**Health Monitoring Device Using Basic Sensors**



Authors: J. Sai Shanmukhnath and Ehsan Ansari

**ABSTRACT**

**We built a small, low‑cost gadget that keeps an eye on three key health signs at once: heartbeat, body temperature, and sudden falls. The core is an Arduino Uno that reads an optical pulse sensor, a DS18B20 digital thermometer, and an MMA7361L three‑axis accelerometer. Data are sent to a computer over USB and shown in the serial port display in the project, but the data can be used to notify than display in later future as required. Early tests with three users show that the readings stay close to common consumer devices. The design is open, cheap, and easy to copy, so it fits student labs or hobby projects.**

**INTRODUCTION**

Vital signs—heart rate, body temperature, and motion—give quick clues about a person’s health. Smart watches and fitness bands track these signs, but many are pricey and closed‑source, making it hard for students to learn from them. Our aim was to design a monitor using readily available sensors that anyone with basic soldering and coding skills could build for under US $100. This project also helps to understand the fundamental principles behind multiple sensors and the ability to demonstrate interfacing a microcontroller with multiple sensors. We kept the circuit simple and chose well‑documented sensors so new learners can focus on understanding the principles, not on fighting hidden firmware.

The project set out to build a low-cost health monitor that could accurately track heart rate, skin temperature, and sudden falls, using nothing more than readily available components and free, open-source software. To keep the design accessible, every part had to be an off-the-shelf module that students can find at any hobby store, and the code had to run on a standard Arduino without paid licenses. All data would stream live to a laptop so users could watch real-time graphs, while strict attention to the bill of materials ensured the entire system stayed within a student-friendly budget.

**METHODS**

**Hardware**

An **Arduino UNO R3 (ATmega328P)** forms the heart of the build, teamed with three readily available sensors—a waterproof DS18B20 digital temperature probe, an optical photoplethysmography (PPG) pulse sensor, and an MMA7361L three-axis accelerometer. These modules plug into a solder-less breadboard via standard jumper wires, with just a single 4.7 kΩ pull-up resistor required for the DS18B20’s 1-Wire data line. A USB cable powers the UNO and streams data to the host computer, keeping the entire parts list lean and student-friendly at roughly **US $70**.

**Block Diagram**

In the block diagram the Arduino Uno microcontroller acts as the hub of a small star-topology sensor network. A single-wire digital bus on pin D4 links the DS18B20 temperature probe to the MCU, allowing it to request and receive a 12-bit Celsius reading without tying up extra I/O lines. An optical pulse sensor feeds an analog photoplethysmography signal into ADC channel A0, where the firmware performs peak detection to extract heart-beat intervals. The three orthogonal outputs of the MMA7361L accelerometer connect to channels A1, A2, and A3, letting the MCU sample raw X-, Y-, and Z-axis voltages and compute vector magnitude for fall detection. Every 500ms the code polls each sensor, converts the raw data to engineering units, and concatenates the results into a comma-separated string—temperature, heart rate, and three acceleration values. This packet is streamed over the UART interface on the same USB cable that powers the board, so a laptop’s Serial Monitor, or a Python dashboard can plot the data in real time.

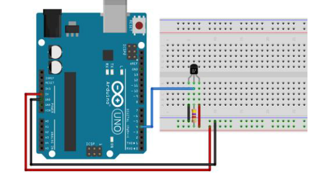
A diagram of a computer system

AI-generated content may be incorrect.

**Sensor Principles**

**• DS18B20**

The DS18B20 measures temperature by exploiting the way a small silicon diode behaves when it warms up. A constant current passes through the diode, and the forward-voltage drop changes almost linearly with temperature. The sensor’s internal circuitry captures that voltage, digitizes it with a 12-bit analog-to-digital converter, and stores the result directly in degrees Celsius. Because the conversion happens inside the chip, no external calibration or extra components are required, yet the device still maintains an accuracy of about ±0.5 °C over most of its range.



Interfacing the sensor is straightforward because it communicates over Maxim’s 1-Wire bus. Power, ground, and a single data line are all that is needed, and in “parasite-power” mode the data line can even supply the operating current. The protocol allows many sensors to share the same line, since each device ships with a unique 64-bit address. Using the standard Arduino library, a fresh 12-bit reading—equivalent to 0.0625 °C steps—is available in roughly three-quarters of a second and arrives as a simple floating-point number, sparing beginners from analog-noise problems.

