.

One issue that we ran into in developing the RSL10 SPI interface was that there seemed to be a surprising amount of noise present on the MOSI line, meaning that the AFE response to the RSL10 master commands were not correct at all. Upon further inspection with the logic analyzer, there were many spikes on the MISO line, which meant that the correct commands were not being sent in the first place to our AFE chips. Even more concerning was the fact that

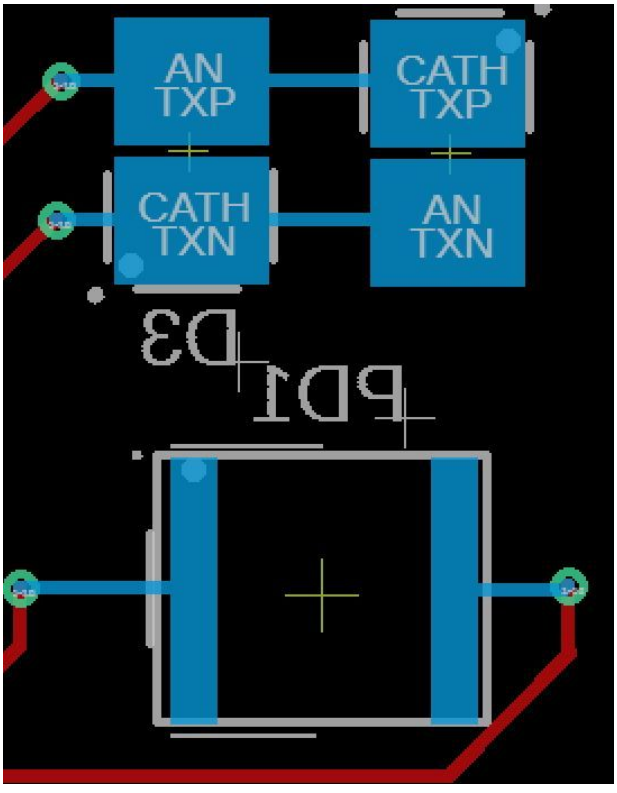
the noise on the MOSI line was almost an exact clock signal, which was enough to convince us that something was happening with interference from the two AFE chips fighting over the shared SPI lines. Due to the fact that the clock signal crossed over the MISO line, it seemed as if there was high frequency interference occurring, and the solution to this issue was figuring out how to tristate the AFEs such that when we were not communicating with an AFE, it would release control of the SPI lines and allow the other AFE to take over completely. This tristate register bit alleviated the issues with the SPI interfaces on the RSL10 development boards.

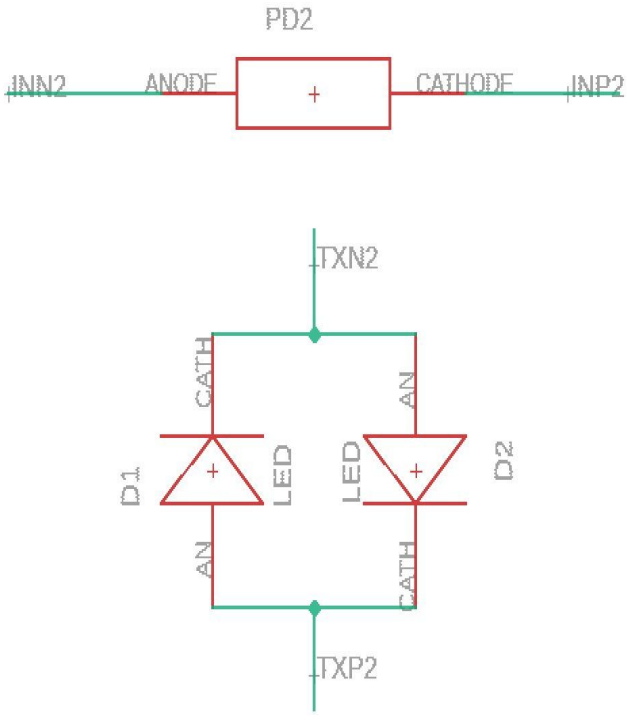
On the full HydroWatch boards, we noticed a similar issue arise on the MISO line from the AFEs during SPI communications. At first, we believed that this issue might have arisen due to timing from the Bluetooth functionality of the program, such that data was being shifted out too quickly than was desired by the AFEs. However, upon further investigation into Bluetooth timing and attempting to increase the time between notifications as well as decreasing the time that the RSL10 is stuck in the advertising mode, but the issue persisted. We were able to observe the correct register values on the first AFE, but the second AFE register values exhibited the similar noise values reminiscent of the clock signal. This led us to inspect the board and try to figure out if the layout played a role in the issues. What we were able to determine was that the second AFE has its MISO line on the top side of the board crossing over the clock signal on the bottom side of the board at a right angle. Running the two SPI interfaces off the RSL10 evaluation board and connected to the two AFE breakout boards, we were able to get stable register readings. But, when we downloaded the very same program to our custom HydroWatch board, the MISO issues persisted. We hypothesize that the crossing of these traces in that orientation has led to noise being always present on that line from the second AFE, and this

hardware issue is likely to remain unless future renditions of the boards feature clock signals on a different digital I/O pin of the RSL10 to avoid this crossing over of critical signals. In our final board, this noise was present only on the 970 nm LED, which is clear due to the 0xFFFFFF, which is the same as a clock signal overwriting very small or zero values. To correct for this, we would lay out a separate clock signal for the second SPI interface that does not cross underneath the MISO line, such that there would no longer exist this interference.

5.4 LEDs and Photodetectors

Subsystem Requirements: The wavelengths of LEDs selected for use in HydroWatch must correspond to biologically significant concentrations within the human body. The photodetectors must also be capable of converting the chosen wavelengths into a measurable





current value, such that the AFE4490 can then convert that reading into a usable voltage to be sent out via Bluetooth.

Figure 9. LED and photodetector schematic and board layout

As can be seen in the schematic of the HydroWatch board in Figure 9, two LEDs are connected in antiparallel fashion, such that the TXN and TXP traces from the AFE4490 can pulse each LED on and off. Also pictured in the schematic are the photodetectors, which are controlled via the INN and INP traces, which return current readings to the AFE4490. The board layout of these components is critical for receiving the actual signals that we desired. In order to account for possible surface reflections and keeping the source from directly entering the photodetector, the LEDs and photodetector were separated by 7.8 mm, which was discussed in Professor O'Sullivan's Biophotonics course. Having this source-detector separation alleviated the possibility of direct interference from the source with the detector and forces the light emanating from the sources to go deeper into the skin in order to be detected. In this way, the layout on the board helped facilitate actual interaction with biological concentrations within the human body.

