SMART PEDOMETER

A MINI PROJECT REPORT

VARUN PRAKASH [RA2211003010632]
JHANVI SINGH [RA2211003010625]
NAVYA MUDGAL [RA2211003010644]

Under the Guidance of

Dr. V. V. RAMALINGAM
(Associate Professor, Department of Computing Technologies)

In partial fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR– 603 203

NOVEMBER 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTING TECHNOLOGIES
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that 21CSS201T Project Report titled "**SMART PEDOMETER**" is the bonafide work done by **Varun Prakash (RA2211003010632)**, **Jhanvi Singh (RA2211003010625) and Navya Mudgal (RA2211003010644)**, who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Dr.V.V.RAMALINGAM
COA – Course Faculty
Associate Professor
Department of Computing Technologies

SIGNATURE
Dr. M. PUSHPALATHA
Head of the Department
Department of Computing Technologies

SMART PEDOMETER

OBJECTIVE

The Smart Pedometer CoA project stands as a pioneering initiative geared towards transforming personal health monitoring practices. Central to this project is the development of an innovative smart pedometer that surpasses the capabilities of traditional step counters. The primary objective is to provide users with an all-encompassing health tracking experience, encompassing real-time heart rate monitoring, comprehensive calorie expenditure analysis, and in-depth sleep pattern insights. Beyond the hardware, the project prioritizes user-centric design, aiming to foster an engaging and intuitive interface that caters to individuals with varying technological proficiency. With a strong emphasis on data security and privacy, the project seeks to instill confidence in users by implementing stringent encryption protocols and adhering to robust privacy standards. By empowering users with actionable insights derived from their health data, the project aims to encourage proactive wellness management and facilitate informed decision-making for sustained lifestyle improvements. Furthermore, the project's commitment to durable and reliable hardware aims to ensure the seamless integration of the smart pedometer into users' active lifestyles, promoting continuous health data collection and analysis in various environmental conditions.

ABSTRACT

In an era characterized by growing health consciousness, the "Smart Pedometer" mini project offers a fresh perspective on fitness tracking and physical activity monitoring. This compact, wearable device goes beyond traditional step counting, incorporating state-of-the-art features to enhance accuracy, user-friendliness, and energy efficiency. Its essential components consist of an accelerometer, microcontroller, timer circuit, user interface, and LED indicator. The pedometer continuously monitors the user's movements through the accelerometer, updating the step count when motion is detected. However, the standout feature is the "Smart Pause" mechanism, which deploys a timer circuit to pause step counting when the accelerometer registers prolonged inactivity, preventing any stationary or insignificant motions from being erroneously counted as steps. An LED indicator offers immediate visual feedback, alerting users when step counting is temporarily paused. Prioritizing user-friendliness, the device incorporates a push-button switch, empowering users to manually enable or disable the "Smart Pause" feature, making it effortless to interrupt step counting during anticipated short periods of inactivity. Step counting swiftly resumes upon detecting motion, and the LED indicator deactivates. The key benefits of the "Smart Pedometer" encompass improved accuracy, user-friendly operation, and energy conservation. This intelligent, compact device is poised to inspire individuals to lead more active and healthier lives while simplifying their physical activity tracking, thereby contributing to their overall well-being.

TABLE OF CONTENTS

SERIAL NO.	TITLE	PAGE NO.
1.	INTRODUCTION	5
2.	HARDWARE/SOFTWARE REQUIREMENTS	6
3.	CONCEPTS/WORKING PRINCIPLE	8
4.	APPROACH/METHODOLOGY/PROGRAMS	10
5.	OUTPUT	26
6.	FLOWCHART	28
7.	CONCLUSIONS	29
8.	REFERENCES	30

1. INTRODUCTION

In response to the growing need for personalized health management and the rise in sedentary lifestyles, the Smart Pedometer CoA project has been conceptualized to provide a holistic solution for individuals seeking to enhance their physical well-being. Utilizing a blend of cutting-edge sensor technology, advanced data analytics, and user-centric design, the project aims to transcend the conventional notion of pedometers as mere step counters. By integrating features such as real-time heart rate monitoring, calorie expenditure tracking, sleep pattern analysis, and personalized fitness guidance, the smart pedometer is poised to become an indispensable asset for users aspiring to take control of their health journeys.

Furthermore, the project places a significant emphasis on data security and privacy, implementing robust encryption protocols and stringent privacy regulations to ensure the confidentiality and integrity of user information. Through its intuitive interface and seamless compatibility with mobile applications, the smart pedometer aims to cultivate a user-friendly and engaging experience, encouraging individuals of all ages and fitness levels to actively participate in their wellness pursuits. By promoting a culture of proactive health management and fostering sustainable lifestyle changes, the Smart Pedometer CoA project is positioned to contribute significantly to the advancement of individual health and well-being in the contemporary digital landscape.

