Environmental and Personal Information Processor

Alberto Rosas

School of Engineering
California State University,
East Bay
Hayward, United States
alberto75543@gmail.com

Cody Hum

School of Engineering
California State Univeristy,
East Bay
Hayward, United States
codyhum03@gmail.com

Noel Anilao
School of Engineering
California State Univeristy,
East Bay
Hayward, United States
noelanilao@gmail.com

Dang Tran
School of Engineering
California State Univeristy,
East Bay
Hayward, United States
khoatran122601@gmail.com

***Abstract*— This paper presents the design, implementation, and evaluation of the Environmental and Personal Information Processor (E-P.I.P.), a modular, wrist-mounted embedded system developed to monitor environmental conditions and personal physiological metrics in real time. Designed for deployment in hazardous work environments, the system integrates multiple sensors—measuring temperature, humidity, air quality, heart rate, and GPS location—alongside user interface components such as a rotary encoder, LED array, and TFT display. The ATmega2560 microcontroller serves as the central processing unit, coordinating data acquisition, visual feedback, event logging, and system control. Robust programming workflows were developed to accommodate hardware limitations and ensure reliable operation, utilizing UART, JTAG, and SPI interfaces. Data is logged locally to a Micro SD card using a buffered SPI architecture for stability. A custom 3D-printed enclosure houses the electronics, designed for comfort, durability, and sensor exposure. The system demonstrated reliable detection of hazardous conditions, with visual and data-logged alerts triggered based on configurable thresholds. While stretch goals such as power optimization and graphical data visualization were not fully realized, the system establishes a functional prototype platform for future expansion and deployment in safety-critical applications.**

***Keywords*— embedded systems, environmental sensing, health monitoring, microcontroller, data logging, sensor integration, workplace safety**

I. INTRODUCTION

Our goal is to develop a wearable safety device designed to assist users working in harsh and hazardous environments, such as oil rigs, construction sites, landscaping, mining sites, and agricultural fields. In these settings, access to real-time information and alerts is critical for ensuring safety and enabling rapid responses to dangerous conditions.

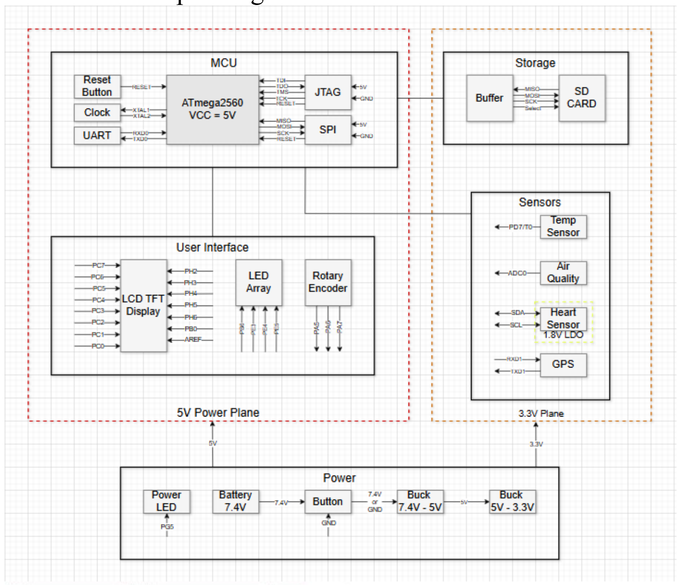
Drawing inspiration from wrist-mounted computers featured in science fiction such as those in the Fallout series as well as historical safety practices like using canaries in coal mines, we aim to blend innovation with practicality. Leveraging Arduino-compatible technology and a suite of modern sensors, our project seeks to deliver a wrist-mounted computer capable of monitoring vitals, tracking environmental conditions, and providing critical geographical data. This integration enhances both personal safety and situational awareness in demanding workplaces. Ultimately, this device pays homage to the tools of the past while offering a tangible step forward in modern occupational safety.

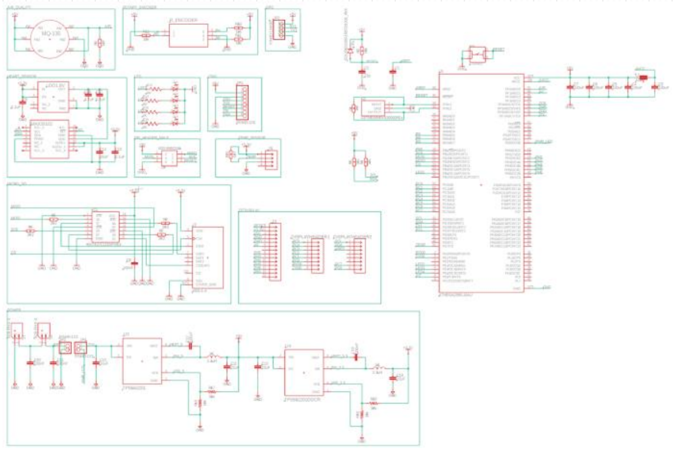
A. Hypothesis

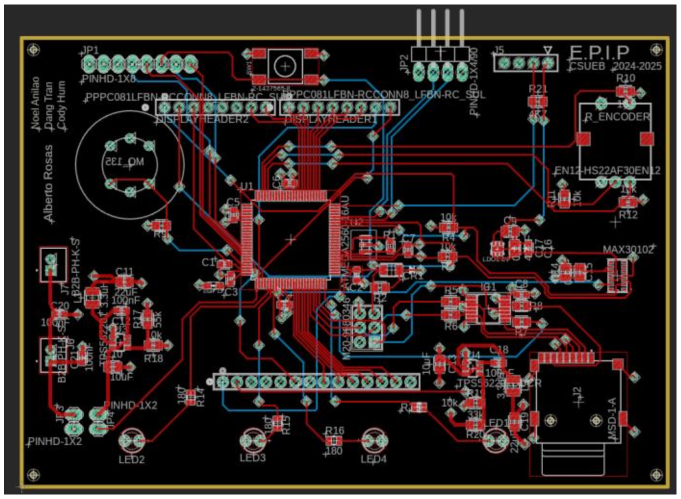
We hypothesize that the E-P.I.P. device can serve as an effective safety tool for individuals operating in hazardous or unhealthy environments. With integrated sensors measuring air quality, temperature, humidity, heart rate, and GPS location, the system will be capable of continuously monitoring both environmental and physiological conditions. We envision it notifying users in real time when thresholds indicate unsafe conditions and, in emergency scenarios, transmitting location data to enable rapid response. The overarching goal is to improve workplace safety and enable timely intervention when health risks arise.

B. Experiment

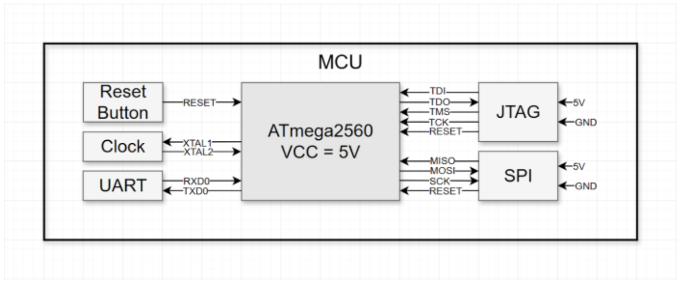
This project focused on designing and implementing a modular embedded system that integrates various sensors and I/O components to collect and display real-time environmental and health data. A primary objective was to assess the performance and reliability of each subsystem, such as heart rate sensing, user interface control, and visual display through iterative prototyping and testing. Our experimental methodology included schematic and PCB design, firmware development, and system integration using the ATmega2560 microcontroller. Each component underwent individual testing for power stability, signal integrity, and communication reliability, followed by full-system tests to evaluate overall performance. Observations were recorded to guide future optimization and ensure that the system performs effectively under realistic operating conditions.