The LEDs and photodetectors are placed on the back of the main HydroWatch board so that the user can comfortably place the device on their skin without having to worry about external light readings affecting the photodetector. The AFE4490 is only capable of discretizing two LED light readings from the photodetector. Because of this, we needed to minimize any sources of extraneous wavelengths of light from the photodetector. To accomplish this, the two pairs of LEDs and photodetectors were placed on the back of the board far away from each other and not in a direct line with the other's sources. The photodetectors were also placed further

towards the edges of the board so that the other LEDs would interfere with readings as little as possible. These hardware decisions all were made for the sake of minimizing error due to extraneous wavelengths of light being present incident on the photodetector. Four wavelengths of LEDs were selected for use on the board. The first was a 450 nm LED, which was used to determine whether an adequate amount of skin contact was being made with the device. This LED was chosen because at this wavelength almost all light that is sent into the tissue is absorbed. This means that when the device is fully in contact with the skin, very low voltage values would be read by the AFE chip, and if the device was even slightly elevated off of the skin, relatively high voltages would be detected. We set the level to be that anything below 0.01 uA of photodetector current would be categorized as good skin contact, while any readings above this value would be categorized as poor skin contact. Next, we utilized a 1200 nm LED for our water detection capabilities. As seen in Figure 10**,** the water absorption spectrum has a relative maximum at roughly 1200 nm. This means that change in the amount of 1200 nm light absorbed is based on changes in water concentration levels with in the skin. We selected this wavelength due to the relative peak in water concentration level, the lack of a similar peak in other molecules found in the human body such as hemoglobin and deoxyhemoglobin, and that using higher wavelength LEDs would have resulted in much higher cost which was not ideal due to the budget constraints on the project. The other two LEDs were 660 nm wavelength and 970 nm wavelength. The data gathered from these LEDs was not as useful due to a number of factors. For the 660 nm LED, our intended goal was to use this to detect changes in hemoglobin, which has a relative peak in its absorption levels at this wavelength. However, to detect the changes in hemoglobin that change based on the heartbeat, we would have needed to be

sampling much faster than the once every 100 ms we were sampling at for our other LEDs. In order to fully capture an average heartbeat, our sampling rate would need to sample upwards of 40 Hz, which is something that could be improved within our device at a later date. The 970 nm LEDs were chosen as another peak in the absorption spectrum of water was roughly at this value, shown again in Figure 10. Analyzing the data from this LED would have given us a second set of data points for water concentration levels, and given us more assurance about the hydration levels of the person. Unfortunately, our board layout consisted of a trace connected to the 970 nm LED intersecting that with one of the SPI clock signals. This signal interrupted our attempts to capture data from the photodetector, which led to the information received being at 0 almost always, rendering it useless. In future iterations of the device this part of the board would have to be redesigned, which is covered in the To-Market Design Changes section of this report.

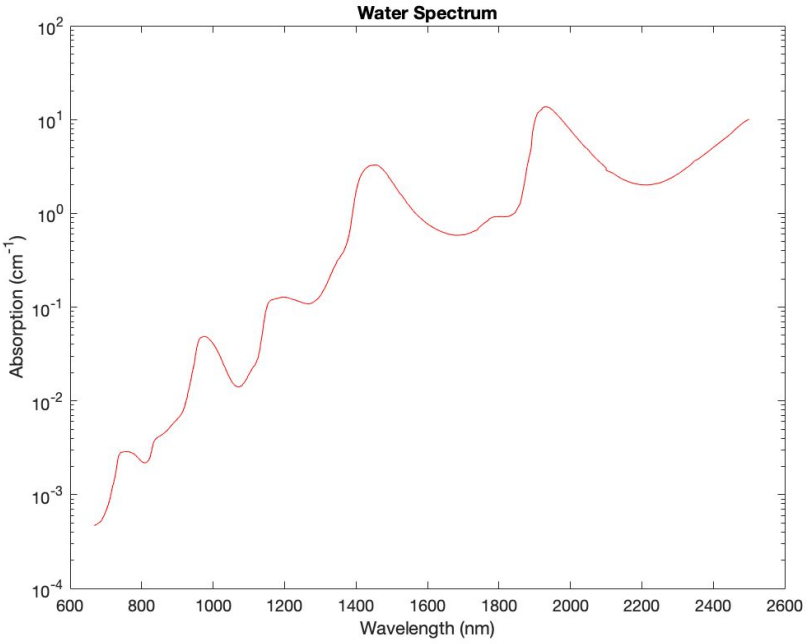


Figure 10. Absorption Spectrum of Water

Two different photodiodes were selected for use on this device. The first was a TEMD5010X01 Si photodiode from Vishay Semiconductors, which was used to detect the changes in light absorption for the 440 nm and 660 nm LEDs. The second photodiode was a MTPDP1346 InGaAs photodiode from Marktech Optoelectronics, which was used to detect the changes in light absorption for the 970 and 1200 nm LEDs. The surface mount photodetector was able to be soldered to the board normally. However, one week removed from demonstration day when we determined that the 1200 nm LEDs were showing more promising data than the 970 nm LEDs, the only possible photodiode to measure this wavelength, the Marktech Optoelectronics photodetector, was a through hole component, and clearly would not be able to be soldered direct to the SMD pad on our board. This resulted in a makeshift attachment to our board that allowed for the through hole photodetector to be attached to the board and soldered into the pads that were designed for a surface mounted photodiode. This did not appear to cause any issues in our results, but future boards should be updated to include a place for a through hole component, or surface mounted photodetectors of the same wavelength range should be obtained. An adjusted board design with the through hole component can be found on the team website.

Subsystem Testing:

In order to test surface mount device LEDs, we had to develop simple test boards with a similar layout and orientation that we would have on the final HydroWatch board. These test boards contained two LED footprints and one photodetector footprint, with the corresponding TXN/TXP and INN/INP header pins to control the LED current and read out the photodetector current. To create the footprint for the various LED sizes that we needed to test, the datasheets of

three different LED packages and suggested solder pads were compiled to create these LED footprints. The source-detector separation on these boards was 11.3mm, which is a significantly larger separation than the final HydroWatch board design. The further separation on these small test boards allowed us to conduct our various experiments and get even deeper measurements of the concentrations we were looking for. Once we had proven that these concentrations could be easily sensed with a very deep measurement, we settled for a smaller overall detection distance to save some space on the final board design.

