## Project Roadmap: Fault-Tolerant Health Monitor

A 5-month plan designed to build your project and develop core embedded engineering skills.

## Foundational Knowledge: The Essentials

Before you start, make sure you're comfortable with these fundamentals. This isn't about being an expert, but having a solid footing will make the journey much smoother.

- **Proficient in C Programming**: Embedded systems run on C. Be solid with pointers, data structures (structs, arrays), bit manipulation (AND, OR, XOR, SHIFT), and memory management.
- **Basic Electronics**: Understand what voltage, current, and resistance are. Know how to read a schematic and understand datasheets for components. Familiarity with protocols like **UART**, **I2C**, and **SPI** is crucial.
- **Version Control**: Start using **Git** from day one. It's a professional habit that will save you from countless headaches. Create a repository on GitHub or GitLab for your project.

## Month 1: Mastering the Microcontroller & Toolchain

The goal this month is to get comfortable with your main tool, the STM32, and its development environment. You'll learn how to control the most fundamental hardware features.

### Mini-Project 1: "The Digital Heartbeat"

- **Objective**: Make an LED blink at a specific rate (e.g., 1 Hz), control it with a push button, and send status messages to your computer.
- **What You'll Do**:
  1. Set up your **STM32CubeIDE** development environment.
  2. Use **STM32CubeMX** to configure the microcontroller's clock and pins.
  3. Write code to configure a pin as a **GPIO** output to control an LED.
  4. Configure another pin as a **GPIO** input to read a push button.
  5. Implement a hardware **interrupt** that triggers when the button is pressed.
  6. Configure the **UART** peripheral to print messages like "Button Pressed!" or "System OK" to a serial terminal on your PC.

### Key Engineering Skills Learned:

- **MCU Toolchain**: Setting up an IDE, compiler, and debugger.
- **Bare-Metal Programming**: Directly controlling hardware registers.
- **Peripheral Configuration**: Using graphical tools (CubeMX) to generate initialization code.

- **Core Peripherals**: Mastering **GPIO** for digital I/O and **UART** for communication/debugging.
- **Interrupt Handling**: Writing efficient, event-driven code instead of polling.

---

## Month 2: Interfacing with the Real World

This month is all about data acquisition. You'll connect your health sensors to the STM32 and learn to read the raw data they produce.

### Mini-Project 2: "The Data Streamer"

- **Objective**: Interface the heart rate/$SpO_2$ sensor (MAX30102) and the temperature sensor (MAX30205). Read raw data from them and stream it over UART to your computer.
- **What You'll Do**:
    1. Study the datasheets for the MAX30102 and MAX30205 sensors.
    2. Connect the sensors to the STM32's **I2C** pins.
    3. Configure the STM32's **I2C** peripheral using CubeMX.
    4. Write C functions to communicate with the sensors—sending configuration commands and reading data registers.
    5. Continuously read the raw PPG (photoplethysmography) values from the MAX30102 and the temperature value from the MAX30205.
    6. Stream this raw data as a comma-separated list (e.g., `23451,23449,25.7\n`) over UART.

### Key Engineering Skills Learned:

- **Communication Protocols**: Hands-on implementation of **I2C**.
- **Reading Datasheets**: A critical skill for any embedded engineer. You'll learn how to find register maps, I2C addresses, and command sequences.
- **Driver Development**: Writing the low-level software that allows your MCU to "talk" to an external chip.
- **Data Handling**: Managing and formatting streams of sensor data.

---

## Month 3: Taming Complexity with an RTOS

As your system does more things, managing tasks becomes complex. A Real-Time Operating System (RTOS) helps you run multiple functions concurrently in an organized way.

### Mini-Project 3: "The Orchestra Conductor"

- **Objective**: Use **FreeRTOS** to create separate, independent tasks for acquiring data from each sensor and another task for basic data processing.
- **What You'll Do**:

1. Enable FreeRTOS in your STM32CubeMX project.
2. Refactor your code from Month 2. Create one task that reads the MAX30102 sensor every few milliseconds.
3. Create a second task that reads the MAX30205 temperature sensor every second.
4. Implement a **software queue** to safely pass the raw PPG data from the sensor task to a new "processing" task.
5. In the processing task, implement a simple **moving average filter** to smooth out the noisy PPG data.
6. The UART communication can be its own task or handled within the processing task.

**Key Engineering Skills Learned:**

- **RTOS Concepts**: Understanding **tasks**, **schedulers**, **queues**, and **mutexes**.
- **Concurrent Programming**: Writing code where multiple things happen at once without interfering with each other.
- **System Design**: Structuring your application for scalability and reliability.
- **Basic Digital Signal Processing (DSP)**: Implementing a simple filter to clean up noisy sensor data.

---

## Month 4: Building the Fault-Tolerant Engine

This is where you implement the core innovation of your project. You'll build the RNS-based arithmetic engine and use it to perform reliable calculations.

**Mini-Project 4: "The Unbreakable Calculator"**

- **Objective**: Re-implement the moving average filter (or a more advanced **FIR filter**) using your RNS engine. Develop an algorithm to calculate Heart Rate (BPM) from the filtered data.
- **What You'll Do**:
    1. Write C functions for the **RNS conversions**: `binary_to_rns()` and `rns_to_binary()` (using the Chinese Remainder Theorem).
    2. Select a moduli set, including one or two redundant moduli for error detection.
    3. Rewrite your filter's multiply-and-add logic to operate entirely in the RNS domain on the residue vectors.
    4. Implement the **error-checking** logic using the redundant moduli after each filtering step.
    5. Develop a **peak detection algorithm** to find the heartbeats in the clean PPG signal.
    6. Calculate the time between peaks to determine the beats per minute (BPM).

**Key Engineering Skills Learned:**

- **Advanced Algorithms**: Implementing complex mathematical concepts in C for an embedded target.
- **Fault Tolerance**: Designing systems that can detect and potentially correct their own computational errors.
- **DSP Algorithms**: Moving beyond basic filters to implementing heartbeat detection algorithms.
- **Optimization**: Thinking about how to make complex math run efficiently on a microcontroller.

---

## Month 5: Connecting to the World

The final step is to give your device a voice. You'll add Bluetooth connectivity to send the verified health data to a smartphone.

### Mini-Project 5: "The Wireless Broadcaster"

- **Objective**: Integrate a Bluetooth Low Energy (BLE) module. Send the calculated heart rate, $SpO_2$, and temperature values wirelessly to a mobile app.
- **What You'll Do**:
    1. Connect the HC-05 (or other BLE module) to a second **UART** port on your STM32.
    2. Write a driver to send **AT commands** to the module to configure it as a BLE peripheral.
    3. Define a custom BLE **GATT Service** with different **Characteristics** for each health parameter (e.g., one for Heart Rate, one for Temperature).
    4. Create a task in your RTOS that takes the final, verified data and updates the BLE characteristic values periodically.
    5. Use a generic BLE scanner app (like **nRF Connect for Mobile** or **LightBlue**) on your phone to connect to your device and see the data values in real-time.

**Key Engineering Skills Learned:**

- **Wireless Communication**: Understanding the fundamentals of **Bluetooth Low Energy**.
- **Protocol Stacks**: Learning about GATT services and characteristics, the building blocks of BLE data exchange.
- **System Integration**: Bringing all the pieces—sensors, RTOS, processing engine, and communication—together into one functional prototype.
- **Power Management**: (Bonus) Think about using the STM32's low-power modes to save battery when not actively measuring.