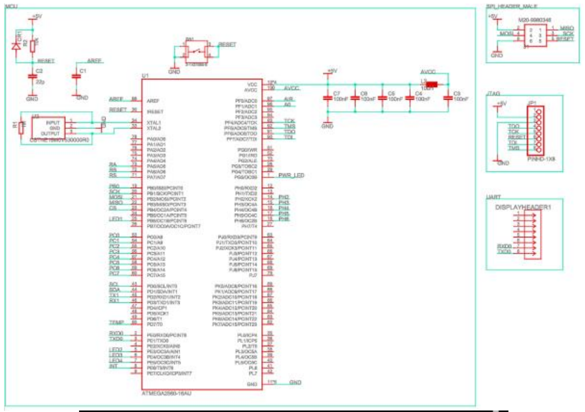
II. Microcontroller Unit

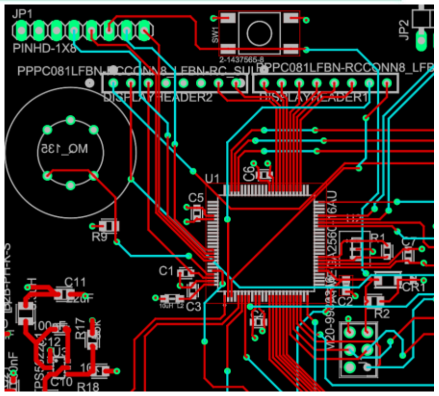


A. MCU: Overview

The ATmega2560-16AU microcontroller served as the central control hub of the E-P.I.P. system. As shown in the MCU block diagram, it coordinated communication between all sensor subsystems, user interface elements, and storage components. Chosen for its large number of I/O pins (86), 256 KB of flash memory, and support for multiple serial interfaces (SPI, UART, I²C, JTAG), it offered the flexibility needed to manage concurrent sensor activity and data logging.

To support stable operation, the design included a 16 MHz external crystal oscillator, a reset circuit with diode protection, and decoupling capacitors across each power and ground pair. The MCU operated on a regulated 5V rail, powering both logic and connected peripherals.



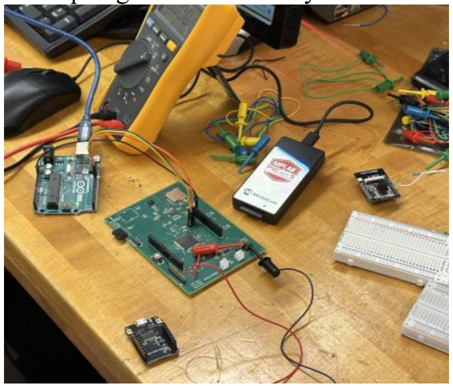


B. MCU: Schematic & Layout

The schematic incorporated key components for reliable operation. All power pins were paired with local decoupling capacitors to minimize noise. The external oscillator circuit connected to XTAL1 and XTAL2 via a series resistor to bias the inverter and improve startup behavior. A momentary tactile switch allowed manual resets by grounding the RESET line, which also included a 10kΩ pull-up and diode for ESD protection.

For programming and debugging, headers were provided for SPI, JTAG, and 2 UART. The primary UART header was shared with the TFT display pins, which created a challenge during software updates that required temporary disconnection of the display or the use of jumper wires.

On the PCB, the ATmega2560 was centrally placed to optimize trace routing. The JTAG interface was routed to the top-right corner to avoid collision with the display, while the SPI header was positioned below the display because it wouldn't be used after initial programming of the MCU as well as near the SD card interface on the bottom-right that also utilized SPI to allow easier routing. The reset button was placed on the board's top edge for accessibility.



C. MCU: Milestones

The MCU subsystem was developed through a series of critical milestones. The first step was verifying stable power across the 5V and 3.3V domains using a multimeter and oscilloscope. While the power rail tested correctly, the external crystal showed no oscillation, and SPI-based programming failed.

This led to the second milestone: successful communication via the JTAG interface. Using a PICkit 5 programmer and MPLAB X IDE, a simple LED blink program was loaded via JTAG, confirming connectivity. A .hex file containing the Arduino Mega 2560 bootloader was then flashed using MPLAB IPE. Configuration bits were adjusted in MPLAB X IDE via JTAG to match Arduino bootloader specifications, including enabling the external clock and disabling JTAG after flashing.

The final milestone involved transitioning to UART-based programming. A standard Arduino UNO was converted into a USB-to-Serial adapter by flashing the ArduinoISP sketch and removing its onboard ATmega328P microcontroller. This modified board was wired to the custom E-P.I.P. PCB (TX, RX, RESET, 5V, GND) and used to upload compiled sketches directly from the Arduino IDE. At this point, the software workflow was complete, and development could proceed normally using UART uploads.

D. MCU: Software

Software development began by preparing the ATmega2560 for Arduino compatibility. A simple "Hello World" sketch was compiled in the Arduino IDE with the board set to "Arduino Mega 2560." The compiled .hex file, which included the Arduino bootloader, was flashed to the microcontroller via JTAG using MPLAB® X IPE and a PICkit™ 5 programmer.
To ensure full compatibility with the Arduino environment, fuse and configuration bits were manually set via MPLAB X IDE:

- LOW (0xFF): EXTXOSC (8 MHz ~ 16KCK/65ms), CKOUT disabled, CKDIV8 disabled
- HIGH (0xD8): SPIEN enabled, BOOTRST enabled, JTAG disabled
- EXTENDED (0xFD): BODLEVEL set to 2.7V (brown-out detection)
- Lock Bits (0xFF): No memory locking enabled (LB, BLB0, BLB1 all set to "no lock")

After the bootloader was installed and verified, a modified Arduino UNO was used as a USB-to-Serial bridge. The onboard ATmega328P was removed, and the ArduinoISP sketch was uploaded prior to removal. The board was connected to the custom E-P.I.P. PCB using TX, RX, RESET, 5V, and GND lines.

Because the custom PCB did not use the same physical pin layout as the standard Arduino Mega 2560, an additional step was required: mapping and verifying all active pins used in the design. This was done by comparing the Arduino Mega Pin schematic with the E-P.I.P. schematic. Knowing the pin names used in the Arduino IDE that corresponded to the pins on the E-P.I.P. was essential for ensuring that the Arduino IDE correctly interfaced with the hardware. Once completed, the UART workflow supported direct sketch uploads through the IDE, enabling standard development and debugging from that point forward.

