University of California

Los Angeles

Rapidly Deployable Internet-of-Things Body Area Network Platform for Medical Devices

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering

by

In Hwan Baek

2016

Abstract of the Thesis

Rapidly Deployable Internet-of-Things Body Area Network Platform for Medical Devices

by

In Hwan Baek

Master of Science in Electrical Engineering

University of California, Los Angeles, 2016

Professor William J. Kaiser, Chair

Biomedical devices in the past provided limited capability for the data acquisition and presented the data in the form of user interface for a care provider to observe. Now, what is required for biomedical devices has fundamentally changed. Many devices must now support secure networking and include a network of sensors to enable machine learning-based sensor fusion for accurate inference of the subject's state.

This thesis introduces an Internet-of-Things (IoT) body area network (BAN) platform for medical devices that will provide rapid development capability with the assurance of security, networking, and the ability to host computationally intensive processes that are now required by medical devices. The BAN platform consists of seven wearable sensor nodes on the chest, wrists, upper legs, and ankles. Each sensor node includes sixteen general-purpose input/output (GPIO) pins, an analog-to-digital converter (ADC), two inter-integrated circuit (I2C) controllers, a serial peripheral interface (SPI), two universal asynchronous receiver transmitters (UART), and a universal serial bus (USB) on-the-go (OTG) to interface with sensors. The platform base model includes 9 degree-of-freedom inertial measurement unit

(9DOF IMU) motion sensors, an electrocardiogram (ECG) sensor, a microphone, and a heart rate sensor. With its flexible interfaces, the platform is highly customizable and more sensors can be easily added.

Each sensor node features an IoT computer-on-module called the Intel Edison. The device can connect to expansion boards for rapid development. Although it has two official expansion options, the BAN platform uses boards from a third party manufacturer due to their small size. Intel provides a library to access the external interfaces. The library is fully compatible only if the Arduino breakout is used. A C library that abstracts /sys/class/gpio interface was developed to access the GPIO. The ADC device used in the platform is an I2C device. A C library was developed that abstracts the I2C communication between the Intel Edison and the ADC to provide an intuitive application program interface (API). The UART interface is accessible via /dev/ttyMFD2. A Python package called PySerial is used to interface the serial port. These interfaces in addition to the Intel's official breakouts and library enable many more applications.

One of the most powerful features of the Intel Edison is its integrated WiFi module, enabling connection within the BAN and to the Internet. Since the BAN platform collects the user's private health and activity data, the connection is secured by transport layer security (TLS). The networking among sensor nodes allows time synchronization with network time protocol (NTP) to have accurate sensor fusion.

Powered by its Intel Atom dual-core processors, the BAN platform can host neural network-based classifiers to monitor users' states. From experiments, the performance of the neural network hosted on the platform was found to be on par with that of neural network implemented in Matlab.

The BAN platform was successfully distributed to freshman, senior, and graduate IoT courses with exceptional assessment records. The IoT courses have shown that the students were able to rapidly develop fully functioning biomedical devices on the BAN platform.

The thesis of In Hwan Baek is approved.

University of California, Los Angeles

2016

Table of Contents

List of Figures

Acknowledgments

I would like to express my sincere gratitude to my advisor, Professor William Kaiser, who has encouraged and inspired me during my master studies. Without his guidance and support, this thesis would not have been possible.

I must also thank my committee members, Professor Gregory Pottie and Professor Robert Candler for their insightful advices on my thesis.

I thank my colleagues at UCLA Wireless Health Institute. My special thanks go to Pranshu Bansal for his help with his expertise in software development, Derrick Chang for his guidance on motion sensing, and Peter Borgstrom for his introduction to artificial neural network and classification.

I have been very fortunate to serve as a teaching assistant for E96 and EE180D. I thank my students for validating the success of my system development. My special thanks go to Susan Krkasharian, Elizabeth Abramyan, Tanya Al-Rehani, Karen Li, Sam Sirott, Nastassy Zagorov, Ryan Romanas, Justin Lazarus, Richard Sun, Timothy Jo, Louis Truong, Yun Zhang, Yuhuai Lin, and Manni Chen for allowing me to mention their projects.

CHAPTER 1

Introduction

In the past, biomedical devices provided limited capability for the acquisition of data from instruments and presented the data in a form of user interface for a care provider to observe. Now, the needs of healthcare have fundamentally changed what is required for biomedical devices. Biomedical devices must support patient monitoring in three main scenarios: patient monitoring in the clinic, that in the residence, and that at an emergency site [4]. Therefore, biomedical devices must now support networking and must be inherently secure. Further, the opportunities for diagnostics now require not just one or a few simple sensors, but a network of sensors with its data being fused with machine learning algorithms to enable accurate inference of the subject's state.

For the last ten years, there has been a growth of wireless sensor networks (WSN), enabled by the availability of sensors that are smaller and cheaper [4][5]. A WSN is a network of computing devices that is mainly developed for surveillance and monitoring [6]. Although WSN was initially meant for large scale systems, it has been applied for human body monitoring in recent years. As a result, the body area networks (BAN) have emerged as a new generation of WSN that consists of wearable devices for health monitoring.

The IEEE 802.15 defines the BAN as the network of "low power devices operating on, in or around the human body (but not limited to humans) to serve a variety of applications including medical, consumer electronics / personal entertainment and other" [7]. The BAN may consist of low-cost, low-power, and non-invasive wearable health or activity monitoring sensors to enable inexpensive monitoring in any scenario. These sensors include electrocardiogram (ECG) sensors, peripheral capillary oxygen saturation (SpO2) sensors, motion sensors, and acoustic sensors.

A major challenge of biomedical BAN system development is meeting the needs of the healthcare provider. As such, numerous clinical trials are required. If an error is found in the system during a clinical trial, the system needs to be reworked. This may involve redesigning of software architecture or replacement of hardware components such as circuit boards and sensors. Such system adjustment needs to be repeated until the clinical trials are successful and the system's performance is robust. The system adjustments increase the development time and cost.

To address the system development challenge and the requirements, this thesis introduces an IoT BAN platform for medical devices that will provide rapid development capability with the assurance of security, networking, and the ability to host computationally intensive processes that are now required by medical devices. The developed platform includes an IoT computer-on-module device on each sensor node that includes GPIO, I2C, UART, and SPI interfaces, integrated WiFi and Bluetooth Low Energy module, and a dual-core CPU that runs embedded Linux, enabling intensive computation hosted on the system. This thesis describes the interface development, sensors, networking, and security. Furthermore, it discusses computation method for machine learning-based sensor fusion. The platform is capable of both training a classifier based on an artificial neural network algorithm and utilizing the trained classifier to monitor patients.

In addition to medical applications, the platform was distributed to freshman, senior, and graduate courses for education missions. The platform was used in various projects including IoT robot coordination system, smart wearables for workplace performance advancement, and athletic performance advancement system. These courses were successful with exceptional assessment records.

CHAPTER 2

System Overview

This chapter provides a system design overview of the BAN platform. In addition, it introduces the Intel Edison, an IoT computer-on-module device used for the BAN platform development. The device is a small module with only one Hirose connector, which requires expansion boards in order to have rapid development capability. This chapter describes the selected expansion boards for the Intel Edison.

2.1 Body Area Network Platform

Figure 2.1 illustrates the proposed BAN platform base model. It consists of seven wearable sensor nodes on the chest, wrists, upper legs, and ankles. WiFi is chosen for the network connection within the system and to routers to access the Internet. The chest node is the center node in the network. Each sensor node includes a WiFi module, enabling the WiFi connection between the chest node and the other nodes. In addition to the node-to-node connection, the chest node is capable of connecting to routers to communicate with healthcare providers via the Internet.

As a sensor node, the chest node includes an electrocardiogram sensor to monitor the heart activity, a microphone to monitor voice activity, and a 9DOF IMU to track the torso motion. The wrist nodes include a 9DOF IMU to track the hand motion and a pulse sensor to measure heartbeats. Only one of the wrist nodes may have a pulse sensor since having one on each wrist node gives redundant data. The upper leg and the ankle nodes monitor the legs' motion with 9DOF IMUs. These sensors are described in chapter 4 in more detail.