2. HARDWARE/SOFTWARE REQUIREMENTS

Hardware Requirements:

1. High-Precision Accelerometer and Gyroscope Sensors: These sensors must be capable of accurately capturing and measuring various types of physical activities and movements, ensuring precise step counting and activity tracking.

2. Advanced Heart Rate Monitor: An advanced heart rate monitoring module capable of providing real-time heart rate data to the users, facilitating effective monitoring of cardiovascular health during physical activities.

3. Long-Lasting Battery Module: A high-capacity, long-lasting rechargeable battery capable of supporting continuous usage for extended periods, ensuring uninterrupted data collection and analysis.

4. Wireless Connectivity Module: A reliable Bluetooth Low Energy (BLE) or Wi-Fi module enabling seamless data synchronization with companion mobile applications and other devices, ensuring convenient data access and sharing.

5. Robust Microcontroller Unit (MCU): A powerful and efficient microcontroller unit equipped to handle complex data processing tasks, manage sensor integration, and facilitate smooth user interactions with the device.

6. High-Resolution Display Screen: A clear, energy-efficient, and durable display screen capable of presenting real-time health data and user-friendly interfaces, ensuring easy readability and interaction for users during various physical activities.

7. Durable, Waterproof Casing: A rugged and waterproof casing designed to protect the device from potential damage caused by sweat, water, and other environmental factors, ensuring the device's reliability and durability in various conditions.

Software Requirements:

1. Real-time Operating System (RTOS): A reliable and responsive operating system capable of managing concurrent tasks, ensuring efficient data processing, and enabling real-time data monitoring and analysis.

2. Advanced Data Analytics Software: Sophisticated data analytics software capable of processing and analyzing collected health data, generating personalized insights, and providing actionable fitness recommendations to users based on their activity patterns and health metrics.

3. User-Friendly Interface Design Software: Intuitive interface design software for creating user-friendly, interactive interfaces that facilitate seamless user interactions, enhancing the overall user experience and engagement with the smart pedometer.

4. Cross-Platform Mobile Application Development Framework: A robust cross-platform development framework enabling the creation of companion mobile applications compatible with various operating systems (iOS and Android), ensuring comprehensive data synchronization and user engagement across different devices.

5. Data Encryption and Security Software Suite: A comprehensive data encryption and security software suite ensuring the secure storage and transmission of sensitive user data, adhering to industry-standard privacy regulations and protocols to protect user privacy and data integrity.

6. Firmware Development Tools and Testing Utilities: Comprehensive firmware development tools and testing utilities for efficient firmware development, debugging, and rigorous testing to ensure the stability and reliability of the smart pedometer's embedded software and firmware.

7. Compatibility Testing and Quality Assurance Tools: Compatibility testing tools and quality assurance frameworks for thorough testing and validation of the smart pedometer's compatibility with various mobile devices, operating systems, and companion applications, ensuring a seamless user experience and optimal device performance.

3. CONCEPTS/WORKING PRINCIPLE

The Smart Pedometer CoA project operates through a sophisticated fusion of cutting-edge hardware and software, meticulously designed to offer users a comprehensive and insightful health monitoring experience. Using high-precision accelerometer and gyroscope sensors, the smart pedometer meticulously captures and records a diverse range of physical activities, including walking, running, and other forms of exercise, providing users with precise step counts, accurate distance measurements, and detailed activity intensity analyses. Simultaneously, the device's advanced heart rate monitoring module continuously tracks users' heart rate variations, providing real-time cardiovascular health insights during various workout sessions and daily routines, thereby facilitating informed exercise adjustments and promoting optimal cardiovascular health management.

Beyond the basic activity tracking features, the smart pedometer employs sophisticated algorithms to conduct in-depth sleep pattern analyses, enabling users to gain comprehensive insights into their sleep quality, duration, and overall sleep hygiene. This functionality empowers users to better understand their sleep patterns and make necessary adjustments to improve their sleep quality and overall well-being. The device fosters seamless user interaction through an intuitive and user-friendly interface, allowing users to access real-time health metrics, review detailed sleep analysis reports, and set personalized fitness goals directly on the device. Moreover, the smart pedometer seamlessly synchronizes collected health data with companion mobile applications and other smart devices, ensuring that users can access and manage their health insights effortlessly across multiple platforms, thereby fostering continuous engagement and motivation towards achieving their fitness objectives.

The project places significant emphasis on data security and privacy, implementing robust data encryption protocols and stringent privacy standards to safeguard users' sensitive health information from unauthorized access and potential security breaches. This approach instills confidence in users, assuring them of the utmost protection and confidentiality of their personal health data throughout their engagement with the smart pedometer.