E. MCU: Summary

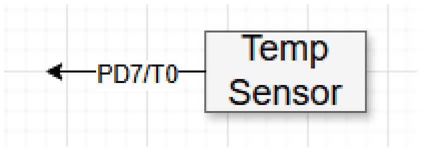
The ATmega2560 microcontroller successfully served as the central processing unit for the E-P.I.P. system. After initial issues with crystal oscillation and SPI programming, the team established a reliable development workflow using JTAG for bootloader installation and UART for ongoing sketch uploads. Including multiple programming interfaces (SPI, UART, JTAG) proved critical during debugging and recovery.

Custom pin mapping was necessary due to deviations from the standard Arduino Mega layout. This additional step allowed full compatibility with the Arduino IDE and ensured that all connected subsystems sensors, display, storage, and user input communicated reliably with the MCU.

Although advanced features like low-power sleep modes and optimized interrupt-driven design were not implemented due to layout and timing constraints, the subsystem met all core functional goals. It remained responsive, stable, and capable of managing real-time sensor data, alerts, and user interaction.

Future work could improve usability and efficiency by integrating an onboard USB-to-Serial interface, refining SPI signal routing, and enabling sleep-mode functionality for power conservation. Exploring non-Arduino toolchains may also yield tighter memory performance and better code control for production-level builds.

III. Temp & Humidity



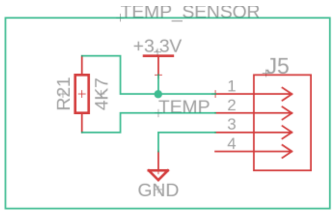
A. Temperature: Overview

The DHT22 digital sensor was selected to monitor ambient temperature and humidity within the E-P.I.P. system. As shown in the system-level block diagram, it is one of several environmental sensors interfaced directly with the microcontroller unit (MCU). Operating at 3.3V, the DHT22 transmits calibrated temperature and humidity data via a single-wire digital protocol.

B. Temperature: Schematic & Layout

The sensor connects to the MCU via three lines: VCC (3.3V), GND, and a data line routed to a GPIO pin, with a 10kΩ pull-up resistor to ensure stable communication.

A 90-degree female header was positioned at the board's edge to allow quick sensor installation or replacement without soldering, minimizing disruption to other components.



C. Temperature: Milestones

Initial tests with Arduino demo code confirmed the sensor was functional. However, persistent negative values in later reads indicated a communication issue. The problem was traced to improper read timing. By enforcing a 3-second read interval exceeding the sensor's >2 second requirement and implementing a non-blocking loop, stable data was restored. At this point, the sensor was fully integrated into the system with real-time flag handling and LED alerts.

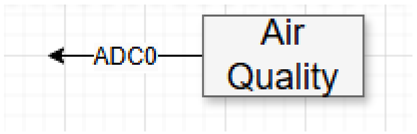
D. Temperature: Software

Sensor communication was handled using the Arduino DHT library over a single-wire digital protocol. Data was collected every 3 seconds using a millis()-based non-blocking loop. To reduce noise, a moving average of the last five readings was calculated. Values were classified into four states (cold, cool, warm, hot). If a threshold was crossed, a flag was set and used to trigger alerts and log entries.

E. Temperature: Summary

The DHT22 provided accurate, stable readings after minimal software data handling. Future improvements could include faster sensors, redundancy for validation, and calibration for specific environments. Tuning thresholds and response logic may further improve performance.

IV. Air Quality

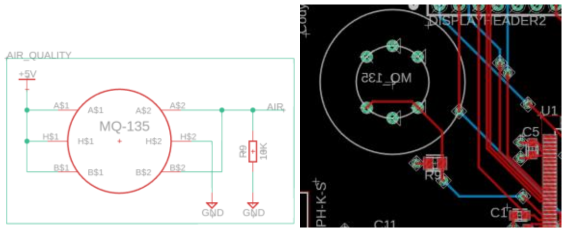


A. Air Quality: Overview

The MQ135 gas sensor was used to monitor environmental air quality in the E-P.I.P. system. It detects pollutants such as CO_2, VOCs, and smoke, outputting an analog voltage relative to gas concentration. This signal is interpreted by the MCU, which sets flags when air quality thresholds are crossed.

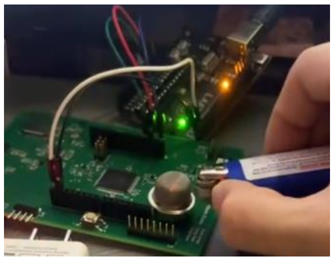
B. Air Quality: Schematic & Layout

The MQ135 connects via GND, 5V, and an analog signal pin on the MCU. A female header allows for easy sensor insertion and replacement. It was placed on the back of the PCB to avoid obstruction and sample air away from the user interface.



C. Air Quality: Milestones

Initial tests showed oversized sensor pins, which were shaved down for PCB compatibility. Once fitted, the sensor operated correctly. Thresholds were mapped to PPM values (e.g., >200 PPM = Hazardous), and flag logic was implemented in the MCU.



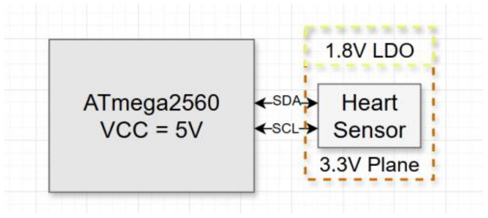
D. Air Quality: Software

Data was read using a reference Arduino library. A moving average of the last five readings smoothed fluctuations. Based on thresholds, flags triggered alerts and system responses.

E. Air Quality: Summary

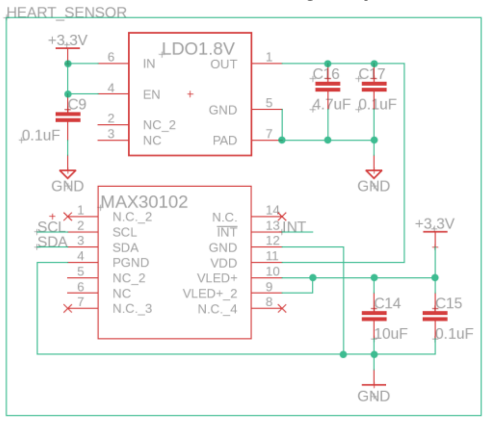
The MQ135 successfully detected hazardous conditions with stable readings after smoothing. Future improvements could include calibration, adaptive thresholds, and advanced filtering or ML techniques for improved accuracy.