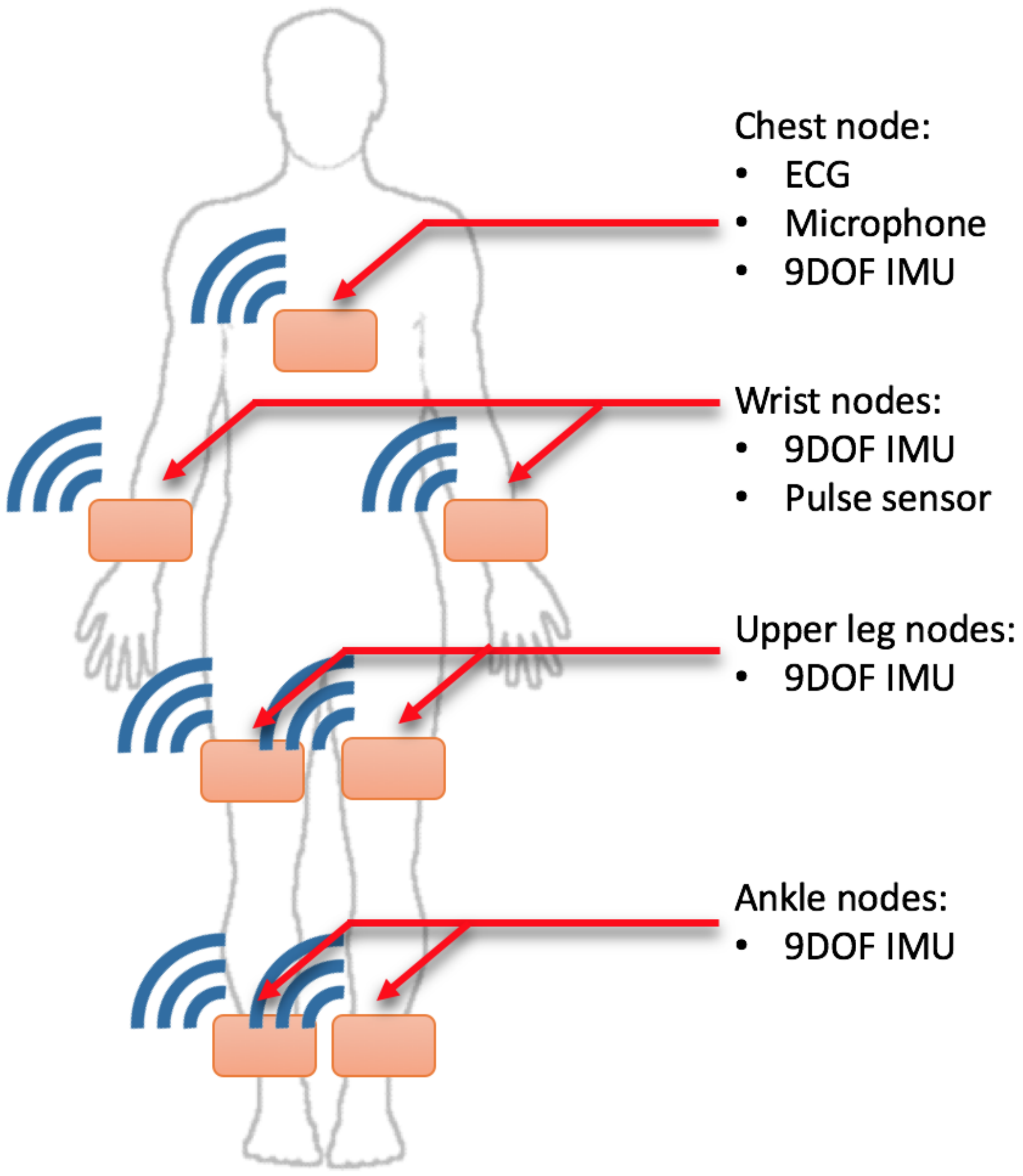


Figure 2.1: Body Area Network Platform

The proposed BAN platform is best described by the following characteristics:

- Highly customizable and expandable
- Capable of on-board intensive computation and data processing
- Remotely accessible for software development even after deployment

As shown in Figure 2.2, each sensor node includes GPIO, ADC, I2C, SPI, UART, and USB to interface with many sensors. These interfaces are described more in detail in chapter 3. Due to the availability of the interfaces, it is very easy to add more sensors to the system. As such, the platform is highly customizable and expandable. For instance, if an SpO2 sensor with a UART interface is to be added to a wrist node, it can be easily integrated into the system.

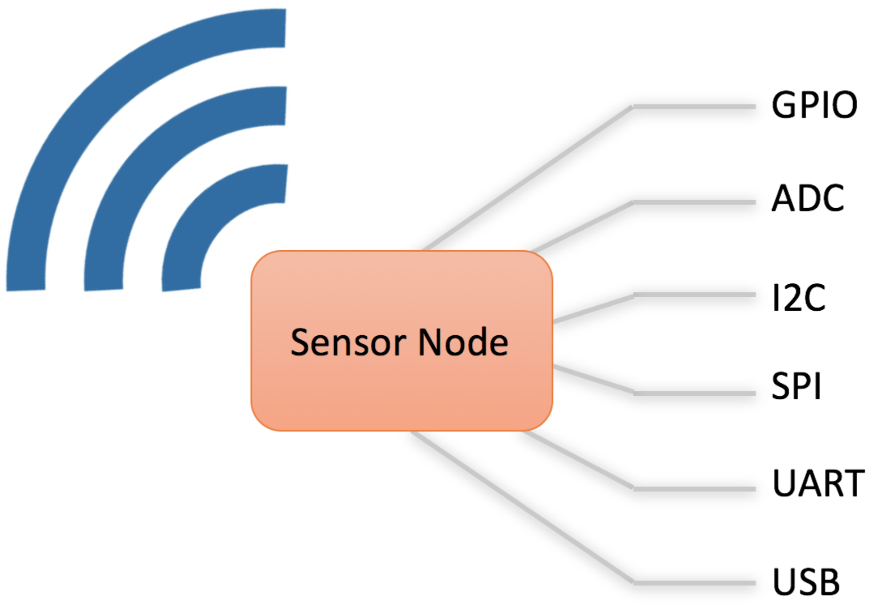


Figure 2.2: Sensor Node

Many other proposed BAN platforms [8][9][10][11] use a smartphone or a PDA as their personal server and central processing device, which integrates and processes the sensor data and then communicates with the healthcare providers through the Internet. The BAN platform proposed in this thesis uses the chest sensor node instead as the personal server and central processing device. It is important to note that every sensor node is also equally capable of data processing. For instance, each node can compute the orientation based on the 9DOF IMU sensor data and send the orientation data to the chest node rather than the raw sensor data.

Each sensor node features a CPU that runs Linux. The Linux OS on each node hosts a secure shell (SSH) server, which allows remote access. The developers can deploy the platform even when the software is not complete. After deployment, the developers can still access each node of the platform, collect and analyze data, and adjust the software.

2.2 Intel Edison Overview

The Intel Edison, shown in Figure 2.3, is a small compute module that is the size of a postage stamp. The exact dimension is 35.5 mm $\times$ 25.0 mm $\times$ 3.9 mm (1.4 in $\times$ 1.0 in $\times$ 0.15 in)