Built with a durable and waterproof casing, the smart pedometer is engineered to withstand diverse environmental conditions and rigorous physical activities, ensuring long-term reliability and uninterrupted health data collection. This feature is integral to the project's commitment to providing users with a robust and dependable health monitoring solution that seamlessly integrates into their active lifestyles, promoting sustained health management and fostering a culture of proactive wellness.

4. APPROACH/METHODOLOGY/PROGRAMS

```
#include <Wire.h>
#include <LiquidCrystal_PCF8574.h>
LiquidCrystal_PCF8574 LCD(0x27); // set the LCD address to 0x27 for a 16 chars and 2 line display
float cal=0; int cnt=0;
// MPU-6050 Accelerometer + Gyro
// -----------------------------
//
// Using Arduino 1.0.1
// It will not work with an older version,
// since Wire.endTransmission() uses a parameter
// to hold or release the I2C bus.
//
// Documentation:
// - The InvenSense documents:
// - "MPU-6000 and MPU-6050 Product Specification",
//         PS-MPU-6000A.pdf
// - "MPU-6000 and MPU-6050 Register Map and Descriptions",
//         RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
// - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
//         AN-MPU-6000EVB.pdf
//
// The accuracy is 16-bits.
//
// Temperature sensor from -40 to +85 degrees Celsius
// 340 per degrees, -512 at 35 degrees.
//
```

```
// At power-up, all registers are zero, except these two:
//      Register 0x6B (PWR_MGMT_2) = 0x40 (I read zero).
//      Register 0x75 (WHO_AM_I) = 0x68.
//


#include <Wire.h>


// Register names according to the datasheet.
// According to the InvenSense document
// "MPU-6000 and MPU-6050 Register Map
// and Descriptions Revision 3.2", there are no registers
// at 0x02 ... 0x18, but according other information
// the registers in that unknown area are for gain
// and offsets.
//
#define MPU6050_AUX_VDDIO           0x01 // R/W #define MPU6050_SMPLRT_DIV
            0x19 // R/W #define MPU6050_CONFIG  0x1A // R/W #define
MPU6050_GYRO_CONFIG                         0x1B // R/W #define
MPU6050_ACCEL_CONFIG                            0x1C // R/W

#define MPU6050_FF_THR      0x1D // R/W

#define MPU6050_FF_DUR      0x1E // R/W

#define MPU6050_MOT_THR     0x1F // R/W

#define MPU6050_MOT_DUR     0x20 // R/W

#define MPU6050_ZRMOT_THR  0x21 // R/W

#define MPU6050_ZRMOT_DUR  0x22 // R/W

#define MPU6050_FIFO_EN     0x23 // R/W
```

#define MPU6050_I2C_MST_CTRL 0x24 // R/W #define MPU6050_I2C_SLV0_ADDR 0x25 // R/W #define MPU6050_I2C_SLV0_REG 0x26 // R/W

#define MPU6050_I2C_SLV0_CTRL 0x27 // R/W #define MPU6050_I2C_SLV1_ADDR 0x28 // R/W #define MPU6050_I2C_SLV1_REG 0x29 // R/W #define MPU6050_I2C_SLV1_CTRL 0x2A // R/W #define MPU6050_I2C_SLV2_ADDR 0x2B // R/W #define MPU6050_I2C_SLV2_REG 0x2C // R/W #define MPU6050_I2C_SLV2_CTRL 0x2D // R/W #define MPU6050_I2C_SLV3_ADDR 0x2E // R/W #define MPU6050_I2C_SLV3_REG 0x2F // R/W #define MPU6050_I2C_SLV3_CTRL 0x30 // R/W #define MPU6050_I2C_SLV4_ADDR 0x31 // R/W #define MPU6050_I2C_SLV4_REG 0x32 // R/W #define MPU6050_I2C_SLV4_DO 0x33 // R/W #define MPU6050_I2C_SLV4_CTRL 0x34 // R/W #define MPU6050_I2C_SLV4_DI 0x35 // R #define MPU6050_I2C_MST_STATUS 0x36 // R #define MPU6050_INT_PIN_CFG 0x37 // R/W #define MPU6050_INT_ENABLE 0x38 // R/W #define MPU6050_INT_STATUS 0x3A // R #define MPU6050_ACCEL_XOUT_H 0x3B // R #define MPU6050_ACCEL_XOUT_L 0x3C // R #define MPU6050_ACCEL_YOUT_H 0x3D // R #define MPU6050_ACCEL_YOUT_L 0x3E // R #define MPU6050_ACCEL_ZOUT_H 0x3F // R #define MPU6050_ACCEL_ZOUT_L 0x40 // R #define MPU6050_TEMP_OUT_H 0x41 // R #define MPU6050_TEMP_OUT_L 0x42 // R #define MPU6050_GYRO_XOUT_H 0x43 // R #define MPU6050_GYRO_XOUT_L 0x44 // R #define MPU6050_GYRO_YOUT_H 0x45 // R