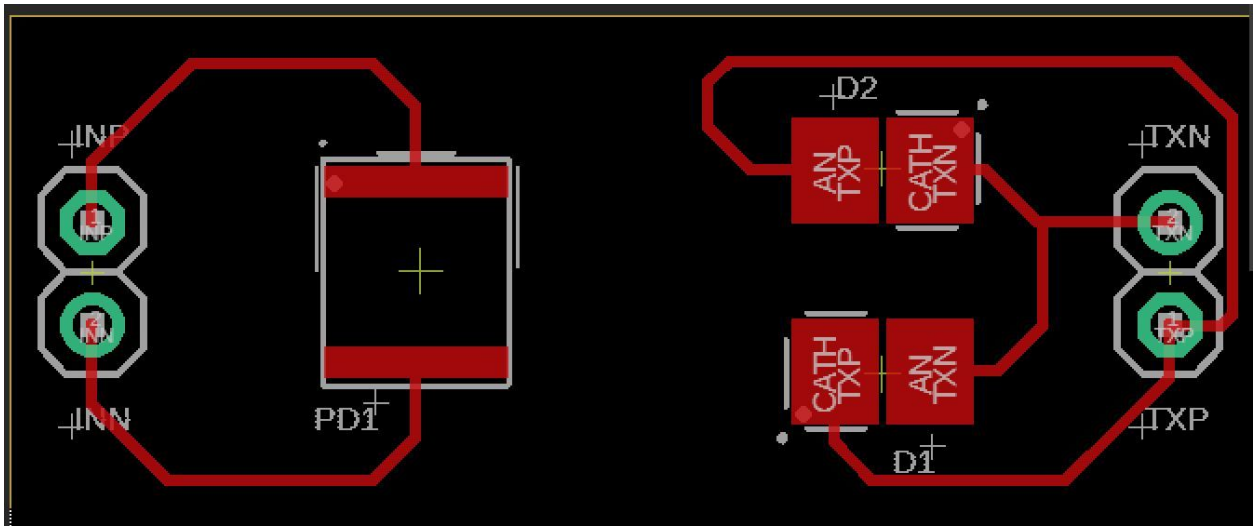


Figure 11. Simple LED and photodetector testing board

We had three different tests setups that were used throughout this process before progressing to our entire system tests. Our first test was to establish that our LEDs were operational and to see what values of optical power they gave off at specific current levels. This allowed us to determine what current values we should use to obtain specific optical output levels for each of our LEDs. Our second test used a series of cuvettes that contained two different liquids. One cuvette contained 2 mL of regular water, and one cuvette contained heavy water, which contains deuterium particles. These two substances, while appearing the same to

the naked eye, have differing absorption spectrums. This allowed us to shine light at specific wavelengths through each cuvette and see if the predicted values for the amount of current generated by the photodetector was in line with our theoretical results. This test allowed us to confirm that a water was able to be detected and that our wavelengths were correct for sensing changes in water compared to other very similar liquids. The final proof of concept test was to test the device on a sponge. We soaked a sponge in water, tested the value of light being read at the photodetector at several locations on the sponge and at several different current input values, and then averaged the recorded photodetector levels. This data can be seen in Figure 12. This test showed that even small changes in the mass of water present at the testing site was detectable by our device, and gave us confidence moving forward that we could detect these changes in water concentrations within a person.

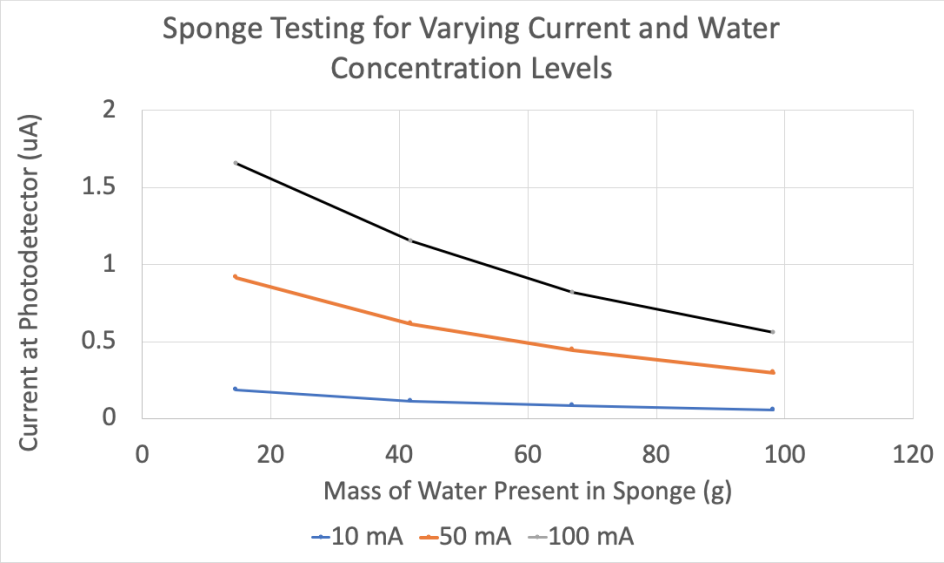


Figure 12. Sponge testing results for varying current and water concentration levels

5.5 Bluetooth Custom Protocol Communication

Figure 12. Bluetooth subsystem block diagram

Subsystem Requirements: Construct a custom bluetooth LE protocol that is specifically designed to broadcast the information from the SPI communication with the AFE4490s.

As stated previously, the RSL10 was selected due to its small size and bluetooth capabilities. The custom protocol in place consists of over 60 files, many of which are specific to SPI communication and the RSL10 CMISIS driver. The custom protocol was written in C and is based on the prepherial_server_uart code that accompanies the RSL10 development board. The example code includes a battery service, receiver/transmitter uart and an example custom service that is looking at identifying TX/RX locations and addresses. We have taken the structure of the

custom protocol in terms of GAPP and GATT setup, and overhauled the code to communicate with the AFE chips and output 8 bytes of LED data. The battery service protocol is not called with in the code for this HydroWatch device, but can be found in the files due to the service being interwoven with the BLE stack created by the RSL10.

At a high level, when the code is run the GAP and GATT stacks are initialized and constructed. The application itself is then initialized followed by the SPI. At this point, if no errors are detected, the device is set to advertise as "HydroWatch." The system stays in advertising mode until a bluetooth connection is achieved. Once a connection is established, the application process, "*app_process.c*", is immediately started and the main function "*app.c*" is called. As long as the device remains in the connected state, the device will output the four LED readings from the subsystem above as determined by the custom protocol. The custom is set up to enable notifications to be advertised without the client, or external device, having to manually interrogate the peripheral, the HydroWatch device.

Notifications are sent via the custom protocol *send_notification()* function. The photodiode values for the notifications are read within the "*app_process.c*" file. For insight in to how the LED values are generated and collected, please see the SPI and LEDs subsystem sections. Once the values are collected, they are advertised by the peripheral (HydroWatch) to the client (an external machine) with unique UUIDs (bec1, bec2, bec3, bec4) and names (LED1, LED2, LED3, LED4). The construction and formatting of the custom protocol can be found in the *ble_custom* main and header files of the same name.