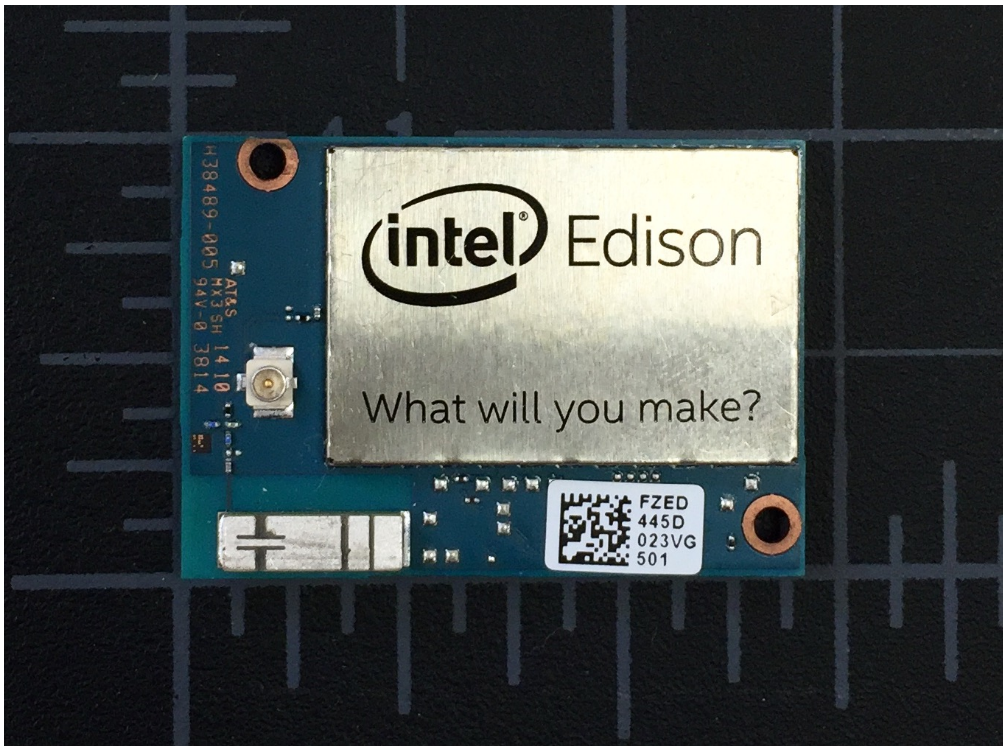


Figure 2.3: Intel Edison

[12]. It features a 22nm Intel System-on-Chip (SoC), that includes a dual-core Intel Atom processor (CPU) and a 32-bit Intel Quark microcontroller unit (MCU) [12]. The featured CPU runs at a fixed clock frequency of 500 MHz and the MCU runs at 100 MHz. The official

OS that runs on the CPU is Yocto Linux. It is also possible to substitute Yocto Linux with Ubilinux, an embedded Linux distribution based on Debian [13]. The MCU runs Viper, a Wind River RTOS [1]. The SoC architecture is shown in Figure 2.4.

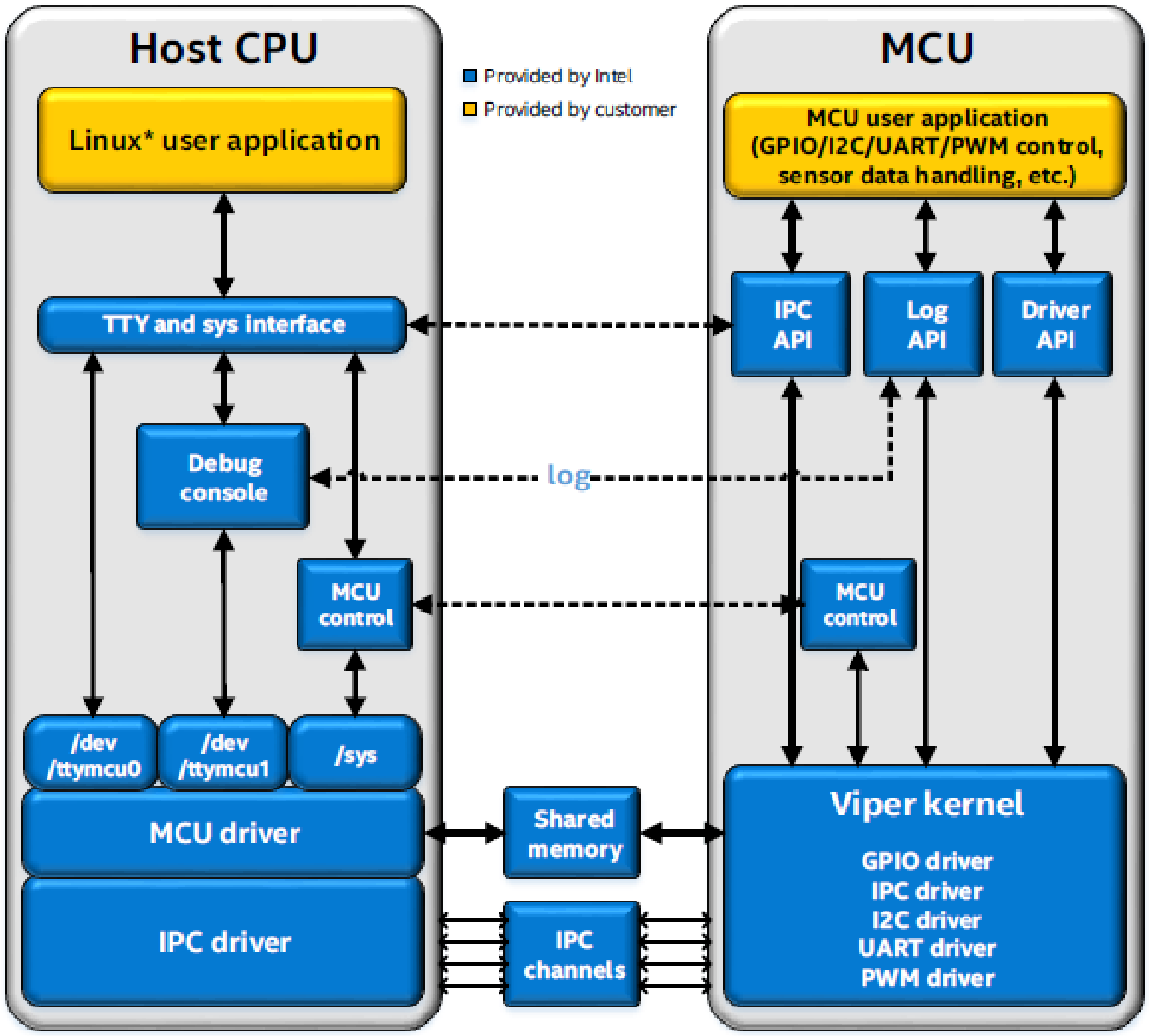


Figure 2.4: Intel Edison CPU and MCU, adopted from [1]

In addition to the SoC, the Intel Edison features other major components. It includes 1 GB LPDDR3 RAM and 4GB eMMC flash storage [12]. One of the most powerful features the Intel Edison provides is the integrated WiFi module, which supports 802.11a, 802.11b, 802.11g, and 802.11n [12]. Another powerful feature is the integrated Bluetooth Low-Energy module.

The Intel Edison has forty multiplexed GPIO interfaces that can be configured as one SD card controller, two UART controllers, two I2C controllers, one SPI controller, one inter-IC sound (I2S) controller, pulse width modulation (PWM), and USB 2.0 [12].

2.3 Available Expansion Options

The Intel Edison features a Hirose DF40 connector, shown in Figure 2.5, which allows connection to an expansion board for rapid development. There are two official expansion boards and other boards supplied by third party manufacturers. The official expansion boards are the Arduino-compatible breakout board and the mini breakout board, shown in Figure 2.6.