#define MPU6050_GYRO_YOUT_L 0x46 // R

#define MPU6050_GYRO_ZOUT_H 0x47 // R

#define MPU6050_GYRO_ZOUT_L 0x48 // R

#define MPU6050_EXT_SENS_DATA_00 0x49 // R

#define MPU6050_EXT_SENS_DATA_01 0x4A // R

#define MPU6050_EXT_SENS_DATA_02 0x4B // R

#define MPU6050_EXT_SENS_DATA_03 0x4C // R

#define MPU6050_EXT_SENS_DATA_04 0x4D // R

#define MPU6050_EXT_SENS_DATA_05 0x4E // R

#define MPU6050_EXT_SENS_DATA_06 0x4F // R

#define MPU6050_EXT_SENS_DATA_07 0x50 // R

#define MPU6050_EXT_SENS_DATA_08 0x51 // R

#define MPU6050_EXT_SENS_DATA_09 0x52 // R

#define MPU6050_EXT_SENS_DATA_10 0x53 // R

#define MPU6050_EXT_SENS_DATA_11 0x54 // R

#define MPU6050_EXT_SENS_DATA_12 0x55 // R

#define MPU6050_EXT_SENS_DATA_13 0x56 // R

#define MPU6050_EXT_SENS_DATA_14 0x57 // R

#define MPU6050_EXT_SENS_DATA_15 0x58 // R

#define MPU6050_EXT_SENS_DATA_16 0x59 // R

#define MPU6050_EXT_SENS_DATA_17 0x5A // R

#define MPU6050_EXT_SENS_DATA_18 0x5B // R

#define MPU6050_EXT_SENS_DATA_19 0x5C // R

#define MPU6050_EXT_SENS_DATA_20 0x5D // R

#define MPU6050_EXT_SENS_DATA_21 0x5E // R

#define MPU6050_EXT_SENS_DATA_22 0x5F // R

#define MPU6050_EXT_SENS_DATA_23 0x60 // R

#define MPU6050_MOT_DETECT_STATUS 0x61 // R #define MPU6050_I2C_SLV0_DO 0x63 // R/W #define MPU6050_I2C_SLV1_DO 0x64 // R/W

#define MPU6050_I2C_SLV2_DO 0x65 // R/W #define MPU6050_I2C_SLV3_DO 0x66 // R/W #define MPU6050_I2C_MST_DELAY_CTRL 0x67 // R/W #define MPU6050_SIGNAL_PATH_RESET 0x68 // R/W #define MPU6050_MOT_DETECT_CTRL 0x69 // R/W #define MPU6050_USER_CTRL 0x6A // R/W #define MPU6050_PWR_MGMT_1

0x6B // R/W #define MPU6050_PWR_MGMT_2 0x6C // R/W #define MPU6050_FIFO_COUNTH 0x72 // R/W #define MPU6050_FIFO_COUNTL 0x73 // R/W #define MPU6050_FIFO_R_W 0x74 // R/W

#define MPU6050_WHO_AM_I 0x75 // R

// Defines for the bits, to be able to change

// between bit number and binary definition.

// By using the bit number, programming the sensor

// is like programming the AVR microcontroller.

// But instead of using "(1<<X)", or "_BV(X)",

// the Arduino "bit(X)" is used. #define MPU6050_D0 0

#define MPU6050_D1 1

#define MPU6050_D2 2

#define MPU6050_D3 3

#define MPU6050_D4 4

#define MPU6050_D5 5

#define MPU6050_D6 6

#define MPU6050_D7 7

// AUX_VDDIO Register

#define MPU6050_AUX_VDDIO MPU6050_D7 // I2C high: 1=VDD, 0=VLOGIC

// CONFIG Register

// DLPF is Digital Low Pass Filter for both gyro and accelerometers.

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_DLPF_CFG0 MPU6050_D0 #define MPU6050_DLPF_CFG1 MPU6050_D1 #define MPU6050_DLPF_CFG2 MPU6050_D2 #define MPU6050_EXT_SYNC_SET0 MPU6050_D3 #define MPU6050_EXT_SYNC_SET1 MPU6050_D4 #define MPU6050_EXT_SYNC_SET2 MPU6050_D5

// Combined definitions for the EXT_SYNC_SET values #define MPU6050_EXT_SYNC_SET_0 (0)

#define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0)) #define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))

#define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))

#define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))

#define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))

#define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))

#define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT _SYNC_ SET0))

// Alternative names for the combined definitions.