V. Heart



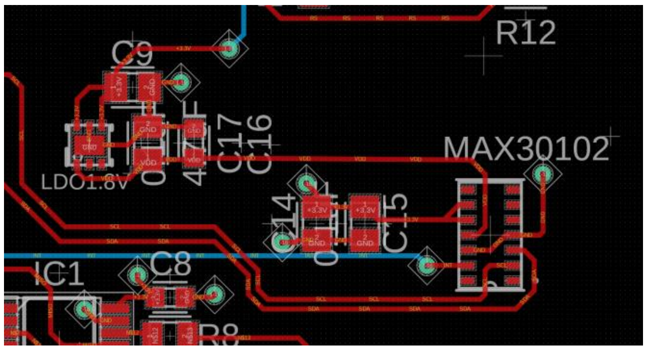
A. Vitals Overview

The figure above illustrates the top-level block diagram of the vitals subsystem and its communication with the MCU. This subsystem employs the MAX30102 pulse oximeter and heart rate sensor, which includes internal LEDs, photodetectors, and low-noise electronics with ambient light rejection



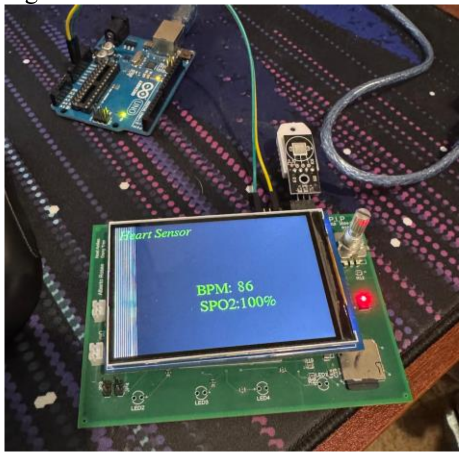
B. Schematic & Layout

The schematic incorporates all necessary components to support stable sensor operation. Decoupling capacitors are placed on both power lines (1.8V and 3.3V) to minimize voltage ripple and noise. The I²C lines are carefully routed to maintain signal integrity and reduce electromagnetic interference. On the PCB, the MAX30102 sensor is positioned on the center-right area to avoid interference with the TFT display, which is mounted above the main board via headers. The decoupling capacitors are placed as close as possible to the sensor's power pins to ensure effective noise suppression.



C. Milestones

Initial hardware testing focused on verifying correct voltage levels for both the 1.8V and 3.3V domains using a multimeter. During early firmware development, the sensor's LEDs remained continuously active, prompting concerns about potential routing issues. However, further investigation revealed that LED behavior is controlled entirely via software, eliminating the need for any hardware changes. Successful I²C communication between the MAX30102 and MCU confirmed the correct wiring and stable power delivery. Sensor responsiveness and data accuracy met expectations during early testing.



D. Software

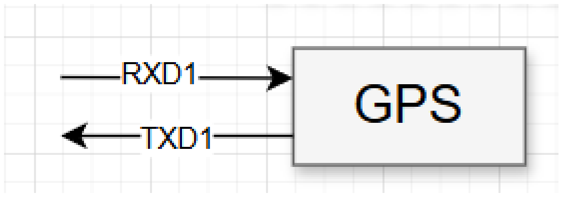
Software development was conducted using the Arduino IDE and the SparkFun MAX301x Particle Sensor Library. Initialization involves importing library headers, defining variables, and setting up the sensor object. Within the setup() function, the system checks for I²C connectivity before applying default configuration settings.

When the Vitals menu is accessed on the TFT display, the sensor is activated, and the user is prompted to place their finger on the optical interface. Infrared data is collected into an array and processed using built-in algorithms to calculate heart rate (BPM) and blood oxygen saturation (SpO_2). This information is displayed in real time, and data acquisition continues as long as finger contact is detected.

E. Vitals Summary

The Vitals subsystem met all core functional requirements. The MAX30102 sensor operated reliably, providing accurate and responsive heart rate and SpO_2 readings. Leveraging existing libraries greatly simplified software development and integration. System integration was successful, with no major hardware or software issues impacting performance. The subsystem delivered essential health monitoring functionality critical to the E-P.I.P. system's purpose.

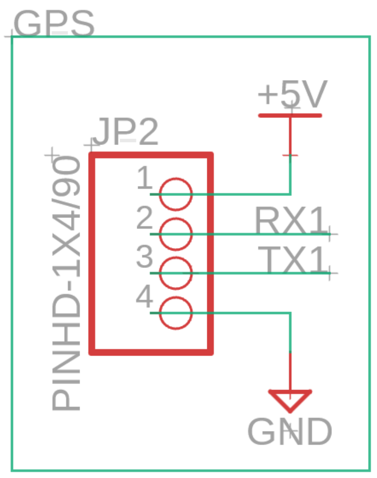
VI. GPS



A. GPS: Overview

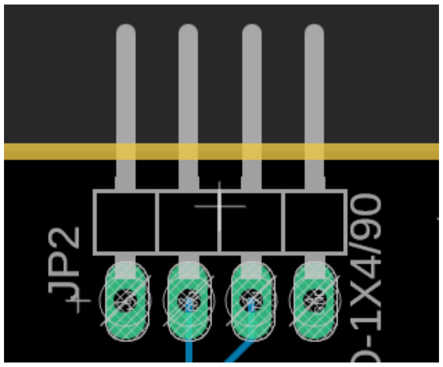
The GPS subsystem utilizes the NEO-6M module with external signal antenna. Its main purpose is to locate the user's current position based on longitude, latitude, and altitude via satellite positioning. Serial communication with the GPS module is established using TX (transmitter) and RX (receiver) pins. RXD1 and TXD1, which are serial ports digital 0 and 1 from the E-P.I.P system, connect to RX and TX respectively.

This configuration is used to read Serial1 on the serial monitor.



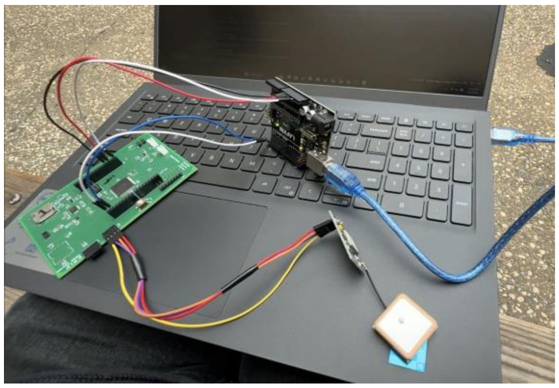
B. GPS: Schematic

The schematic shown is the pinout for the GPS subsystem on the E-P.I.P. Pin 1 connects to 5V VDD. Pin 2 is the RX1 and connects to the RX. Pin 3 is the TX1 and connects to TX. Pin 4 connects to GND.



C. GPS: Layout

The layout shown above is a right angle 1x4 male pin header that connects to the NEO-6M GPS module via female-to-female pin connection.



D. GPS: Milestones

The GPS module successfully integrated with our E-P.I.P. system, confirmed by the blue blinking LED. After flashing the boot drive and wiring the module to the appropriate pins, testing began with no major hardware issues.

Reaching the second milestone involved a lot of troubleshooting, the main issue was the module's long satellite fix time after a cold start (5-10 minutes). However, once a signal was acquired, it provided consistent, accurate geolocation data via satellite positioning. We set a three-second read interval to balance power use and simplify coordinate parsing. The NEO-6M module performed reliably well overall.

At our third milestone, we found that maintaining continuous uptime improves GPS read speed performance. This includes reduced delays and immediate GPS readings when the user switches to GPS display.

The main limitation is the antenna's sensitivity. Outdoor, unobstructed environments are necessary for stable satellite connections. Without satellite visibility, only the last known coordinates are available.

E. GPS: Software

The GPS subsystem communicates with the MCU via UART and is programmed using Arduino IDE with TinyGPSPlus library. TinyGPSPlus simplifies parsing NMEA sentences into readable data such as latitude, longitude, altitude, and time.

Currently, the GPS module is set to the location, altitude, time, and satellite count. It works in tandem with *Vitals* and *Air Quality* subsystems as part of a black box feature, logging geolocation data to a MicroSD when critical vitals or hazardous air quality are detected.