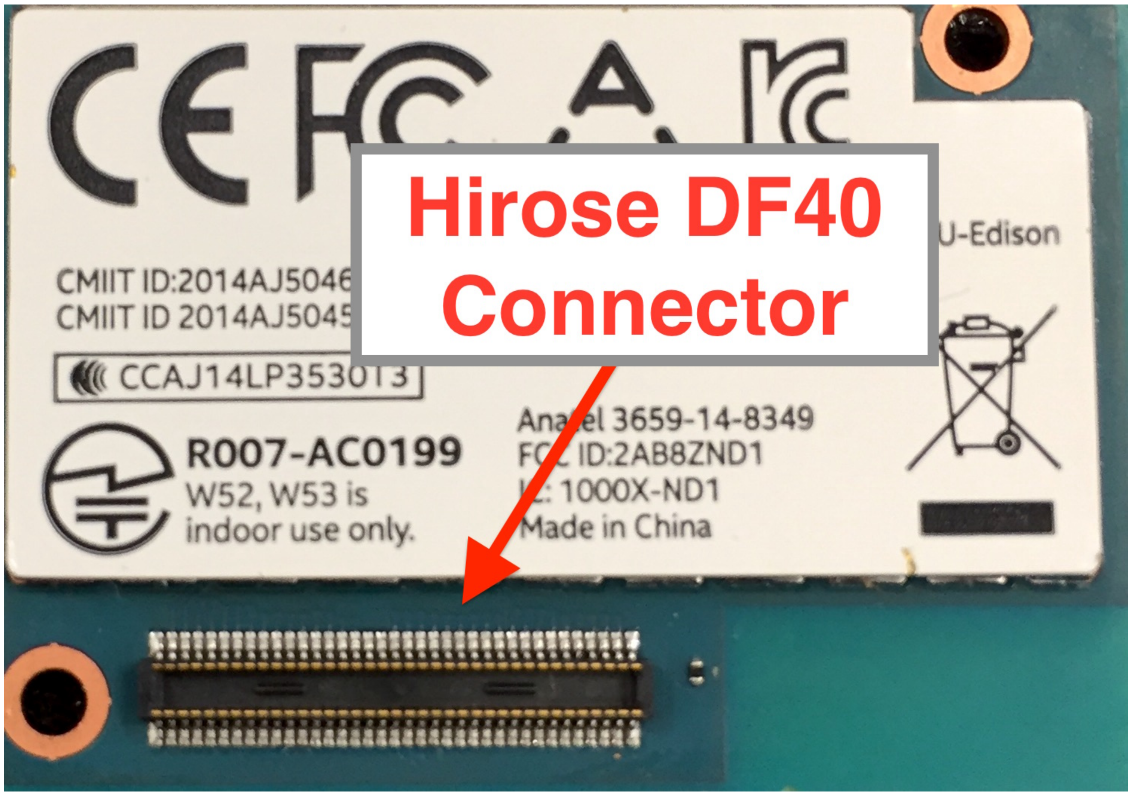


Figure 2.5: Hirose DF40 Connector

The platform development presented on this thesis involves two expansion options: the

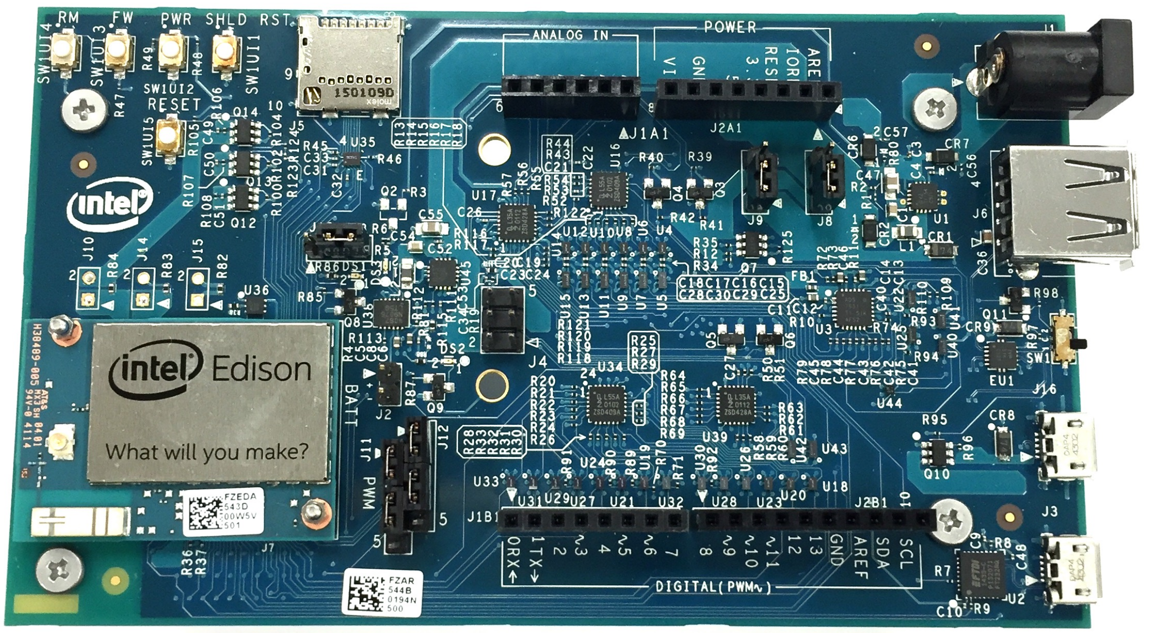
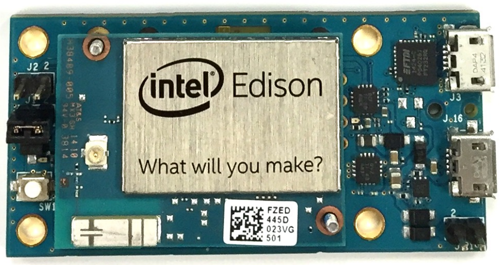



Figure 2.6: An Arduino breakout (top) and a mini breakout (bottom)

Arduino breakout and the SparkFun expansion blocks. The Arduino breakout board is compatible with Arduino Uno boards and requires very little efforts on the hardware development. Despite its rapid development capability, the Arduino breakout board is about the size of a hand and hence not suitable for wearable devices. Although the Arduino breakout is not used in the BAN platform, it was used for the prototype development due to its convenient ready-for-development interfaces. It includes twenty digital input/output pins, some of which can be configured as TX/RX pins for the UART interface, SPI pins, and PWM pins. It also features an on-board TI ADS7951ADC, which provides six analog input pins on the board [2]. The board includes two micro USB ports and a USB 2.0 Type-A port. The USB 2.0 port and one of the micro USB ports are connected to a USB OTG interface, which allows USB peripherals. The other micro USB port is connected to an FTDI chip, which enables debugging via serial console.

The Intel Edison's small size enables wearable device development. SparkFun manufactures very compact breakout boards, lithium polymer (LiPo) batteries, sensors, and actuators for the Intel Edison. Figure 2.7 shows a few of these SparkFun boards. These boards are

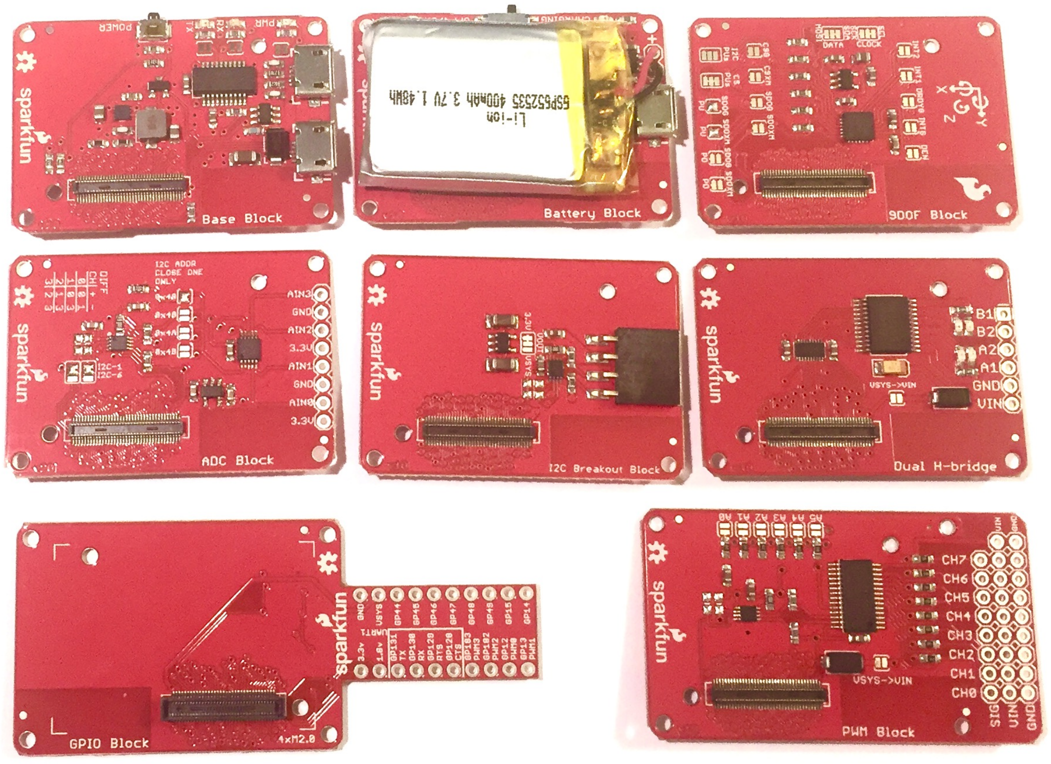


Figure 2.7: SparkFun boards

only slightly larger than the Intel Edison module and stackable for easy expansion as shown in Figure 2.8. The boards used in the BAN platform base model are the following:

- Base block

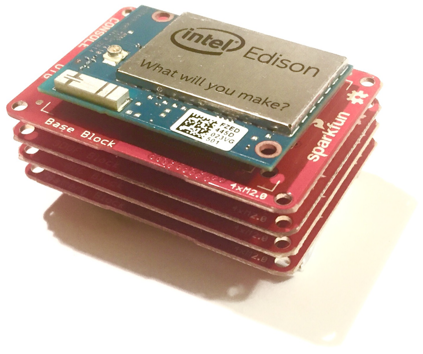


Figure 2.8: Stacked SparkFun boards with Intel Edison Module

  - Features USB OTG and FTDI

- GPIO block
  - Provides access to GPIO, PWM, and UART pins
  - Includes TXB0108 voltage level shifters

- ADC block
  - Features ADS1015 ADC
  - Communicates with Intel Edison via I2C

- 9DOF block
  - Features LSM9DS0 9DOF IMU
  - Communicates with Intel Edison via I2C

- Battery block
  - Features 400mAh LiPo battery and charger

CHAPTER 3

Interfaces

This chapter describes how to access the BAN platform's external interfaces: GPIO, I2C, SPI, UART, USB, and ADC. With the Arduino-compatible breakout board, these interfaces are easily accessed by writing Arduino sketches. However, only one sketch can be uploaded to an Edison and it is overwritten when a new sketch is uploaded. Instead, a program can be written, saved in the Edison's internal memory, and executed whenever it is needed. The interface access on the Edison is made easier with a library called MRAA. MRAA is "a C/C++ library with bindings to JavaScript and Python to interface with the IO on Galileo, Edison, and other platforms, with a structured and sane API" for GPIO, I2C, ADC, PWM, SPI, and UART [14]. This library allows developers to control low-level communication protocol by high-level language. The Yocto Embedded Linux image includes MRAA library by default. It is also possible to install the MRAA library on the Ubilinux.

Although the MRAA library provides a convenient API, it is fully compatible only when the Intel Edison is connected to an Arduino breakout board. For instance, the ADC API works for the Arduino breakout's on-board ADC module. Another problem is that the MRAA pin numbers directly map to the Arduino pin numbers rather than the GPIO numbers. Thus, direct access to the Intel Edison's GPIO interfaces via MRAA's API may not be suggested. Instead, the Linux GPIO interface, /sys/class/gpio, can be utilized. If an ADC module other than the Arduino breakout's on-board ADC is used, the ADC module must be accessed via I2C, SPI, or UART. Each section of this chapter describes different methods to access the platform's external interfaces.

3.1 General Purpose Input Output (GPIO)

The platform development involves the Arduino breakout for prototyping and the SparkFun board for actual system implementation. Since the MRAA library's API for the Intel Edison's GPIO access is not compatible without the Arduino breakout, another library to access the GPIO is developed. The first part of this section describes the Arduino breakouts GPIO interface and the MRAA's GPIO API. The second part describes the Linux interface for GPIO and the development of a C library that abstracts the Linux interface.

3.1.1 Arduino Breakout and MRAA API

The Intel Edison's GPIOs are mapped to the digital pins on the Arduino breakout as shown in Figure 3.1. The MRAA library includes functions to initialize the digital pins, set the

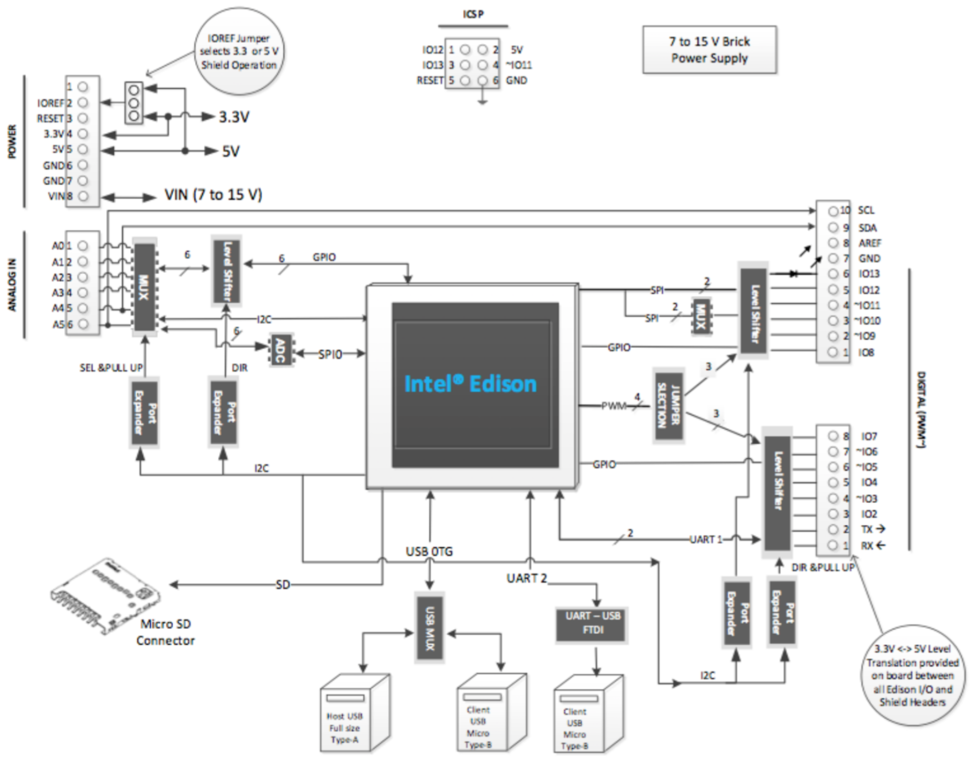
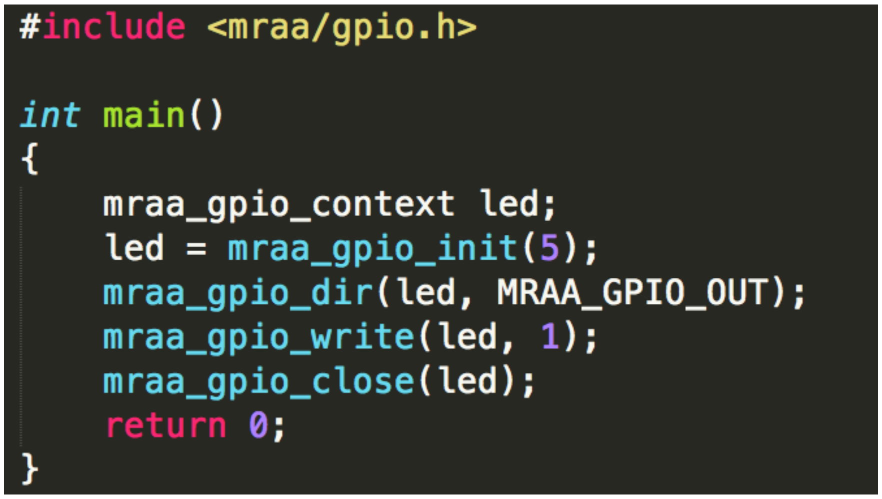


Figure 3.1: Intel Edison for Arduino Block Diagram, adopted from [2]

direction, and read from or write to the pins. An example scenario to demonstrate the

API is given as follows: Pin 5 is to be configured to send digital high signal. First, a variable with mraa_gpio_context type must be declared. Then, mraa_init_gpio function must be called with an integer 5 as its argument to initialize the digital pin 5 as a GPIO pin. The initialization function returns a GPIO context if the initialization is successful and null if it was unsuccessful. The returned GPIO context is then assigned to the variable created. The pin direction is configured with mraa_gpio_dir function, which takes two arguments: a GPIO context and a macro indicating the pin direction. In this scenario, the function is called with the GPIO context variable and MRAA_GPIO_OUT as the arguments. Sending a digital signal to the pin is done in a similar fashion. A function, mraa_gpio_write, takes two arguments: a GPIO context and an integer with value 0 or 1. Finally, the GPIO pin is closed with mraa_gpio_close function. Figure 3.2 shows this example in C code.



```c
#include <mraa/gpio.h>

int main()
{
    mraa_gpio_context led;
    led = mraa_gpio_init(5);
    mraa_gpio_dir(led, MRAA_GPIO_OUT);
    mraa_gpio_write(led, 1);
    mraa_gpio_close(led);
    return 0;
}
```

Figure 3.2: MRAA Library GPIO API Demo

3.1.2 SparkFun GPIO breakout and Linux GPIO interface

SparkFun GPIO breakout, shown in Figure 3.3, is a compact alternative to the Arduino breakout for the GPIO access. It has sixteen GPIO pins that are connected to bidirectional level shifters for the Intel Edison's GPIO voltage of 1.8 V. The breakout also includes pins for the Intel Edison's internal power supply and ground. These power pins are useful when powering on sensors or adding pull-up resistors for GPIO inputs.