Within *ble_custom*, we create our unique service. We start by creating a custom service message, or a unique identifier for the service when it is discovered by bluetooth enabled devices

known as clients. The GATT stacks are then set up and the service is setup to start broadcasting. If there is an issue in the stack, the program will go no further. If there is no issue, we release the identifiers of all the components of the custom service that are able to be read. In bluetooth, a service component can either be read, written to, or send out notification. A read requires both a request from the client and an accept from the peripheral. A write is a command from the client to the peripheral or device running the bluetooth. A notification is like a read in the sense that the device broadcasts a value, however, a notification does not require a request from the client. The peripheral automatically pushes out information based on what is written in code.

In our case, we release four different identifier in our custom protocol, LED1, LED2, LED3, and LED4, which hold the values from the four different LEDs on the HydroWatch board. We configure the components to be readable, writable, and to have the ability to send notifications to a client. In our application of the device, we rely upon the notifications to generate our data. The fastest that the ble can transmit data is every 10ms or at a frequency of 100 Hz. For our purposes, notifications are sent every 100ms. The overall bluetooth code is based on the Bluetooth protocol already in place. This structure looks for all services present in the code and then constructs the GAPM stack as long as no errors are present. The system is then configured to have a unique identification number and addresses. The GAP deals with connectivity and advertising. All code related to the overall bluetooth is found in *ble_std.c*.

After the service has been configured to be readable, writable, and to send notifications, we are ready to integrate into the overarching bluetooth and the AFE spi communication. The LED values from gathered from SPI are written into four different variables mentioned previously. These variables are then inverted to be sent over UART, so the reading directly to the

client will be in the proper order and will require little postprocessing. Once these have been adjusted, the custom service is called to send the values directly to the bluetooth explorer program, discussed next, every 100ms.

It is important to note that the functioning of the bluetooth is reliant upon the connectivity of the device. When testing the bluetooth, we performed tests exclusively in the Senior Design room. On demo day, we noted that the bluetooth connection was unable to maintain connectivity for prolonged periods of time. We theorize that this was due to all the competing devices in the room. For future projects that use bluetooth, we might suggest testing under different setting and looking into different antennas than the one we used.

5.5 Postprocessing

Figure 13. External processing subsystem block diagram

Subsystem Requirements: Information gathered by the previous subsystems should be transmitted to an interface that allows the end user to interact and track their relative hydration levels. The user should be able to subscribe to text or email updates so they are not tethered to the processing machine.

The final subsystem is the external processing of the information coming out of the previously described subsystems. To establish the bluetooth connection, we made use of a RSL10 dongle. This dongle is manufactured by OnSemiconductors, the makers of our microcontroller the RSL10. We selected this dongle due to it ability to support both Master and Slave interactions with our device and it supporting multiple devices at once. A screen capture of the environment is seen below.

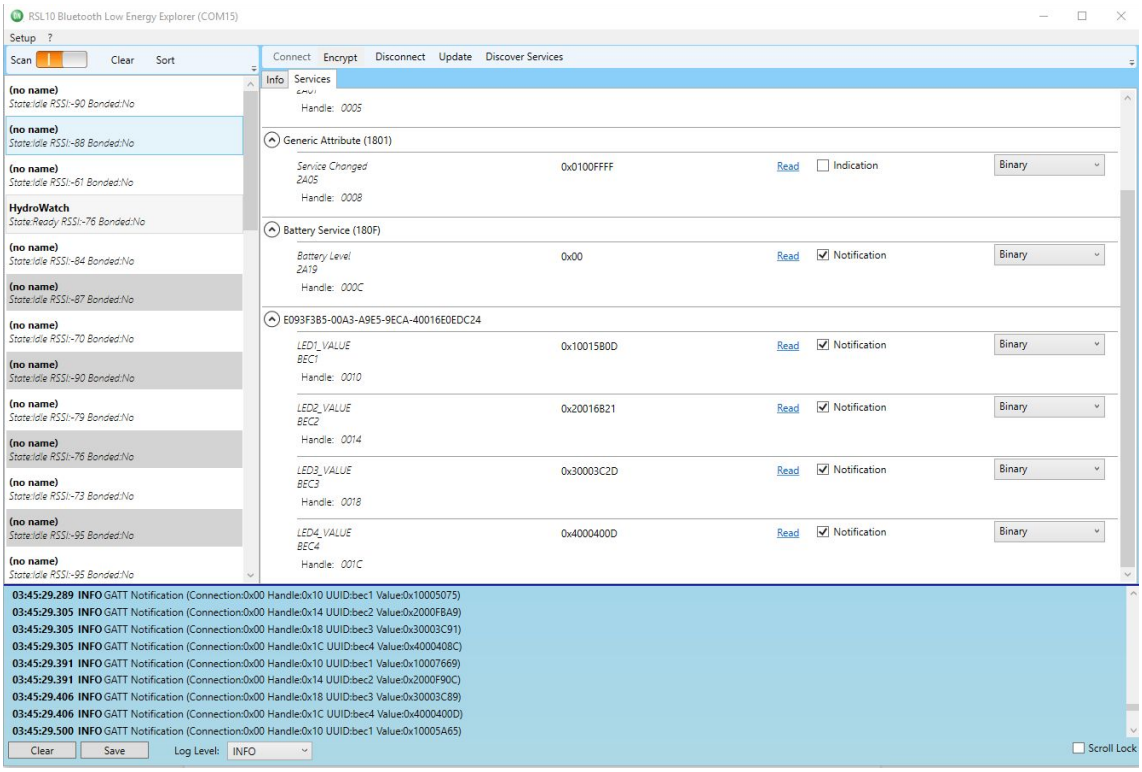


Figure 14. Screen capture of RSL10 dongle client explorer

This dongle acted as a general central client that supported both slave and master information via a service call “Bluetooth LE explorer” that is put out to complement this particular dongle. In the case of our internal chip software, we only required the device to receive information for the HydroWatch via notifications sets in the program. These notification were collected in a terminal in the environment until we disconnected the device or the connect was interrupted. The terminal window is then saved as a log file.

After generation of the log file, the file is passed to a MATLAB graphical user interface (GUI) , pictured below.