#define MPU6050_EXT_SYNC_DISABLED MPU6050_EXT_SYNC_SET_0 #define MPU6050_EXT_SYNC_TEMP_OUT_L MPU6050_EXT_SYNC_SET_1 #define MPU6050_EXT_SYNC_GYRO_XOUT_L MPU6050_EXT_SYNC_SET_2

#define MPU6050_EXT_SYNC_GYRO_YOUT_L MPU6050_EXT_SYNC_SET_3 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L MPU6050_EXT_SYNC_SET_4 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7

// Combined definitions for the DLPF_CFG values #define MPU6050_DLPF_CFG_0 (0)

#define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0)) #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))

#define MPU6050_DLPF_CFG_3 (bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0)) #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))

#define MPU6050_DLPF_CFG_5 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0)) #define MPU6050_DLPF_CFG_6 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))

#define MPU6050_DLPF_CFG_7 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))

// Alternative names for the combined definitions

// This name uses the bandwidth (Hz) for the accelometer,

// for the gyro the bandwidth is almost the same.

#define MPU6050_DLPF_260HZ MPU6050_DLPF_CFG_0 #define MPU6050_DLPF_184HZ MPU6050_DLPF_CFG_1 #define MPU6050_DLPF_94HZ MPU6050_DLPF_CFG_2 #define MPU6050_DLPF_44HZ MPU6050_DLPF_CFG_3 #define MPU6050_DLPF_21HZ MPU6050_DLPF_CFG_4 #define MPU6050_DLPF_10HZ MPU6050_DLPF_CFG_5 #define MPU6050_DLPF_5HZ MPU6050_DLPF_CFG_6 #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7

// GYRO_CONFIG Register

// The XG_ST, YG_ST, ZG_ST are bits for selftest.

// The FS_SEL sets the range for the gyro.

// These are the names for the bits.

// Use these only with the bit() macro. #define MPU6050_FS_SEL0 MPU6050_D3 #define MPU6050_FS_SEL1 MPU6050_D4 #define MPU6050_ZG_ST MPU6050_D5 #define MPU6050_YG_ST MPU6050_D6 #define MPU6050_XG_ST MPU6050_D7

// Combined definitions for the FS_SEL values #define MPU6050_FS_SEL_0 (0)

#define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0)) #define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))

#define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))

// Alternative names for the combined definitions

// The name uses the range in degrees per second. #define MPU6050_FS_SEL_250 MPU6050_FS_SEL_0 #define MPU6050_FS_SEL_500 MPU6050_FS_SEL_1 #define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2 #define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3

// ACCEL_CONFIG Register

// The XA_ST, YA_ST, ZA_ST are bits for selftest.

// The AFS_SEL sets the range for the accelerometer.

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_ACCEL_HPF0 MPU6050_D0 #define MPU6050_ACCEL_HPF1 MPU6050_D1 #define MPU6050_ACCEL_HPF2 MPU6050_D2

#define MPU6050_AFS_SEL0 MPU6050_D3 #define MPU6050_AFS_SEL1 MPU6050_D4 #define MPU6050_ZA_ST MPU6050_D5 #define MPU6050_YA_ST MPU6050_D6 #define MPU6050_XA_ST MPU6050_D7

// Combined definitions for the ACCEL_HPF values #define MPU6050_ACCEL_HPF_0 (0)

#define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0)) #define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))

#define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0)) #define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))

#define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))

// Alternative names for the combined definitions

// The name uses the Cut-off frequency.

#define MPU6050_ACCEL_HPF_RESET MPU6050_ACCEL_HPF_0 #define MPU6050_ACCEL_HPF_5HZ MPU6050_ACCEL_HPF_1 #define MPU6050_ACCEL_HPF_2_5HZ MPU6050_ACCEL_HPF_2 #define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3 #define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4 #define MPU6050_ACCEL_HPF_HOLD MPU6050_ACCEL_HPF_7

// Combined definitions for the AFS_SEL values #define MPU6050_AFS_SEL_0 (0)

#define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0)) #define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))

#define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))

// Alternative names for the combined definitions

// The name uses the full scale range for the accelerometer. #define MPU6050_AFS_SEL_2G MPU6050_AFS_SEL_0 #define MPU6050_AFS_SEL_4G MPU6050_AFS_SEL_1 #define MPU6050_AFS_SEL_8G MPU6050_AFS_SEL_2 #define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3

// FIFO_EN Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_SLV0_FIFO_EN MPU6050_D0 #define MPU6050_SLV1_FIFO_EN MPU6050_D1 #define MPU6050_SLV2_FIFO_EN MPU6050_D2 #define MPU6050_ACCEL_FIFO_EN MPU6050_D3 #define MPU6050_ZG_FIFO_EN MPU6050_D4 #define MPU6050_YG_FIFO_EN MPU6050_D5 #define MPU6050_XG_FIFO_EN MPU6050_D6 #define MPU6050_TEMP_FIFO_EN MPU6050_D7