F. GPS: Summary

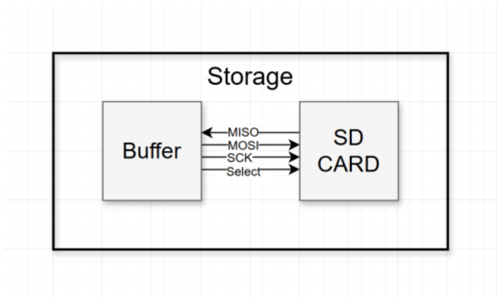
The NEO-6M GPS module is a key component of the E-P.I.P. system, providing accurate and reliable satellite-based location tracking, ideal for outdoor use. With an external antenna, it delivers precise latitude, longitude, and altitude readings.

Communication between the NEO-6M and the microcontroller unit (MCU) is handled via UART serial interface and uses TinyGPSPlus library to parse NMEA data into readable coordinates for real-time tracking.

Beyond navigation, the module works with the heart rate and air quality sensors to enhance safety. In the case of emergencies, it logs the user's last known location to a MicroSD card. This is to ensure critical data is preserved.

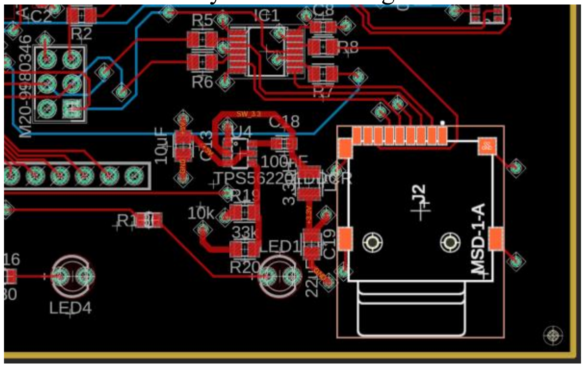
Overall, the module offers reliable, real-time tracking essential for both navigation and emergency response.

VII. Storage



A. Storage: Overview

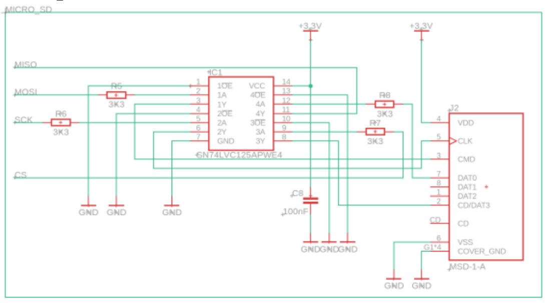
The storage subsystem provides reliable, event-driven data logging functionality for the E-P.I.P. system. As illustrated in the Storage block diagram, it consists of a Micro SD card slot and a quad bus buffer (SN74LVC125A). The buffer plays a critical role in protecting the SPI bus from contention by isolating the Micro SD card when inactive and ensuring that only one device drives the bus at a time. This setup ensures signal integrity and safe communication between the card and the microcontroller. Data is transmitted over the SPI interface, using the standard MISO, MOSI, SCK, and CS lines. The chip select line is controlled by a GPIO pin on the MCU to manage when the card is actively communicating.



B. Storage: Schematic & Layout

The schematic identifies the SD card reader as J2 and the quad buffer as IC1. The SD card slot is powered by a regulated 3.3V source, and each SPI line is routed through the buffer to enhance signal integrity. The MOSI line connects to the SD card's CMD pin, the MISO line carries data from the SD card to the MCU, the SCK line connects to the DAT3 pin, and the CS (PB4) line activates the card via a buffered control line. To prevent overshoot during signal transitions, 3.3 kΩ series resistors are placed on all SPI lines. The buffer is also decoupled with bypass capacitors for stable operation.

The physical layout places the SD card reader at the bottom-right corner of the PCB for easy access and proximity to the 3.3V power supply. The buffer is positioned after the shared SPI header, allowing the Micro SD card to share SPI lines with other peripherals while maintaining signal integrity. This configuration ensures clean communication between the MCU and SD card, with the buffer acting as an intermediary to isolate and protect the SD card from bus contention.



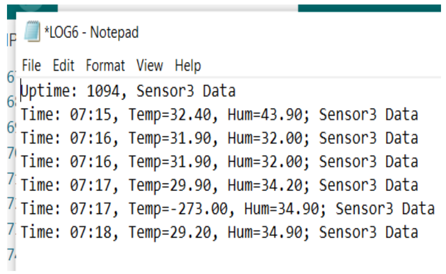
C. Storage: Milestones

Initial testing began with verifying stable 3.3V power delivery to both the SD card and buffer using a multimeter. With power confirmed, functionality was tested using the Arduino SD library example sketch, which successfully detected card insertion and removal.

Once hardware-level communication was verified, the MCU was programmed to log uptime and events to the SD card. Each log entry is written to a uniquely named file (e.g., Log, Log1, Log2), allowing for session-based tracking and post-event review.

D. Storage: Software

The system uses the Arduino SD library to interface with the Micro SD card. On boot, the software checks for card presence. If the card is missing, a global flag (sdFlag) is raised to signal unavailability. If a card is detected, the program automatically determines the next available log filename for the session. During operation, the software monitors all sensor condition flags using the readFlag() function. When any sensor changes state from safe to unsafe, the logSensorData() function is triggered. This function logs relevant sensor data to the SD card, attaching a timestamp using GPS data if available, or system uptime if not. Logging is designed to be event-driven rather than periodic, ensuring efficient memory use and reduced wear on the SD card.

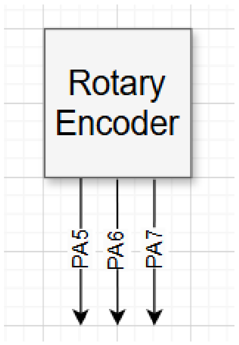


```
Uptime: 1094, Sensor3 Data
Time: 07:15, Temp=32.40, Hum=43.90; Sensor3 Data
Time: 07:16, Temp=31.90, Hum=32.00; Sensor3 Data
Time: 07:16, Temp=31.90, Hum=32.00; Sensor3 Data
Time: 07:17, Temp=29.90, Hum=34.20; Sensor3 Data
Time: 07:17, Temp=-273.00, Hum=34.90; Sensor3 Data
Time: 07:18, Temp=29.20, Hum=34.90; Sensor3 Data
```

E. Storage: Summary

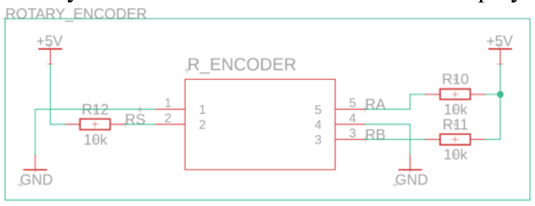
The storage subsystem met all its primary design goals. It reliably handled SPI communication with the SD card, maintained stable power through the 3.3V rail, and successfully recorded time-stamped sensor data when triggered by alert conditions. The buffer greatly improved signal isolation, preventing bus contention and reducing noise during communication. While core logging functionality was implemented, future improvements could include integrating onboard flash or EEPROM for faster, USB-accessible logs, implementing periodic logging for long-term trend tracking, and enabling on-screen log viewing via the TFT display. Structured data formatting could also be introduced to make logs easier to parse during post-analysis.