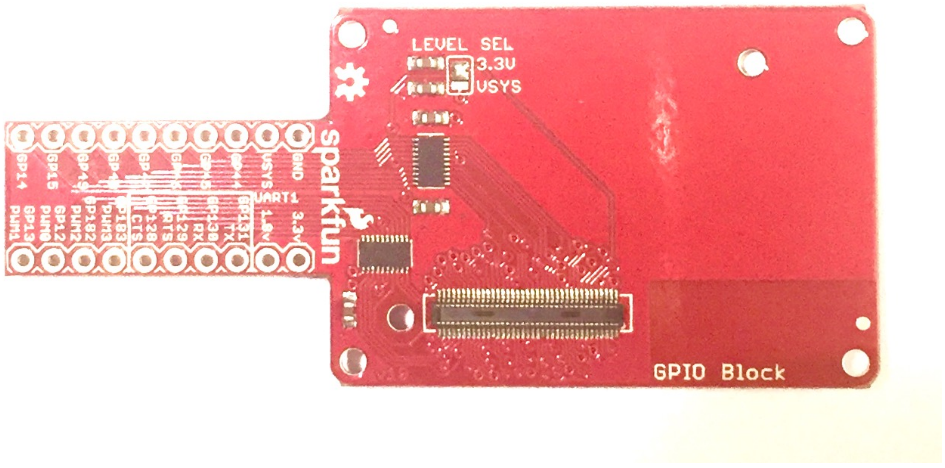


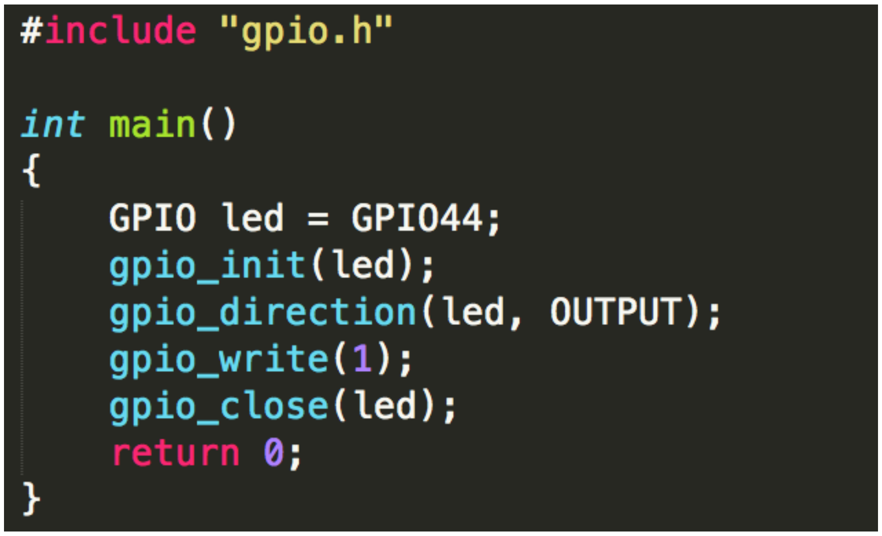
Figure 3.3: SparkFun GPIO Block for Intel Edison

The embedded Linux provides a userspace interface for GPIO. On the Linux, the GPIOs are accessed from /sys/class/gpio. The procedure to send digital signals with the Linux GPIO interface is similar to MRAA's. First, the GPIO must be made available before accessing it. This is done by writing the GPIO number to /sys/class/gpio/export. If the GPIO number is 44, there now must be /sys/class/gpio/gpio44. Then, the GPIO's direction is set by writing "out" to /sys/class/gpio/gpio44/direction. Writing "1" to /sys/class/gpio/gpio44/ value sends digital high to the GPIO pin.

The Linux GPIO interface can be used directly to access the GPIO. However, direct use of the interface is not intuitive. Thus, a C library that abstracts the Linux GPIO interface is developed for more intuitive GPIO access. The library's API is designed to be similar to MRAA library's in order to fast port the prototype developed on the Arduino breakout. The C library provides useful functions: gpio_init, gpio_close, gpio_direction, gpio_write, and gpio_read. The gpio_init function initializes a GPIO pin for a simple digital input or output by writing the GPIO number to /sys/class/gpio/export. It takes one argument to specify the GPIO number. Since the library is designed for the SparkFun block, the available pins are limited to the listed values below:

- GPIO12, GPIO13, GPIO14, GPIO15, GPIO44, GPIO45, GPIO46, GPIO47, GPIO48, GPIO49, GPIO182, and GPIO183.

The gpio_close function closes a GPIO pin by writing the GPIO number to /sys/class/gpio/unexport. It takes one argument to specify the GPIO number. The gpio_direction function sets the direction of the GPIO pin by writing "in" or "out" to /sys/class/gpio/gpioN/direction. It takes two arguments: one to specify the pin number and the other to specify the direction. The gpio_write function writes a digital value (0 or 1) to the pin by writing "0" or "1" to /sys/class/gpio/gpioN/value. It takes two arguments: one to specify the pin number and the other to specify the digital value. This function is available only if the GPIO pin is set as an output. The gpio_read function reads the value from a GPIO pin by reading /sys/class/gpio/gpioN/value and returns the value as an integer. It takes one argument to specify the pin number. This function is available only if the GPIO pin is set as an input. Figure 3.4 shows an example, which utilizes the C library to write a digital high value to GPIO44.



```c
#include "gpio.h"

int main()
{
    GPIO led = GPIO44;
    gpio_init(led);
    gpio_direction(led, OUTPUT);
    gpio_write(1);
    gpio_close(led);
    return 0;
}
```

Figure 3.4: GPIO Library Demo

3.2 Inter-Integrated Circuit (I2C)

I2C is developed by Philips in the 1980s to minimize the number of wires. It is a protocol that allows a master integrated circuit (IC) device to communicate with multiple slave IC devices. It uses two wires: serial clock line (SCL) and serial data line (SDA). Since I2C devices are open-drain, they can only pull the signal low. Thus, there are pull-up resistors for SCL and SDA. Figure 3.5 illustrates a common I2C bus.