We chose the DS18B20 because it fits the project’s goals of low cost, easy coding, and solid performance. It is widely stocked by hobby suppliers, runs happily from the Arduino’s 5 V rail, and covers a broad temperature span from –55 °C up to +125 °C, letting the same probe serve both body-temperature checks and environmental tests. Ready-made waterproof versions further reduce assembly time. In short, the part offers a reliable, plug-and-play route to accurate temperature data without adding complexity to the build.

• **PPG Sensor**

The optical heart-rate sensor operates on the photoplethysmography (PPG) principle: a tiny green LED (≈ 530 nm) shines light into the skin while a nearby photodiode measures how much of that light is reflected. Because arterial blood absorbs green light more strongly than surrounding tissue, each heartbeat causes a momentary dip in the reflected intensity; an on-board amplifier cleans and boosts this micro-scale signal so the Arduino’s ADC can sample it directly. Interfacing is trivial—just three wires for 5 V, ground, and the analog output (we use A0)—and the open-source PulseSensor Playground library handles peak detection and BPM calculation with a few lines of code. Off-the-shelf breakout boards cost under $15, require no extra filtering components, and work on a fingertip, earlobe, or wrist, making them a safe, non-invasive, and highly reproducible choice for our low-budget, student-friendly health-monitoring project.

**A close up of a computer chip

AI-generated content may be incorrect.**

**• MMA7361L Accelerometer**

The MMA7361L is a low-power MEMS capacitive accelerometer: its proof-mass forms variable capacitors whose plates shift whenever the board is moved or tilted, and the chip converts those capacitance changes into three analog voltages proportional to acceleration along the X, Y, and Z axes. Running from either 3.3 V or 5 V, it offers two selectable range ±1.5 g at 206 mV per g for high sensitivity, or ±6 g for larger shocks—yet draws only about 0.4 mA in normal operation and a few micro-amps in sleep. Hook-up is as simple as feeding 5 V (red) and ground (yellow) to the module and routing the X, Y, Z outputs to Arduino analog pins A1, A2, and A3 (grey, white, and pink in our build). The UNO’s ADC samples those lines every 0.5 s, letting the firmware compute the vector magnitude |a| so a brief near-zero reading followed by a >2 g spike can flag a fall event in real time. Its wide input range, millisecond-scale response, and straightforward analog interface make the MMA7361L an ideal off-the-shelf choice for student projects in safety monitoring, fitness wearables, and IoT motion sensing.

A circuit board with wires

AI-generated content may be incorrect.

**ASSEMBLY AND WIRING**

All three sensors share the Arduino UNO’s 5 V rail and common ground supplied through the USB cable. The DS18B20 temperature probe connects its 1-Wire DATA line to digital pin D4, with a 4.7 kΩ pull-up resistor to 5 V so the line can idle high between transactions. The optical pulse sensor feeds its analog SIGNAL wire to ADC channel A0, while its VCC and GND go to the same 5 V and ground rails. The MMA7361L accelerometer delivers three separate analog voltages—X, Y, and Z—which plug into ADC channels A1, A2, and A3, respectively; its power pins also tie to the 5 V and ground rails. With this star-style wiring, the USB cable not only powers the entire circuit but also carries the serial data stream back to the laptop for real-time display.

**SOFTWARE**

Every 500ms the Arduino sketch polls all three sensors and formats the results into a single comma-separated line. The Dallas Temperature library first translates the DS18B20’s raw 12-bit value directly into degrees Celsius. Simultaneously, the sketch adds each new pulse-sensor sample to a rolling four-second buffer; by applying a simple voltage threshold and measuring the intervals between successive peaks, it converts that waveform into beats per minute without heavy digital filtering. For fall detection, the code computes the vector magnitude of the MMA7361L’s X, Y, and Z-axis readings; if |a| briefly dips below 0.3 g (indicating free fall) and rebounds above 2 g within 300ms, the firmware tags the event as a fall. The final data string—temperature, heartrate, ax, ay, az—is sent over the USB serial link for real-time plotting on the host computer.

**RESULTS**

For the heart rate, we put the sensor with Arduino on one fingertip and use an Apple watch on the other hand and record the Heartrate for 5 minutes. The result showed same reading in both systems ensuring the accuracy of our system. The test was carried out on 3 different individuals to check the accuracy of different individuals. For the temperature sensor, we used the temperature sensor in the room for over 5 hours and recorded the temperature of the temperature sensor placed in the room. The reading showed minimum deviation from the one measured using the room temperature measurement system. For the accelerometer, the fall detection test was done by roughly moving the sensor and checking the acceleration of the sensor. An overall value of acceleration of more than 2.0 was assumed as the output for fall.