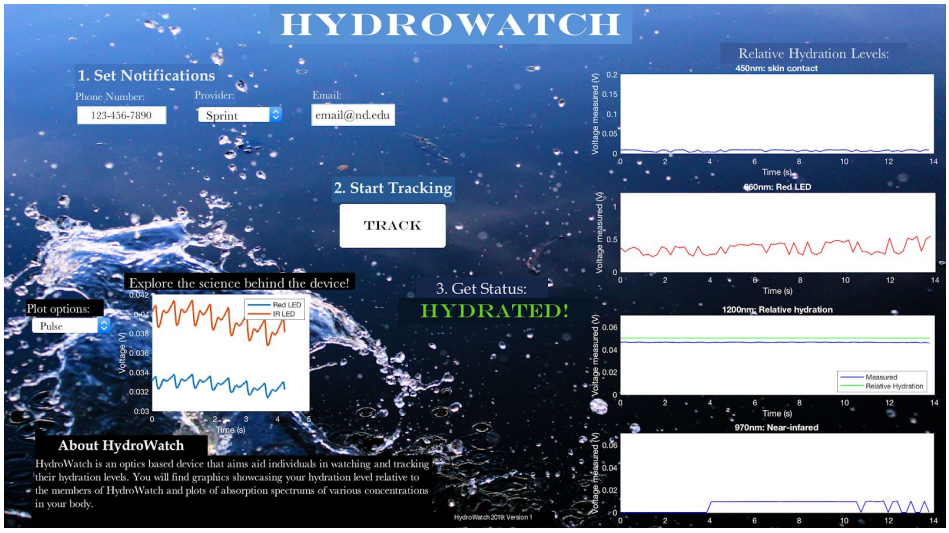


Figure 15. Screenshot of the HydroWatch MATLAB GUI

Once the track button is pressed, MATLAB does a system call of python to run a specific file called “parselogfile.py” written specifically for this application by our team. This files goes through and pulls out the six byte hex value of each LED notification that is compiled in the log

file. MATLAB was chosen as the language/interface for the post processing of the data due to familiarity with in the engineering community and ease of creation of a visually appealing GUI. MATLAB also has to ability to interface with a variety of different languages, including python, which is another language that the team is comfortable with.

As mentioned previously in the custom protocol section, each LED reading has two leading bytes that serve as identification codes for the LEDs. These identification codes are filtered out, along with the two bytes of data, and a file of the following form is generated.

LED_code	Hex_Output
1	1000
2	2000
3	007D76
4	4146
1	1000
2	2000
3	7076
4	4102
1	1000
2	2000

Figure 16. File showing LED identifiers and output values

This file is in turn is read back into the main MATLAB loop. The hex outputs from the LEDs are converted to a base voltage reading based on the quantization of the ADC local to the AFE chips (2^21). These conversions are then stored in a vector based upon the LEDs of origin. The system also checks to see if there is noise present in the readings, ie the AFE is giving a reading of 0xFFFF or some reading above the saturation value of 1.2V. If noise is detected, it is cast to a voltage reading of 0V. Once all information from the LEDs as been converted, the data is post processed. LED 1 corresponds to the 450nm (blue) LED which was used to determine is skin contact was made. If the reading in the 450nm led is greater than 0.2V for more than half of the

measurement time, the data is classified as bad contact and the user is alerted of bad skin contact. If good skin contact is made, we are then able to go on to determine our qualitative measurement of hydration.

The reading for hydration comes from LED 3, the 1200nm LED discussed in the LED/Photodiode subsystem. Our system relies upon a baseline measurement that is intrinsic to the wearer of the device. For the sake of demonstration, we set the relative hydration level based upon reads of two HydroWatch members, seen below.

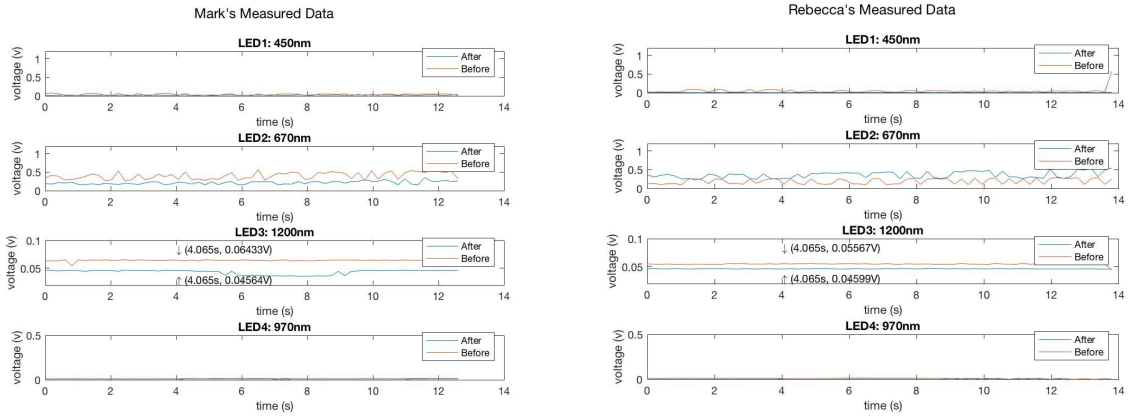


Figure 17. Measured data of HydroWatch members' hydration levels

These measurements were taken after both members did not actively intake a gross amount of water for approximately 24 hours. This set the red baseline, which we defined as dehydrated. Both members proceeded to consume a few bottles of water and then waited an hour for the water to be processed by the body. The device measured the two members again, and we noted that they both dropped to approximately 20% for both individuals. This suggests that more water is present in the skin, because more light was absorbed. We define their final levels as hydrated.

We note that both members are roughly around .05V when they were dehydrated, thus for the sake of demonstration, our relative value was set at .05V as our threshold for a qualitative assessment of hydrated vs. dehydrated.

At this point, the results are released to the end user both on the GUI and by text and/or email. On the GUI, the end user will see the plots of the data collected, while the texts and email update the user whether they are hydrated or not. Both the text and email updates are reliant upon the *sendmail()* function in MATLAB. The text updates rely upon the same principles that companies use to send out mass texts to their customers. Each phone number can be alerted by using the carrier mail handle (for example, @messaging.sprintpcs.com for sprint networks). This method sends a sms to any phone that one knows the carrier of via an email address that you have specified. Note that this is easiest done with a gmail account, but it must be an account in which the security settings can be altered to allow for MATLAB to log in and access the account. This user alerting code can be found on the team website. Note that it will not run for we have removed the password of our account from the code. If you are interested in running the code, please enter your own email and password.

6 System Integration Testing

The coding for the SPI communication was run on the RSL10 evaluation boards with logic analyzers attached to see if SPI was sending out correct info from the RSL10 to the AFE. Once the communication between the RSL10 and the AFE was established, the AFE breakout boards were connected to the RSL10. A logic analyzer was used to see if SPI still received what

was expected based on the evaluation board testing. We used SPI to read in data from the photodetectors and send an identifier along with that data over Bluetooth to a computer via the Bluetooth Low Energy (BLE) dongle from ON Semiconductor. The BLE explorer was used to see basic registers being sent back and forth between the computer and the RSL10. The logic analyzer was used to see if there was any noise present in the connection between the AFE chips and the RSL10. Then we combined the Bluetooth peripheral server UART protocol and the SPI interface with initializations for AFE4490s into one code. The BLE explorer software displayed ambient LED values when we established what our baseline voltage values were. This baseline data ensured that the communication channels (SPI and Bluetooth) were working. We then gathered hydrated and dehydrated data from each of the HydroWatch members. Each time data was gathered, it was saved to a log file and imported to MATLAB. This data can be found on the website under "Documents." In the MATLAB code, the LED reading is converted and scaled. The skin contact of the device is a binary measurement based on the ambient light measurements and when the device is on the skin. There was a clear difference in voltage of the photodetector reading of the 1200 nm light when dehydrated and hydrated levels were compared. These results were used to establish a baseline for hydration within our post-processing environment. Everytime a new measurement is taken, the GUI is updates with voltage incident of the photodiodes for three of the four LEDs and the user is notified via text and email about their hydration status relative to the members of HydroWatch.