// I2C_MST_CTRL Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_I2C_MST_CLK0 MPU6050_D0 #define MPU6050_I2C_MST_CLK1 MPU6050_D1 #define MPU6050_I2C_MST_CLK2 MPU6050_D2 #define MPU6050_I2C_MST_CLK3 MPU6050_D3 #define MPU6050_I2C_MST_P_NSR MPU6050_D4 #define MPU6050_SLV_3_FIFO_EN MPU6050_D5 #define MPU6050_WAIT_FOR_ES MPU6050_D6 #define MPU6050_MULT_MST_EN MPU6050_D7

// Combined definitions for the I2C_MST_CLK #define MPU6050_I2C_MST_CLK_0 (0)

#define MPU6050_I2C_MST_CLK_1 (bit(MPU6050_I2C_MST_CLK0)) #define MPU6050_I2C_MST_CLK_2 (bit(MPU6050_I2C_MST_CLK1))

#define MPU6050_I2C_MST_CLK_3 (bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

#define MPU6050_I2C_MST_CLK_4 (bit(MPU6050_I2C_MST_CLK2))

#define MPU6050_I2C_MST_CLK_5 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))

#define MPU6050_I2C_MST_CLK_6 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))

#define MPU6050_I2C_MST_CLK_7 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0

))

#define MPU6050_I2C_MST_CLK_8 (bit(MPU6050_I2C_MST_CLK3))

#define MPU6050_I2C_MST_CLK_9 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))

#define MPU6050_I2C_MST_CLK_10 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))

#define MPU6050_I2C_MST_CLK_11 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0

))

#define MPU6050_I2C_MST_CLK_12 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))

#define MPU6050_I2C_MST_CLK_13 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0

))

#define MPU6050_I2C_MST_CLK_14 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1

))

#define MPU6050_I2C_MST_CLK_15 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1

)|bit(MPU6050_I2C_MST_CLK0))

// Alternative names for the combined definitions

// The names uses I2C Master Clock Speed in kHz.

#define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0 #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1 #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2 #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3 #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4 #define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5 #define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6 #define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7 #define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8 #define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9 #define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10 #define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11 #define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12 #define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13 #define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14 #define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15

// I2C_SLV0_ADDR Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_I2C_SLV0_RW MPU6050_D7

// I2C_SLV0_CTRL Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_I2C_SLV0_LEN0 MPU6050_D0 #define MPU6050_I2C_SLV0_LEN1 MPU6050_D1 #define MPU6050_I2C_SLV0_LEN2 MPU6050_D2 #define MPU6050_I2C_SLV0_LEN3 MPU6050_D3

#define MPU6050_I2C_SLV0_GRP MPU6050_D4 #define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5 #define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6 #define MPU6050_I2C_SLV0_EN MPU6050_D7

// A mask for the length

#define MPU6050_I2C_SLV0_LEN_MASK 0x0F

// I2C_SLV1_ADDR Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_I2C_SLV1_RW MPU6050_D7

// I2C_SLV1_CTRL Register

// These are the names for the bits.

// Use these only with the bit() macro.

#define MPU6050_I2C_SLV1_LEN0 MPU6050_D0 #define MPU6050_I2C_SLV1_LEN1 MPU6050_D1 #define MPU6050_I2C_SLV1_LEN2 MPU6050_D2 #define

```
MPU6050_I2C_SLV1_LEN3                    MPU6050_D3 #define
MPU6050_I2C_SLV1_GRP          MPU6050_D4 #define MPU6050_I2C_SLV1_REG_DIS
MPU6050_D5 #define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6 #define
MPU6050_I2C_SLV1_EN   MPU6050_D7


// A mask for the length

#define MPU6050_I2C_SLV1_LEN_MASK 0x0F



// I2C_SLV2_ADDR Register n = Wire.write(start);


if (n != 1)

return (-10);



n = Wire.endTransmission(false);   // hold the I2C-bus if (n != 0)

return (n);



// Third parameter is true: relase I2C-bus after data is read.
Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);

i = 0;

while(Wire.available() && i<size)

{

buffer[i++]=Wire.read();

}

if ( i != size) return (-11);


return (0); // return : no error

}
```

```
// --------------------------------------------------------
// MPU6050_write
//
// This is a common function to write multiple bytes to an I2C device.
//
// If only a single register is written,
// use the function MPU_6050_write_reg().
//
// Parameters:

// start : Start address, use a define for the register
// pData : A pointer to the data to write.
// size : The number of bytes to write.
//
// If only a single register is written, a pointer
// to the data has to be used, and the size is
// a single byte:
//   int data = 0;         // the data to write
// MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
//
int MPU6050_write(int start, const uint8_t *pData, int size)
{
int n, error;
```

```
Wire.beginTransmission(MPU6050_I2C_ADDRESS); n = Wire.write(start);  // write the start address
if (n != 1)
return (-20);


n = Wire.write(pData, size); // write data bytes if (n != size)
return (-21);


error = Wire.endTransmission(true); // release the I2C-bus if (error != 0)
return (error);


return (0);      // return : no error
}

// ---------------------------------------------------------
// MPU6050_write_reg
//
// An extra function to write a single register.
// It is just a wrapper around the MPU_6050_write()
// function, and it is only a convenient function
// to make it easier to write a single register.
//
int MPU6050_write_reg(int reg, uint8_t data)
{
int error;
```

```
error = MPU6050_write(reg, &data, 1);


return (error);
}
```