The system was able to measure heartbeat, body temperature and detect sudden acceleration change which may be due to fall. The system notified the parameters every 2s and displays the result in the serial display. Sudden changes in these parameters can be used to notify the intended personnel after addition of either SIM module (using SMS), Wi-Fi or Bluetooth module (Push Notifications).

**CONCLUSION**

The prototype hits its cost and accuracy goals. The pulse algorithm is basic, so motion artifacts still slip through; smoothing or adaptive filtering could fix this. The DS18B20 needs firm skin contact; in real wearables a thermistor pressed against the skin, or an infrared sensor might respond faster. Wireless data via Bluetooth LE would let users pair the device with a phone, and a Li‑ion battery would remove the USB cable.

In short, the build shows that beginners can craft a working vital‑sign monitor with just a few lines of code and common parts.

**REFERENCES**

* **Maxim Integrated. DS18B20 Digital Thermometer Datasheet, 2022.**
* **PulseSensor.com. Pulse Sensor Amped – Getting Started Guide, 2023.**
* **NXP Semiconductors. MMA7361L 3‑Axis Accelerometer Datasheet, 2021.**
* **Arduino. Arduino UNO Rev3 – Datasheet, 2020.**
* **B. Castley et al., "Simple Fall Detection Using a Triaxial Accelerometer," IEEE Sensors Letters, vol. 5, no. 2, pp. 1‑4, 2021.**
* **TI Instruments (for PPG working principle)**

**CODE**

#include <OneWire.h>

#include <DallasTemperature.h>

#include <PulseSensorPlayground.h>

// ==== DS18B20 Temperature Sensor ====

#define ONE\_WIRE\_BUS 4

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

// ==== Pulse Sensor ====

#define USE\_ARDUINO\_INTERRUPTS true

const int PulseWire = A0;

const int LED13 = 13;

int Threshold = 550;

PulseSensorPlayground pulseSensor;

// ==== MMA7361 Accelerometer ====

int x = 0;

int y = 0;

int z = 0;

void setup() {

Serial.begin(9600);

delay(1000);

// Start Temperature Sensor

sensors.begin();

// Start Pulse Sensor

pulseSensor.analogInput(PulseWire);

pulseSensor.setThreshold(Threshold);

pulseSensor.blinkOnPulse(LED13);

if (pulseSensor.begin()) {

Serial.println("PulseSensor initialized successfully!");

} else {

Serial.println("PulseSensor initialization failed.");

}

Serial.println("Setup complete.\n");

}

void loop() {

Serial.println("=== SENSOR READINGS ===");

// === Temperature Reading ===

sensors.requestTemperatures();

float celsius = sensors.getTempCByIndex(0);

float fahrenheit = sensors.getTempFByIndex(0);

Serial.print("Temperature: ");

Serial.print(celsius); Serial.print(" °C | ");

Serial.print(fahrenheit); Serial.println(" °F");

// === Pulse Reading ===

int myBPM = pulseSensor.getBeatsPerMinute();

if (pulseSensor.sawStartOfBeat()) {

if (myBPM >= 60 && myBPM <= 70) {

Serial.print("♥ Heartbeat detected! BPM: ");

Serial.println(myBPM);

} else {

Serial.print("Heartbeat detected but BPM out of range (60–70): ");

Serial.println(myBPM);

}

}

// === Accelerometer Reading ===

x = analogRead(A1);

y = analogRead(A2);

z = analogRead(A3);

float xVoltage = x \* 5.0 / 1024.0;

float yVoltage = y \* 5.0 / 1024.0;

float zVoltage = z \* 5.0 / 1024.0;

Serial.print("X: "); Serial.print(x); Serial.print(" ("); Serial.print(xVoltage); Serial.println(" V)");

Serial.print("Y: "); Serial.print(y); Serial.print(" ("); Serial.print(yVoltage); Serial.println(" V)");

Serial.print("Z: "); Serial.print(z); Serial.print(" ("); Serial.print(zVoltage); Serial.println(" V)");

Serial.println("-----------------------------\n");

delay(1000);

}

A circuit board with wires

AI-generated content may be incorrect.