The design requirements of the system required us to have effective LEDs and photodetectors that gathered data about the characteristics we wanted to measure, stable SPI and Bluetooth connections, as well as an effective user interface so the information that the user

wants to know is communicated. The testing of the LEDs, AFE, RSL10, and computer communication shows that we were able to achieve stable SPI and Bluetooth communication. The noticeable change in LED readings between skin contact vs. no skin contact and between hydrated vs. dehydrated means we selected LEDs with wavelengths that characterized hydration well. The MATLAB GUI display and the text and email notifications shows that we effectively communicate with the user relevant information about hydration.

7 Users Manual/Installation manual

Our product is still in the development stage and is not set for consumer release. Thus, we will use this section to discuss the set-up and running of HydroWatch as it currently stands in its development stage. Note that various programs need to be installed to run the product.

7.1 How to install your product

Before actual testing and programming of the board can be done, one must first download various software packages. Note that most of these packages operate solely on a Windows machine.

Software Packages needed:

- RSL10 Bluetooth Low Energy Explorer: Found on ONsemiconductors website
- Eclipse bundle: Found on ON Semiconductors website under RSL10 development board
    - We suggest one also reads the accompanying user guide

- J-link debugger
- Java: Note that configuration has to match eclipse (ie x32 or x64)
- MATLAB: active license is required.
- Python 3.x: Easiest to install Anaconda

Download all the above packages in accordance with the prompts set forth by the manufacturers. Once that can been completed, one will need access to the HydroWatch files. After ensure that eclipse is properly set-up, add HydroWatch_UARTfresh to the work space. Build the eclipse project and check for errors. All needed files should be present and the code is commented for the user understanding. One may need to install the CMISIS driver from the RSL10 that is mentioned in the *Getting Started With the RSL10* documentation that is provided by ONsemiconductors. Next, we have to create a debug session that runs off the j-link. Select the green bug on the menu and go to debug properties. Configure the session as seen in the below figures.