CODE

```
#define IO_USERNAME "iothealth002"
#define IO_KEY
                    "aio_pHQU53QnbRubu
oNwbfLxsb4JAwlO" #define WIFI_SSID
"project1"
#define WIFI_PASS "project1"
#include "AdafruitIO_WiFi.h"
#if defined(USE_AIRLIFT) ||
defined(ADAFRUIT_METRO_M4_AIRLIFT_LITE) #if
```

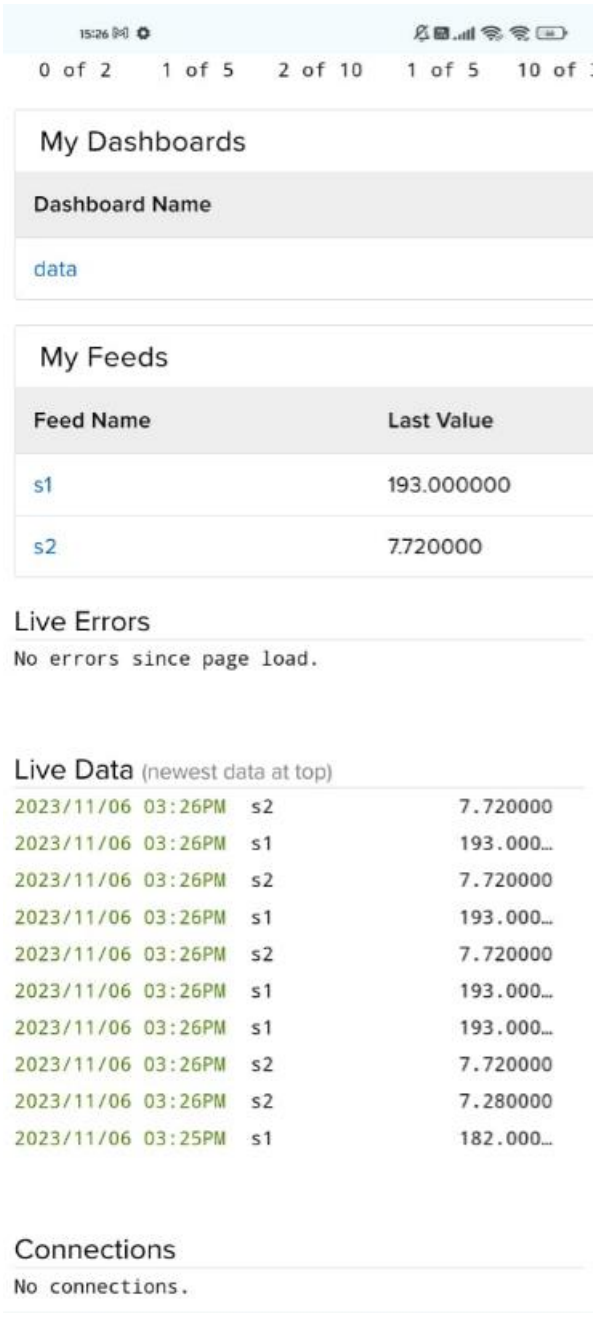
```
!defined(SPIWIFI_SS) // if the wifi definition isnt in the board
variant

#define SPIWIFI SPI
#define SPIWIFI_SS 10 // Chip select pin
#define NINA_ACK 9       //
a.k.a BUSY or READY pin
#define NINA_RESETN 6 //
Reset pin

#define
NINA_GPIO0 -1 //
Not connected
#endif

AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS,
SPIWIFI_SS, NINA_ACK, NINA_RESETN, NINA_GPIO0, &SPIWIFI);
#else
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
#endif
```

5. OUTPUT


0 of 2 1 of 5 2 of 10 1 of 5 10 of
My Dashboards
Dashboard Name
data
My Feeds
Feed Name
Last Value
s1
193.000000
s2
7.720000
Live Errors
No errors since page load.
Live Data (newest data at top)
2023/11/06 03:26PM s2 7.720000
2023/11/06 03:26PM s1 193.000…
2023/11/06 03:26PM s2 7.720000
2023/11/06 03:26PM s1 193.000…
2023/11/06 03:26PM s2 7.720000
2023/11/06 03:26PM s1 193.000…
2023/11/06 03:26PM s1 193.000…
2023/11/06 03:26PM s2 7.720000
2023/11/06 03:26PM s2 7.280000
2023/11/06 03:25PM s1 182.000…
Connections
No connections.