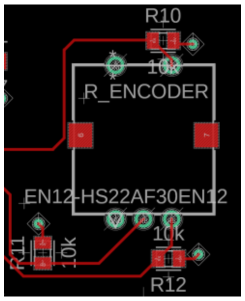
VIII. Rotary Encoder



A. Rotary Encoder: Overview

The EN12 rotary encoder serves as the primary input method for navigating sensor data on the E-P.I.P. system. Connected directly to the microcontroller via GPIO pins, it provides both rotational and button-press input. Operating between 3.3V and 5V, the encoder generates digital pulses interpreted by the MCU to switch menus on the display.





B. Rotary Encoder: Schematic & Layout

The encoder interfaces with the MCU through three GPIO pins (PA5-7): CLK, DT, and SW, each paired with pull-up resistors to ensure clean logic levels. It is positioned on the top right of the PCB to avoid layout conflicts, with nearby resistors minimizing signal noise.

A mechanical issue arose during assembly; the encoder's stability pins, intended for through-hole mounting, did not match the PCB's surface-mount layout. To resolve this, the pins were trimmed, allowing flush mounting while retaining electrical functionality.

C. Rotary Encoder: Milestones

Initial power testing confirmed proper 5V supply, but rotation produced incorrect output. Debugging revealed that the clock and ground pins were swapped in the schematic. This was corrected in software through GPIO remapping, restoring bidirectional control and button function.

Further testing revealed intermittent glitches skipped or stuck values due to mechanical bounce or signal instability. These were reduced through refined polling logic, achieving around 90% input reliability in test conditions.

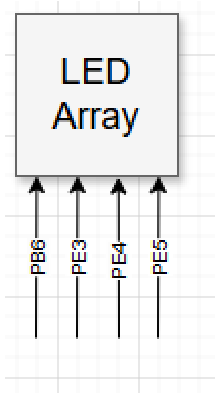
D. Rotary Encoder: Software

The rotary encoder is managed using a simple state machine within the main control loop. It connects to Arduino Mega pins 29 (rotation signal A), 28 (rotation signal B), and 27 (push-button switch). The system begins in an idle state, with counter = -1. Pressing the button enters rotation mode, allowing the user to cycle through values 0 to 3 by turning the encoder. Pressing the button again exits to idle. Inputs are polled continuously, and a 200 ms software debounce delay is applied after each button press to prevent false triggers. Direction is determined by tracking changes in the ENCODER_CLK signal using the lastStateCLK variable, ensuring stable and responsive input.

E. Rotary Encoder: Summary

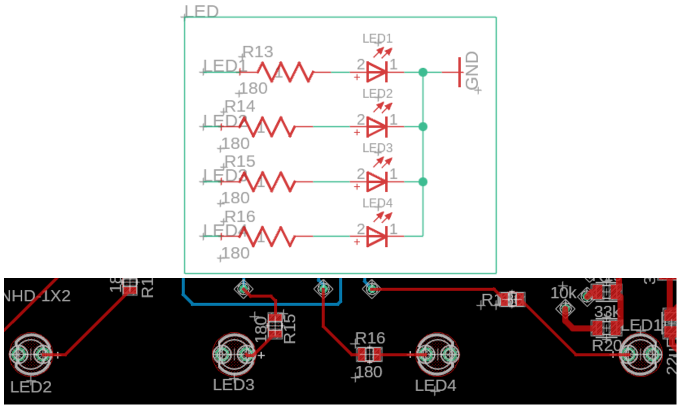
The rotary encoder met its functional goal of enabling directional input and menu control. A schematic error was resolved through software pin reassignment and further tuning improved input stability. However, occasional glitches persisted due to mechanical and electrical limitations. To improve future performance, PCB routing should be corrected, through-holes added for mechanical stability, and filtering techniques considered. Alternatives such as digital or optical encoders may offer increased reliability and resolution.

IX. LED Array



A. LED Array: Overview

The LED alert subsystem provides real-time visual notifications when sensor readings exceed safe thresholds. As shown in the LED Array block diagram, four red LEDs are connected to the microcontroller unit (MCU) via GPIO pins PB6 and PE3–PE5. Each LED corresponds to a critical sensor: temperature, air quality, GPS, and heart rate.
When the MCU detects a flagged condition, it drives the appropriate LED high, providing an immediate visual cue to the user. The rotary encoder can then be used to navigate to the relevant screen for detailed sensor information. This visual alert system supports fast, intuitive interaction and improves user awareness in hazardous conditions.



B. LED Array: Schematic & Layout

Each red LED is connected to the microcontroller via a dedicated GPIO pin: PB6, PE3-5. A 180 Ω resistor is placed in series with each LED to limit current, based on the 5V output of the MCU and the typical forward voltage of red LEDs. This value ensures safe operation below the LED's rated current, helping to prevent damage and extend lifespan.
The LEDs are arranged in a single labeled row on the PCB to clearly indicate their associated sensors (temperature, air quality, GPS, heart rate). Through-hole footprints were used for both the LEDs and resistors to allow straightforward assembly and secure mechanical connections. No pull-down resistors are required, as each LED is controlled directly through digital output pins from the MCU.

D. LED Array: Milestones

Initial testing confirmed correct GPIO output to each LED. The pins used were Digital 10 (PB6), 5 (PE3), 2 (PE4), and 3 (PE5), corresponding to heart rate, temperature, air quality, and GPS alerts, respectively. When paired with sensor flag logic in the software, the LEDs reliably responded to threshold events, illuminating in real time during system operation. The visual alert system proved useful during both functional testing and simulation of unsafe conditions.

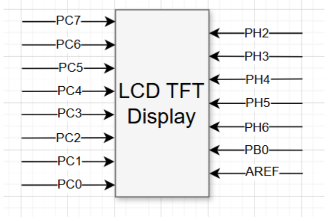
F. LED Array: Software

LED behavior is controlled via the updateLED() function, which checks each sensor flag (e.g., tempFlag, airFlag) and sets the corresponding LED state using digital output. The system architecture supports modular control, allowing each sensor routine to independently trigger LED alerts. A HIGH signal turns on the LED, indicating the presence of a warning condition. The user can then navigate to the relevant sensor screen using the rotary encoder interface.

E. LED Array: Summary

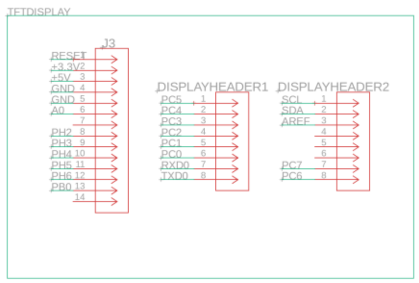
The LED subsystem met all functional goals, providing reliable, low-latency visual indicators for environmental and physiological hazards. Integration with the MCU and software flag logic ensured seamless responsiveness during system operation. Future enhancements include enabling PWM for flashing alerts and transitioning to RGB LEDs for multicolor status indication (e.g., green = safe, yellow = caution, red = danger). These upgrades would enhance visibility and improve alert granularity in complex scenarios.