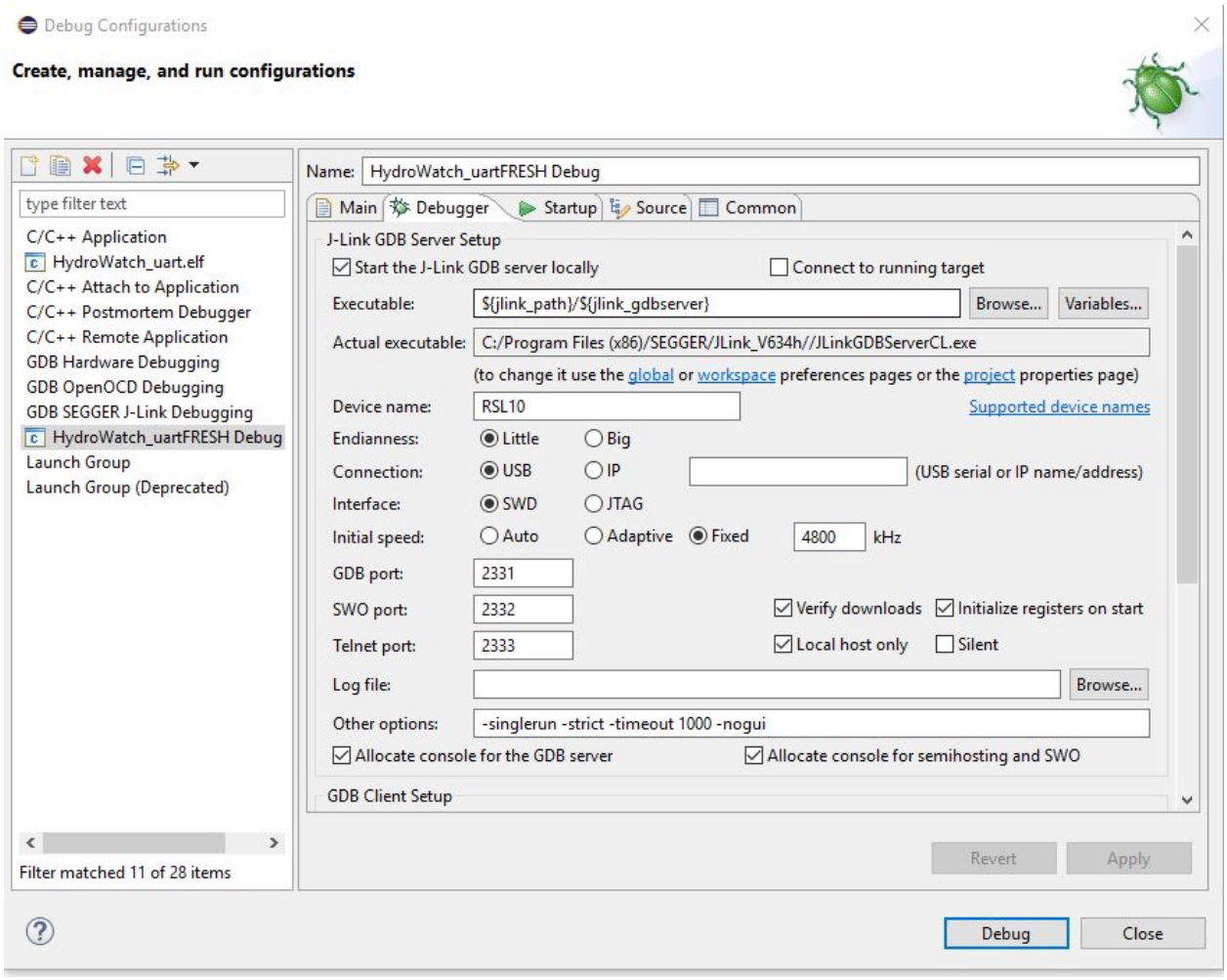


Figure 18. Screen capture of proper debug mode set up

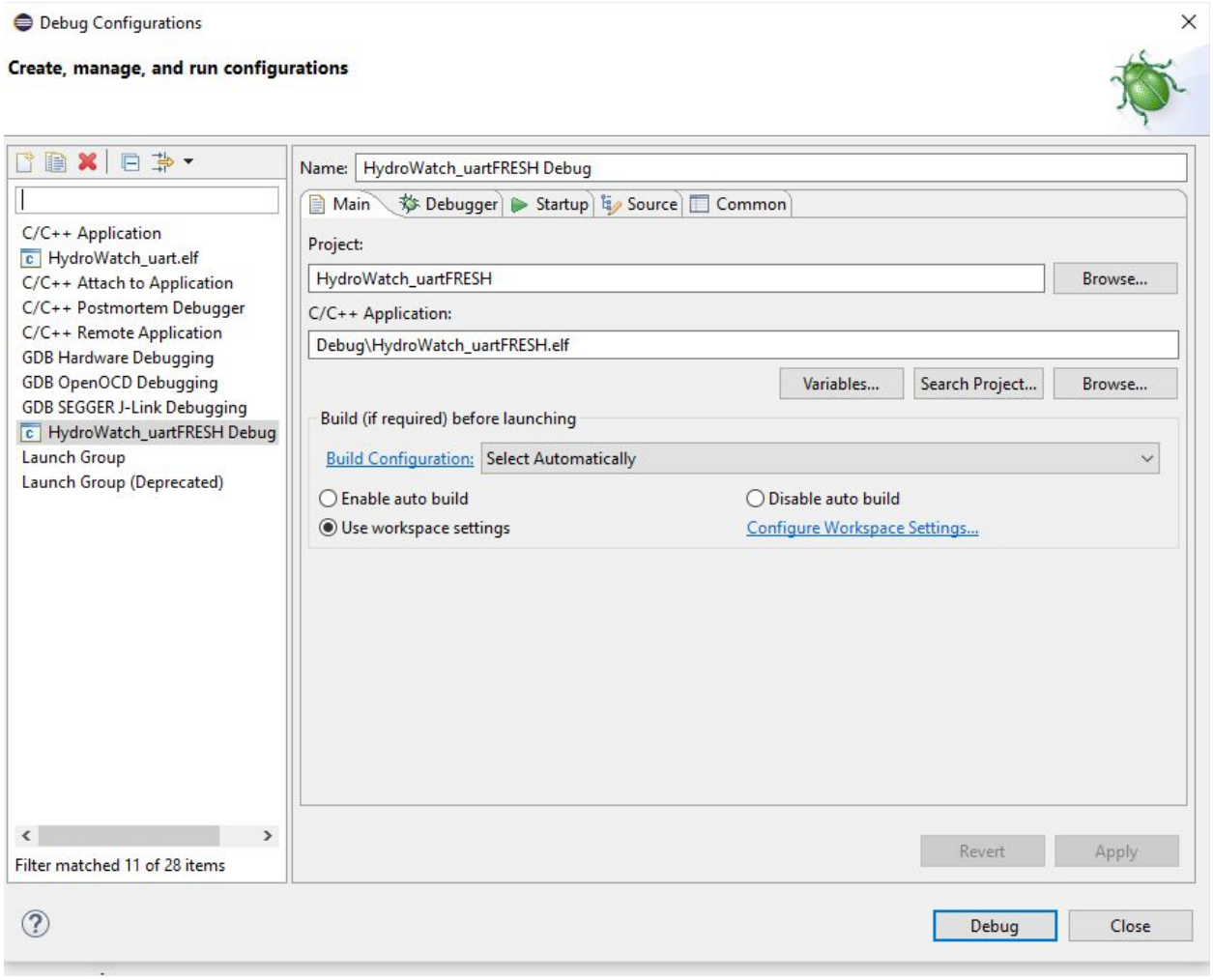


Figure 19. Another screen capture of proper debug mode set up

Test that the debug session launches and accept the terms and conditions. Now the HydroWatch programming environment is set-up, we can move on to installing the post-processing environments.

Launch the Bluetooth Explorer Environment, checking that the dongle is plugged in. Initialize the dongle, place into scan mode and check that you are receiving signals from bluetooth devices. Next launch the MATLAB GUI. Adjust the GUI to fit your screen by

launching *guide* in the command terminal. Finally open the python file *parselogfile.py* and check that the output log file included in the folder is parsed as you would expect with the value being a 2 byte (4 hex value) output. This has been found to differ based on the system one is using. To adjust this value, simple change the subtracted value in second loop in accordance with your needs.

7.2 How to setup your product

Once all the software has been installed and checked, one is ready to integrate the hardware into the system. The hardware needed to set up the system is listed below:

Hardware needed:

- Source of power: mini USB or litho battery
- J-Link
- HydroWatch Board
- RSL10 dongle

Take the HydroWatch board and supply power to the board. Next connect the J-link to the HydroWatch board and check the orientation on the J-Link board. The notch on the connector should face the inside of the debugger board. You are now ready to program the board. If making changes to the board, it is best to run the system in debug mode by using the configuration we specified in the previous subsection. At this point you should be able to see the registers on the board being read into Eclipse. If the registers are not visible, there is a power

issue with either the board or the j-link and connections should be checked. If no errors are found, the development will go into ready mode. Press the green go button to start the CPU of the device, starting the bluetooth advertising.

We now set up and check the Bluetooth connection. Place the dongle in an available COM Port. Open the Bluetooth Low Energy Explorer software and initialize the dongle on the appropriate COM Port. Place the explorer software into scan mode. You should now see the device “HydroWatch” being broadcasted. Select the device and click connect. At this point, the stack is called and you can now record information from the device by selecting “discover services.” If there are no issues in connectivity, the device will send four notifications every 100ms. The data will be automatically compiled to the log file seen the bottom portion of the explorer. To save the log file, click save and save as “capture.log” in the location of the MATLAB GUI. This file will be read automatically when the track button on the MATLAB GUI is pressed. Before the GUI can be run, you will have to go in and change the email account that sends the updates out. The HydroWatch account is closed and not accessed by people outside the creators of HydroWatch. Open the GUI, enter text or email information and press track to check end to end functioning.

7.3 How the user can tell if the product is working

Determining if the device is not properly operating is fairly simple when going through the basic setup described above. You should see the red and the blue lights on the back of the board light up at rather bright intensities. The other two LEDs are IR and not visible with the naked eye. When in doubt, pull a logic analyzer out. Basic errors and checks are listed below.

Bluetooth:

- Device is not connecting for long
    - There may be a fatal error that is not caught by the debugger. If no issues arose when launching debugger and pressing the “Go” button in Eclipse and you see no errors in the Eclipse terminal, then there is an issue in the custom protocol.
    - If a few notifications are logged but then the device disconnects, there is a connectivity problem. Check that the antenna has a clear view of the dongle, and in particular, for best connectivity, orient the antenna directly at the horn antenna of the RSL10 BLE dongle connected to your system’s COM Port.

MATLAB

- Follow the errors that arise in the terminal. Most likely a file is missing or misnamed.