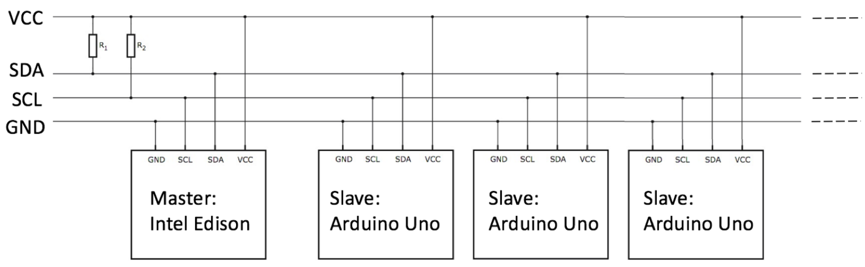


Figure 3.5: I2C

Messages are broken up into two frame types: address frame and data frame. The address frame lets the master device indicate which slave device it wants to communicate with. The data frame contains an 8-bit message to be placed on the SDA. Figure 3.6 illustrates the protocol. First, the start condition must be set up so that the master device pulls down the

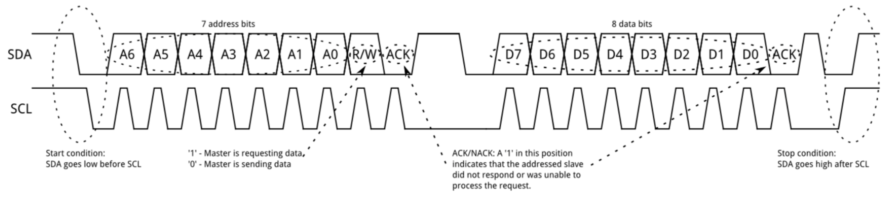

7 address bits
8 data bits
SDA
SCL
A6 A5 A4 A3 A2 A1 A0 R/W ACK
D7 D6 D5 D4 D3 D2 D1 D0 ACK
Start condition: SDA goes low before SCL
'1' - Master is requesting data, '0' - Master is sending data
ACK/NACK: A '1' in this position indicates that the addressed slave did not respond or was unable to process the request.
Stop condition: SDA goes high after SCL

Figure 3.6: I2C Protocol, adopted from [3]

SDA while keeping the clock high. Then, all slave devices are notified that a transmission is about to begin. The 7-bit address is then transmitted from the most significant bit to the least significant bit in seven clock pulses. What follows after the address bits is a read or write bit. If the read or write bit is high, the master is requesting data. If it is low, the master is sending data. The receiving slave device pulls down the SDA to indicate that it has received the request. If the SDA remains high, the slave device did not receive the request. Once the request is received, the slave device sends the data. The data transmission is similar to the address transmission. Depending on whether the master device is reading or sending data, the device that controls the SDA to send an 8-bit data is the slave device or the master device. What follows after the data is an acknowledge bit, which is pulled down by the receiving device to indicate that the data is received. Once it is received, the transmission must stop. The SDA becomes high after the SCL becomes high to indicate the transmission is finished.

The Intel Edison includes two I2C controllers for I2C-1 bus and I2C-6 bus. Four pins of the Intel Edisons Hirose connector are dedicated for the I2C buses. Pin 41 and pin 43 are respectively the SCL and the SDA of I2C-1 bus while Pin 45 and pin 47 are the SCL and the SDA for I2C-6 bus [15]. The Arduino breakout maps these pins to A4, A5, SCL, and SDA pins and port expanders as shown in Figure 3.1. Pull-up resistors need to be placed between these pins and the VCC pin. As long as the signal voltage level matches the Arduino breakout's, slave devices' SCLs and SDAs can be directly connected to these pins. A sensor node in the BAN platform base model includes up to two I2C slave devices: the 9DOF IMU and the ADC. These devices are connected directly to the sensor node's Intel Edison module via the Hirose connector pins. If an I2C device cannot be directly connected via the Hirose connector to the Intel Edison modules, a SparkFun's compact I2C breakout board can be used. This breakout board is shown in Figure 3.7. Unlike the Arduino breakout, this board includes on-board 10 K ohm pull-up resistors, enabling direct connection between the female pin connector and slave devices with 3.3 V level.

The Intel Edison's I2C module supports three modes: standard mode with data rates up to 100 kbps, fast mode with data rates up to 400 kbps, and high-speed mode with data rates up to 3.4 Mbps [15]. It is important to note that the Intel Edison can only be configured as a master I2C device. Another limitation is that multiple master configuration is not supported.

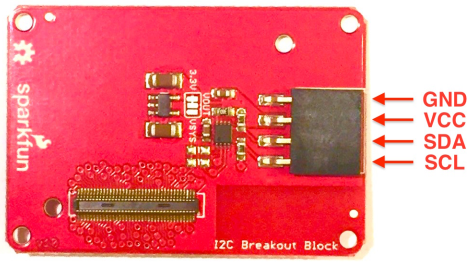


Figure 3.7: SparkFun I2C Breakout

In Linux, an I2C adapter can be used to access all I2C devices from userspace. The MRAA library abstracts the Linux's I2C dev interface to provide a more convenient API for the Intel Edison's I2C interface. The MRAA is used for both Arduino breakout and SparkFun board options. The MRAA library functions initialize the Intel Edison as a master device, configure the operation speed/mode, read data from a slave device, and send data to a slave device. Figure 3.8 shows an example of the MRAA's I2C API.

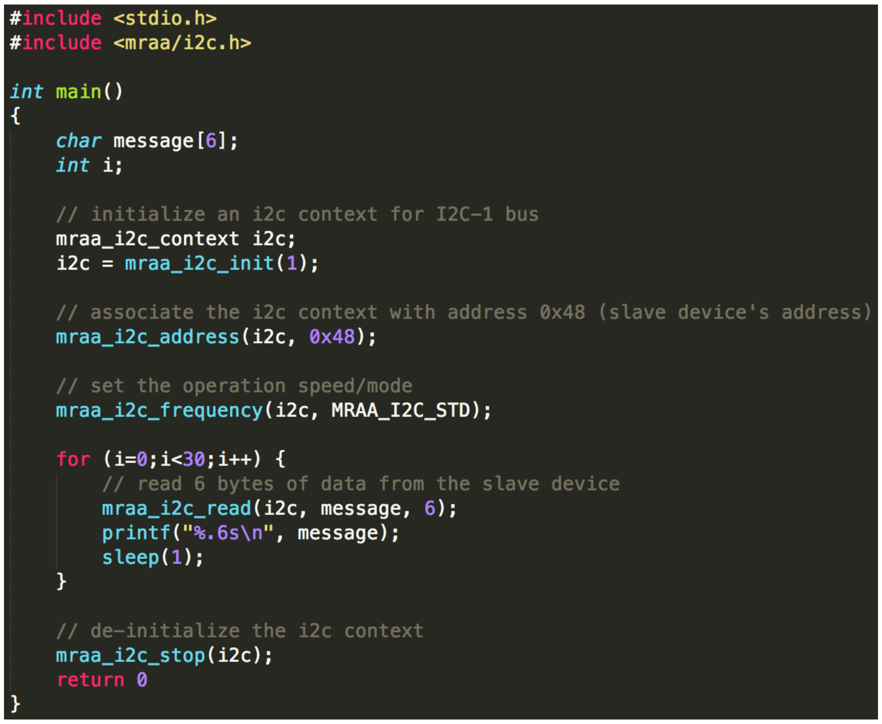

```
#include <stdio.h>
#include <mraa/i2c.h>

int main()
{
    char message[6];
    int i;

    // initialize an i2c context for I2C-1 bus
    mraa_i2c_context i2c;
    i2c = mraa_i2c_init(1);

    // associate the i2c context with address 0x48 (slave device's address)
    mraa_i2c_address(i2c, 0x48);

    // set the operation speed/mode
    mraa_i2c_frequency(i2c, MRAA_I2C_STD);

    for (i=0;i<30;i++) {
        // read 6 bytes of data from the slave device
        mraa_i2c_read(i2c, message, 6);
        printf("%.6s\n", message);
        sleep(1);
    }

    // de-initialize the i2c context
    mraa_i2c_stop(i2c);
    return 0
}
```

Figure 3.8: MRAA I2C Demo

3.3 Serial Peripheral Interface

SPI is a synchronous serial communication between one master device and one or more slave devices for a short distance. Along with I2C, SPI is primarily used in embedded systems. The motivation for the development of these buses is to reduce the number of wires. Another common communication method is parallel communication. This method can result in a bus having a lot of wires. Having many wires is not desirable for embedded systems due to their limited board space. In comparison to parallel buses, SPI operates with only four wires. Figure 3.9 illustrates a single master to a single slave SPI bus.