X. Display



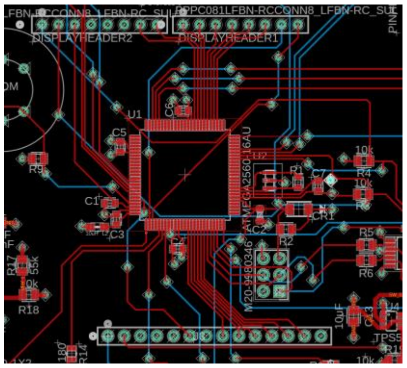
A. Display Overview

The image above is the top-level block diagram of the display subsystem and its communication protocols with the microcontroller unit (MCU). The display module is the QD3502 TFT LCD, which consists of a TFT panel, a driver circuit, and a backlight unit. The panel features a 3.5-inch diagonal size with a resolution of 320×480 pixels and operates at a 3.3V supply. Communication between the MCU and the display is accomplished via a parallel interface, utilizing GPIO pins for data transfer and control signals.



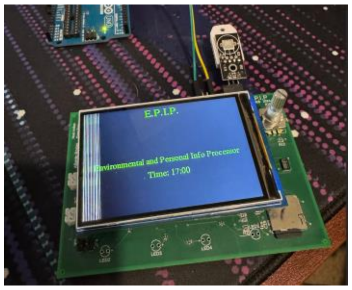
B. Schematic & Layout

The schematic ensures reliable operation of the display module by routing the necessary GPIO connections. Pin headers are used for direct communication with the MCU and for the required power connections to the module. The layout places the QD3502 TFT display module on the PCB, connected through pin headers positioned above and below the MCU. This placement avoids potential clearance issues with other system components, such as the JTAG and SPI communication ports. The headers are arranged to match the display module's pinout, though several pins remain unused in the E-P.I.P. design as they are not necessary for the system's functionality.



C. Milestones

Initial hardware testing confirmed that the display module received stable 3.3V power. Despite the display's capability to support 5V input on one pin, the system was designed to operate entirely on 3.3V for power efficiency. Early attempts to assign GPIO pins for the display interface encountered issues due to incorrect pin mapping in the default Arduino libraries. Manual pin remapping in the open-source Arduino TFT library also proved ineffective. A comprehensive solution required modifications to both the TFT library and the Arduino Mega core libraries, as the necessary pins were not natively defined. After these adjustments were made, the display successfully rendered graphics, enabling further development of the graphical user interface (GUI).



D. Software

The software for the display subsystem focuses on presenting sensor data in a user-friendly graphical interface. The GUI includes a main menu displaying a welcome message, user information, and system uptime. Additional dedicated menus are available for temperature, air quality, vitals, and GPS sensor data, with contextual messages provided based on the values reported by each sensor. The display is driven using the MCUFriend library, which provides high-level functions for rendering text and graphics. The Adafruit_GFX library is also utilized to customize the display with custom fonts and drawing utilities for enhanced visual representation.

E. Display: Summary

The display subsystem met all core functional requirements, providing a clear and efficient means to display real-time sensor data to the user. The system's design and implementation were facilitated by the use of open-source libraries, which streamlined development and mitigated the need for low-level driver coding. Despite initial challenges with pin mapping and library configuration, the display was successfully integrated into the system, enabling the development of a robust graphical user interface for user interaction. Future improvements could include additional menu options and more dynamic visual elements.

XI. Software

Recall flashing the ATmega2560 with the bootloader .hex file and setting fuse bits via JTAG, to make the microcontroller compatible with the Arduino IDE. Software developed is uploaded through USB to Serial, in this case a modified Arduino UNO wired as a USB-to-serial interface. This enables efficient development and debugging with standard Arduino tools while preserving compatibility with the custom E-P.I.P. board.

The software, written in C++ using the Arduino IDE, follows a modular structure. Each subsystem display, sensors, rotary encoder, LED array, SD card is implemented using external libraries and custom functions. Development involved testing each module individually, followed by system-level integration. Serial output is used for real-time debugging.

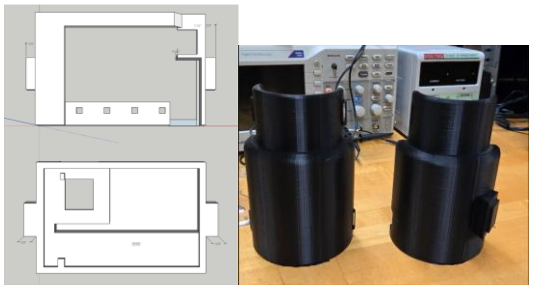
Global variables store sensor data (temperature, humidity, air quality, GPS, heart rate, SpO_2) and alert flags. Pin definitions and external libraries are declared at the top. Libraries include

SD.h, Adafruit_GFX.h, UTFTGLUE.h, DHT22.h, MQSensor.h, TinyGPS++.h, and MAX30105.h.
The program operates in two phases: setup() and loop().
In setup(), all peripherals are initialized. Serial ports are started, rotary encoder inputs configured, the LCD display is initialized, and sensors are calibrated. The SD card is prepared and the log filename is determined. Initial sensor readings are taken to establish baseline values.
The loop() handles runtime tasks:

- User Input: Reads rotary encoder to switch screens.
- Sensor Sampling: Every 3 seconds, temperature, air quality, and GPS data as well as their corresponding flags are updated using non-blocking millis(), a built-in timer counting milliseconds, logic.
- Display Update: If the screen changes, it redraws; otherwise, only values are refreshed.
- Vitals Monitoring: Monitoring vitals has to be independently read from the rest of the sensors as it requires user input. The MAX30102 sensor is only powered on when the heart rate screen is active then powers down automatically when the user navigates away, conserving the last readings.
- Alerts & LEDs: Flags are used to update the LED status High or Low for True or False.
- Logging: If a flag changes, a timestamp alongside corresponding sensor data are logged to the SD card.

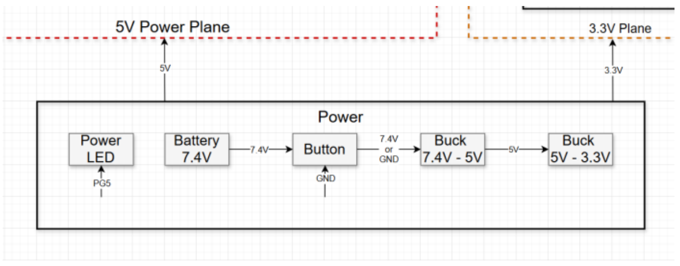
This flow provides a responsive, low-power, real-time system for environmental and personal monitoring, with extensible, library-driven code suitable for rapid development and testing.

XII. Casing



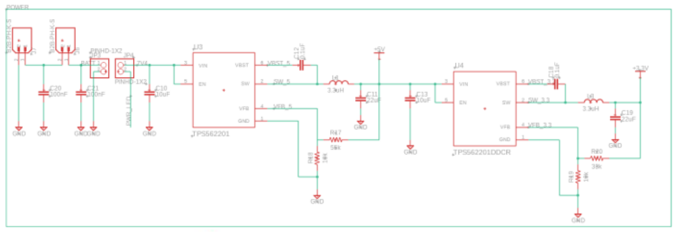
To house the E-P.I.P. system, a custom wrist-mounted enclosure was developed through three design iterations. Initial prototypes were created based on PCB measurements and refined in Fusion 360, with STL models printed on an Ender 3. Early designs suffered from misalignments due to incorrect tolerances and material shrinkage. These were corrected in later versions by adjusting dimensions, improving sensor cutouts, and incorporating a hinge-lock mechanism and LED light guides. The final enclosure, mounted on a wrist strap adapted from a Pip-Boy 2000 Mk VI design [17], was further adjusted using a soldering iron to fine-tune fit. The casing ensures portability, component protection, and ease of use in harsh environments.