Eclipse

- Target not detected
    - The J-link is improperly connected
    - Voltage regulator is not operating correctly
        - Note if the board is producing heat immediately unplug. You should check for any short circuit connections between the VDO power traces and ground connections, and you may need to replace the TPS61201 part
- 0xDEADBEE

    - The J-link connection is messed up. Terminate the session and restart.
- Debug fails to run
    - The .elf file is being run instead of debug
    - Eclipse is being weird. Close the program and reload
- Warning files missing that are not missing
    - Eclipse is being weird. Close the program and reload
    - Should also go away as soon as debug is run

LED values

- AFEs are off when reading 0x__FFFF
    - There is noise or an issue the the AFE chip. Check for solder bridges and correct placement of oscillators.
- The values should be between 0V and 1.2 V. If values are saturating, turn down the current in the *spi.c* file *AFEwrite()* function.

7.4 How the user can troubleshoot the product

If the user applies what is found above and compares results to that of HydroWatch, trouble shooting should be straightforward. Perform sanity checks on the LED readings to check that you are not reading back only noise.

8 To-Market Design Changes

There are several improvements on the device that could be made before it would able to used by the average person. First, the overall size of the board and its components should be condensed. The board itself is able to be worn on the the arm currently, but it is very bulky and not a feasible size. If components on the board are placed closer together and some potentially unnecessary parts were removed, the device could be made to be a similar size as current market available devices such as a fitbit. Another component that would need to be improved is the battery. We are currently using a LiPO battery that is rather large and hangs off the board. Improving this would again lower the overall size of the device and increase its robustness. Additionally, increasing the sampling rate within the AFE would allow for additional tests to be done, including the attempted testing of a heartbeat and pulse oximetry measurements. While these were not the focus of our device, adding them to a final product would make it more viable for consumers to use. Furthermore, an app or similar notification system could be developed to allow for easier access to results. Currently connecting the device to a bluetooth program and then running the device through a MATLAB script is not the most user friendly way of giving results to the user. Additionally, using a surface mounted photodetector capable of sensing infrared light would be ideal and would eliminate the additional component to the board that we had to include.

There are a few design considerations in terms of board layout that are considered here for future improvements to the project. Seeing that our SPI continued to demonstrate a noise

issue on the second AFE communication line to the microcontroller due to passing a clock signal underneath a trace, the final schematic and board reflect the changes required for the second clock signal to not become an issue any longer. The user would have to change the pin numberings in our *RTE_Device.h* file in the HydroWatch software in order to correctly initialize the digital input/output pins on the RSL10 to have the SPI interfaces where we desire them to be. Additionally, to account for the through-hole infrared photodetector that was jerry-rigged off the side of the HydroWatch board with leads soldered to the pads, the final HydroWatch board design contains a three-pin design so that this piece can be integrated into the board and not be floating off the side of the main board. In relation to the board is also the "wearability" aspect of the future of the project, which could use a fancier strap that is more ergonomic than a giant velcro strap that is as wide as the entire board itself. If the board were to be made even skinnier, the watch band options would be more comfortable on even the smallest of users' wrists, and even more people could use HydroWatch and determine their own hydration levels.

In terms of software or programming changes to the demo day version of HydroWatch, it would be very useful if the TX_START button were to be utilized with the proper interrupt functionality to start hydration measurements once the watch is in place, the entire program could be simplified a bit. With a set sampling rate, we would be able to determine the exact amount of data points to take, which would be triggered only when the button is pressed. Another change to the software and hardware of the project would be to place the TX_START button on the WAKEUP pad of the RSL10, which would allow our board to be placed into sleep mode and then correctly woken up to start the transmission of data only when required. This would save a significant amount of energy and also would preserve the quality and functionality

of our components. Another possible software change to be implemented to improve the overall project would be to implement the post processing logic on the actual internal storage and processor included in the RSL10. This would be rather straightforward, as the user's first measurement of the day on the 1200 nm light could be used as the baseline and the rest of the measurements could be compared back to that baseline easily. Finally, the newest board versions contain LEDs on output pins, and when the internal processor determines that the user is hydrated, that light on the main board could be made to light up to signal very quickly to the user that he or she is properly hydrated in comparison with his or her initial baseline hydration measurement.

9 Conclusions

The device was an overall success. We accomplished all of the objectives that were outlined in the High Level Design for this project. This included the ability to utilize a microcontroller to determine LED flash rates and received signals from our photodiodes using SPI commands. We were also able to communicate this data over bluetooth for post processing within a Matlab script, and we created a well designed GUI that allowed for easy user interaction and notification of results. We ran into some issues along the way, and there were some changes that could be made to improve our design, as we touched on within this report, particularly within the To-Market Design Changes section, but overall we approached the problem correctly and achieved what we established to be our goals. We learned a lot from this project and will certainly apply this new knowledge and the skills we gained towards future projects.

10 Appendices

Complete hardware schematics can be found at the HydroWatch Senior Design website:

http://seniordesign.ee.nd.edu/2019/Design%20Teams/hydra/index.html#Documents

Complete Software listings:

- RSL10 Bluetooth Low Energy Explorer: Found on ONsemiconductors website
- Eclipse bundle: Found on ONsemiconductors website under RSL10 development board
    - We suggest one also reads the accompanying user guide
- J-link debugger
- Java: Note that configuration has to match eclipse (ie x32 or x64)
- MATLAB: active license is required.
- Python 3.x: Easiest to install Anaconda

Relevant parts or component data sheets:

RSL10 dongle:

(https://www.onsemi.com/pub/Collateral/RSL10%20USB%20DONGLE%20USER%20GUIDE.PDF)

RSL10 Getting Started: https://www.mouser.com/pdfdocs/ONSemi_RSL10_Start.pdf

AFE4490 Datasheet: http://www.ti.com/lit/ds/symlink/afe4490.pdf

970 nm LED: https://www.tech-led.com/wp-content/uploads/2013/10/SMT970-232.pdf

660 nm & 450 nm LEDs: https://www.lumileds.com/uploads/415/DS105-pdf

1200 nm LED:

https://www.digikey.com/product-detail/en/marktech-optoelectronics/MTE0012-095-IR/1125-1329-ND/5872598

SMT Photodetector:

https://www.mouser.com/ProductDetail/Vishay-Semiconductors/TEMD5010X01?qs=sGAEpiMZZMtWNtIk7yMEsbNebl7bzYiWhNjZ6ipXWYQ%3d

Infrared Photodetector:

https://www.digikey.com/product-detail/en/marktech-optoelectronics/MTPD1346D-030/1125-1364-ND/5870138

TPS61201 (DC-DC Converter): http://www.ti.com/lit/ds/symlink/tps61200.pdf

Antenna Chip (2.4 GHz): https://linxtechnologies.com/wp/wp-content/uploads/ant-fff-chp-x.pdf

FTSH Connector for J-Link Programming:

http://suddendocs.samtec.com/catalog_english/ftsh_smt.pdf