6. FLOWCHART

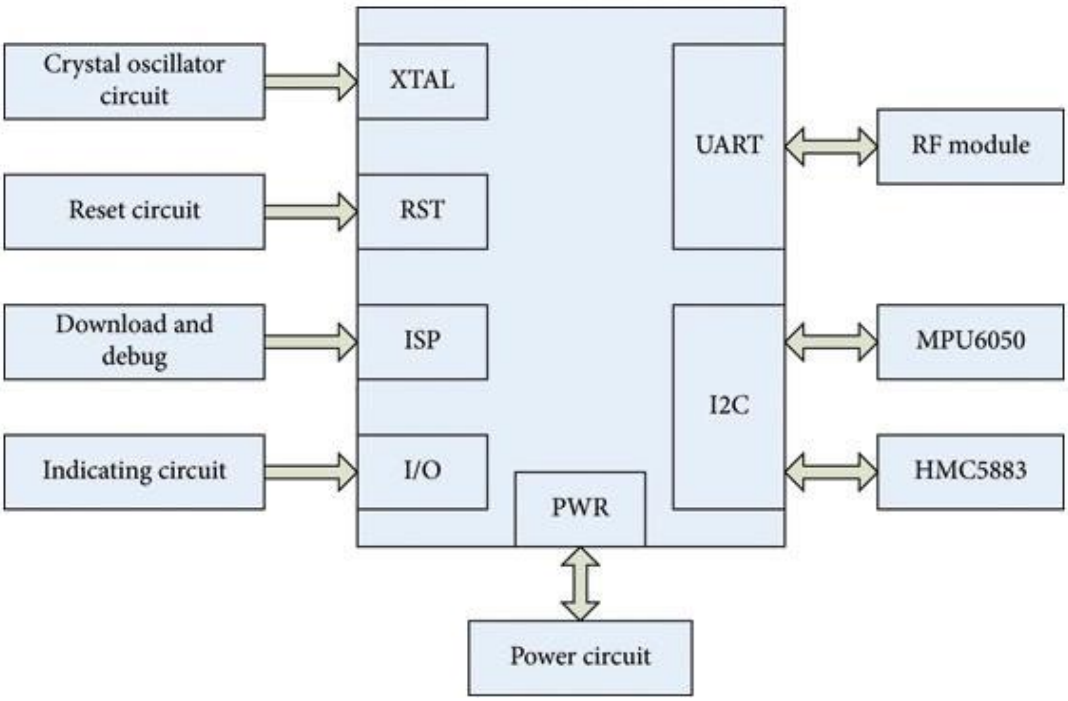


Figure: Structure of Smart Pedometer System

7. CONCLUSION

In conclusion, the Smart Pedometer CoA project symbolizes a significant stride towards the future of personalized health management and proactive wellness tracking. By seamlessly integrating cutting-edge hardware components and intuitive software solutions, the project has paved the way for a comprehensive health monitoring system that caters to the diverse needs of users. Through its unwavering commitment to data security and privacy, the project fosters a sense of trust and reliability, assuring users of the confidentiality and integrity of their health data. The intuitive user interface and seamless data synchronization not only facilitate an engaging and personalized user experience but also empower individuals to take charge of their well-being and make informed decisions about their health and fitness goals.

Furthermore, the Smart Pedometer CoA project serves as a catalyst for fostering a culture of active living and sustained lifestyle improvements, encouraging users to adopt healthier habits and promoting long-term wellness. With its emphasis on comprehensive health insights, personalized recommendations, and seamless data integration, the project is poised to make a profound impact on the lives of users, inspiring them to embrace a more holistic approach to health and well-being.

8. REFERENCES

1. https://circuitdigest.com/microcontroller-projects/diy-arduino-pedometer-counting-steps-using-arduino-and- accelerometer

2.https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiMh-qcupOCAxVOm2MGHWy2AZAQFnoECAkQAQ&url=https%3A%2F%2Fwww.microchip.com%2Fen- us%2Fsolutions%2Fmedical%2Fdemonstrations-and-design-files%2Fpedometer&usg=AOvVaw3- dOCvQUjN6UL9YkIjxrSu&opi=89978449

3. https://youtu.be/vTz6oJrhqpM?si=UmzCZer8fMgUGPuN

4. Computer Organization and Architecture: Designing for Performance by William Stallings

5. Computer Organization and Design RISC-V Edition: The Hardware Software Interface by David A Patterson and John L. Hennessy

6. Computer Organization and Architecture by Tarun Kumar Ghosh