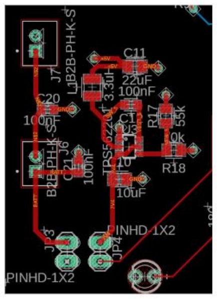
XIII. Power

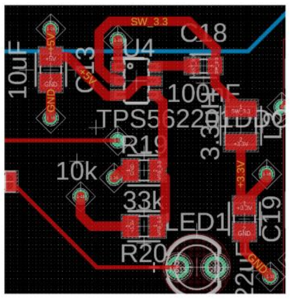


A. Power: Overview

The power subsystem supplies 5V and 3.3V voltage domains for the E-P.I.P. Two 3.7V lithium-ion batteries, in series, output 7.4V. The 7.4V is step-down to 5V using buck-switching regulators, and 5V is step-down to 3.3V using another buck-switching regulator.





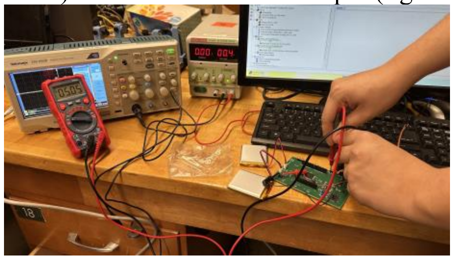


7.4V → 5.0V

5.0V → 3.3V

B. Power: Schematic & Layout

The schematic shown above has two 3.7V lithium-ion batteries in series, supplying 7.4V. The capacitors, resistors, and inductors help to filter noise and regulate the output voltages. The buck-switching regulators step down 7.4V to 5V output (left side) and then 5V to 3.3V output (right side).



C. Power: Milestones

After receiving the two 3.7V lithium-ion batteries, we connected them in series to supply 7.4V and confirmed that the left most buck-switching regulator shown above successfully stepped it down to 5V.

We also noticed a minor residual voltage reading after shutting off the power, gradually dissipating to 0.001V. We have proper ground and power planes in place, so we suspect that this is a generally normal phenomenon for voltage to slowly drop to a very small value. Probable causes are capacitor discharges and bleed-down from resistors, essentially energy dissipation when no load is present.

D. Power: Summary

The power subsystem uses two 3.7V batteries in series to supply 7.4V, regulated by buck-switching regulators into 5V and 3.3V domains. These outputs power the various other components in the E-P.I.P. This ensures efficient and stable operation.

XIV. Conclusion

What began as a daunting concept has transformed into a compelling and rewarding engineering endeavor. The Environmental and Personal Information Processor (E-P.I.P.) marks a major step forward in our educational career, by developing a rugged, wearable safety device tailored for hazardous environments.

At its core lies the ATmega2560 microcontroller, selected for its extensive I/O, dependable processing power, and its ability of managing concurrent sensor data streams and peripheral communication. The system integrates critical modules including the MQ135 (air quality), DHT22 (temperature and humidity), MAX30102 (heart rate and oxygen), and NEO-6M (GPS), all operating in real time to ensure the user's awareness and safety.

A rotary encoder enables intuitive user interaction, with LED indicators and a 3.5" LCD display to provide immediate visual alerts. Data is logged to a Micro SD card for both real-time reference and post-event analysis. The device is powered by dual Li-ion batteries supported by efficient voltage regulation circuits, ensuring reliable operation in off-grid environments.

Our rigorous performance testing confirms that the sensors can operate concurrently without overloading the MCU. With a total power draw of just 4.452W, the system remains energy-efficient and portable. A compact, thermally optimized PCB layout enables the device to maintain a wearable form factor.

With hardware assembly completed and software near ideal parameters, we are confident in this device's potential to enhance safety and support emergency response in challenging environments. We look forward to seeing the E-P.I.P. become a vital tool for professionals facing extreme conditions.

WORKS CITED

[1] D. Prentice, MCUFRIEND_kbv - TFT display library for Arduino, GitHub. [Accessed: Apr. 10, 2025].

[2] Adafruit Industries, Adafruit GFX Library - Core graphics library for displays, GitHub. [Accessed: Apr. 10, 2025].

[3] dvarrel, DHT22 Arduino Library, GitHub. [Accessed: Apr. 10, 2025].

[4] Daguer, MQSensor - MQ-series gas sensor interface library, GitHub. [Accessed: Apr. 10, 2025].

[5] M. Hart, TinyGPSPlus - A new Arduino library for parsing NMEA data, GitHub. [Accessed: Apr. 10, 2025].

[6] SparkFun Electronics, SparkFun MAX3010x Pulse and Proximity Sensor Library, GitHub. [Accessed: Apr. 10, 2025].

[7] Arduino, Wire Library - I2C Communication, Arduino Reference. [Accessed: Apr. 10, 2025].

[8] Arduino, SD Library - Reading and writing files on SD cards, Arduino Reference. [Accessed: Apr. 10, 2025].

[9] V. R. K. C., ATMEGA2560 Standalone Using Arduino UNO, Instructables. [Online]. Available: https://www.instructables.com/ATMEGA2560-Standalone-Using-Arduino-UNO/. [Accessed: Apr. 10, 2025].

[10] Microchip Technology, ATmega2560-16AU 8-bit Microcontroller Datasheet, [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf. [Accessed: Apr. 10, 2025].

[11] Aosong, DHT22 Temperature and Humidity Sensor Datasheet, [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf. [Accessed: Apr. 10, 2025].

[12] u-blox, NEO-6M GPS Module Datasheet, [Online]. Available: https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf. [Accessed: Apr. 10, 2025].

[13] Analog Devices, MAX30102 Pulse Oximeter and Heart-Rate Sensor Datasheet, [Online]. Available: https://cdn.sparkfun.com/assets/8/3/c/3/2/MAX30102_Datasheet.pdf. [Accessed: Apr. 10, 2025].

[14] Winsen Electronics, MQ-135 Air Quality Sensor Datasheet, [Online]. Available: https://soldered.com/productdata/2022/03/Soldered_MQ-135_datasheet.pdf. [Accessed: Apr. 10, 2025].

[15] Texas Instruments, TPS562201 Buck Converter Datasheet, [Online]. Available: https://www.ti.com/lit/ds/symlink/tps562201.pdf. [Accessed: Apr. 10, 2025].

[16] Adafruit Industries, *3D Enclosure for 3.5" TFT Arduino Mega* [Online]. Available: https://www.thingiverse.com/thing:3058946. [Accessed: Apr. 10, 2025].

[17] Ryan Khoo (Ryan1705), Fallout 76 Pip-Boy 2000 Mk VI, MyMiniFactory. [Online]. Available: https://www.myminifactory.com/object/3d-print-fallout-76-pip-boy-2000-mk-vi-87485. [Accessed: Apr. 10, 2025].