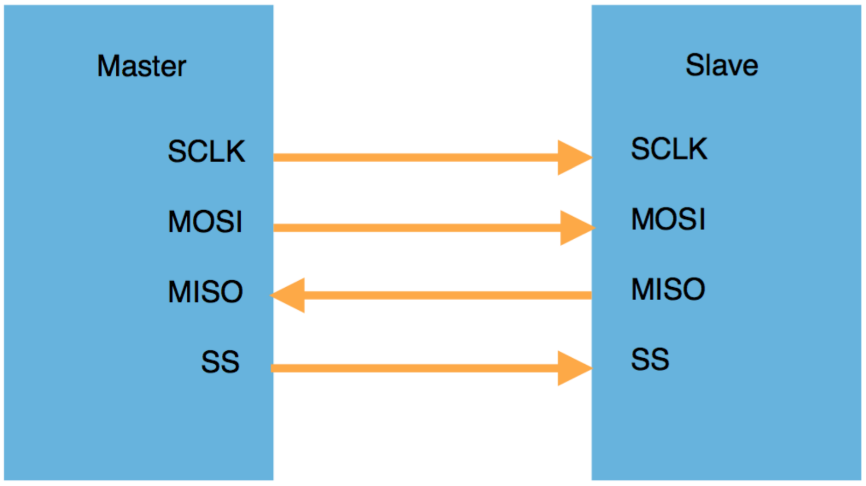


Figure 3.9: SPI Between a Single Master and a Single Slave

SCLK (serial clock) is the clock line, through which the master device sends the clock signal to its slave devices to synchronize the data communication. MOSI (master out, slave in) and MISO (master in, slave out) are the data lines. The reason why there are two data lines in SPI is to have full duplex communication. In every clock cycle, the master device sends a bit to the slave via MOSI and the slave device sends a bit to the master via MISO. SS (slave select) is the line the master device uses to select the slave device for data transmission/request by pulling down.

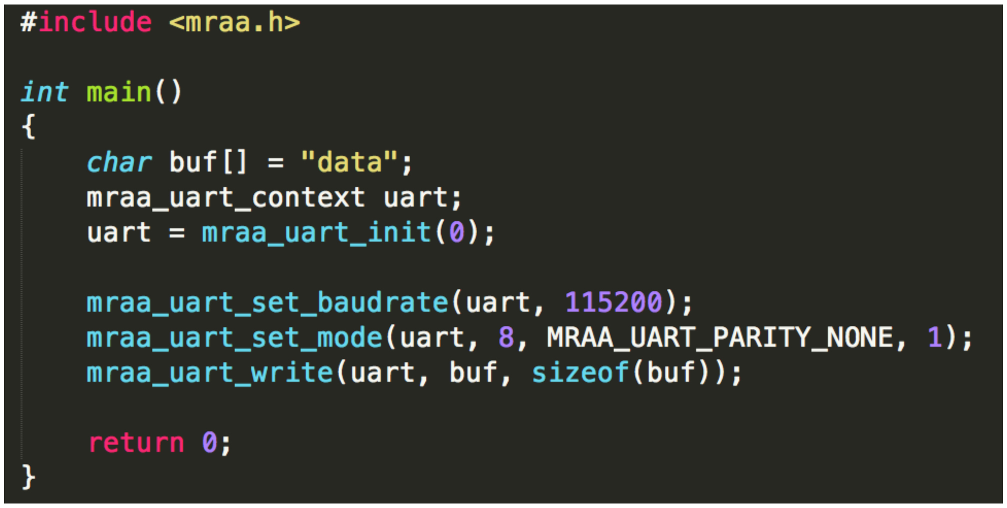
The Intel Edison's Hirose connector pins 51, 53, 55, 57, and 59 are dedicated to the SPI interface [15]. The Arduino breakout maps these pins to its digital pins for intuitive SPI access. The BAN platform does not include a compact breakout for the SPI interface. Thus, the interface must be accessed directly from the Hirose connector pins. The BAN platform base model does not include any SPI devices. Nevertheless, the SPI interface is investigated to allow the platform to expand with SPI devices.

MRAA library abstracts Linux's userspace API for SPI to provide a more intuitive API. The API includes functions to initialize an SPI context, select the SPI mode and the transmission mode, set the operating clock frequency, write data to an SPI device, and de-initialize an SPI context. The initialization will choose the default SPI mode and the transmission mode. The default SPI mode is low clock phase and low clock polarity. This mode means the idle clock state is low; data is captured on the rising edge; and data is output on the falling edge. There are three other SPI modes: low clock phase and high clock polarity, high clock phase and low clock polarity, and high clock phase and high clock polarity. These modes are denoted by MRAA as MRAA_SPI_MODE1, MRAA_SPI_MODE2, and MRAA_SPI_MODE3. The mraa_spi_mode function can be called with the SPI context and one of the SPI modes as its arguments to change the SPI mode. The default transmission mode is "transfer most significant bit first". By calling the mraa_spi_lsbmode function, the least significant bit can be transferred first. There are multiple functions to write data to an SPI device. These functions give the user an option to choose the size of the data to write: one byte (uint8_t), two bytes (uint16_t), a buffer of bytes (uint8_t), and a buffer of uint16_t.

3.4 Universal Asynchronous Receiver/Transmitter (UART)

The Intel Edison features two UART controllers for communication over serial ports. One of the controllers has RTS, CTS, RX, and TX while the other controller has only RX and TX. If the Arduino breakout is used, the controller with full flow control is connected to the breakout's on-board FTDI chip for USB debugging and the controller with only RX and TX is connected to digital pins 0 and 1.

The MRAA library provides an API for the UART controller that is connected to digital pins 0 and 1. The API includes functions to set the baud rate, the data bit size, the parity bits, and the stop bit size. A baud rate is the speed of data transmission over a serial line. Each transmitted data is sent in a frame, which includes a start bit, data bits, parity bits, and one or two stop bits. The data bits are the data to be transmitted. The parity is a simple error checking method for the data. Once these settings are done, MRAA functions can be used to read from or write to the serial line. Figure 3.10 shows a C code example that demonstrates the MRAA UART API.



```c
#include <mraa.h>

int main()
{
    char buf[] = "data";
    mraa_uart_context uart;
    uart = mraa_uart_init(0);

    mraa_uart_set_baudrate(uart, 115200);
    mraa_uart_set_mode(uart, 8, MRAA_UART_PARITY_NONE, 1);
    mraa_uart_write(uart, buf, sizeof(buf));

    return 0;
}
```

Figure 3.10: MRAA UART Demo

When the SparkFun boards are used, the UART controllers are used in the opposite way. The UART controller with only RX and TX are used for the SparkFun base block's FTDI chip for USB debugging. The same GPIO block introduced in 3.1.2 is used for the UART. The GPIO block is connected to the UART controller with full flow control as shown in Figure 3.11. For this reason, the MRAA library cannot be used since it is designed to interact with the other controller. These UART controllers can be accessed via Linux serial ports. /dev/ttyMFD2 is connected to the UART controller with full flow control and /dev/ttyMFD1 is connected to the controller with only RX and TX. Instead of MRAA, a Python package called PySerial is used for the BAN platform's UART interface. Using PySerial API, /dev/ttyMFD2 serial port is opened and data framing setting is done. Then,

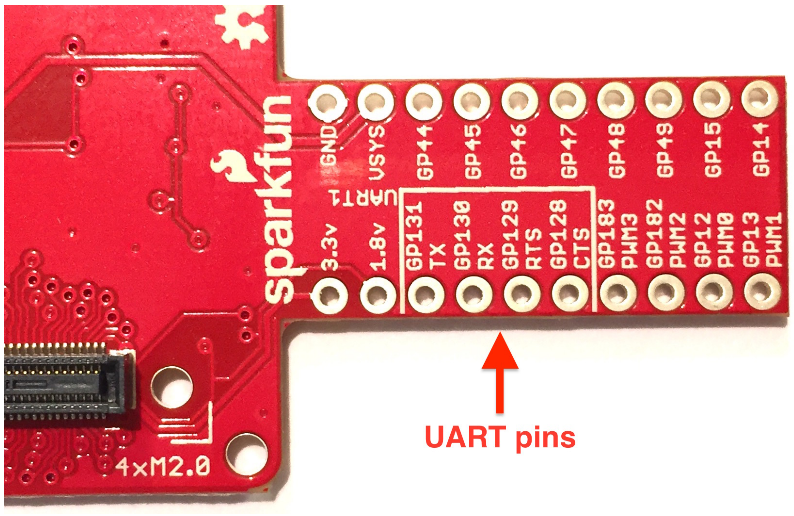
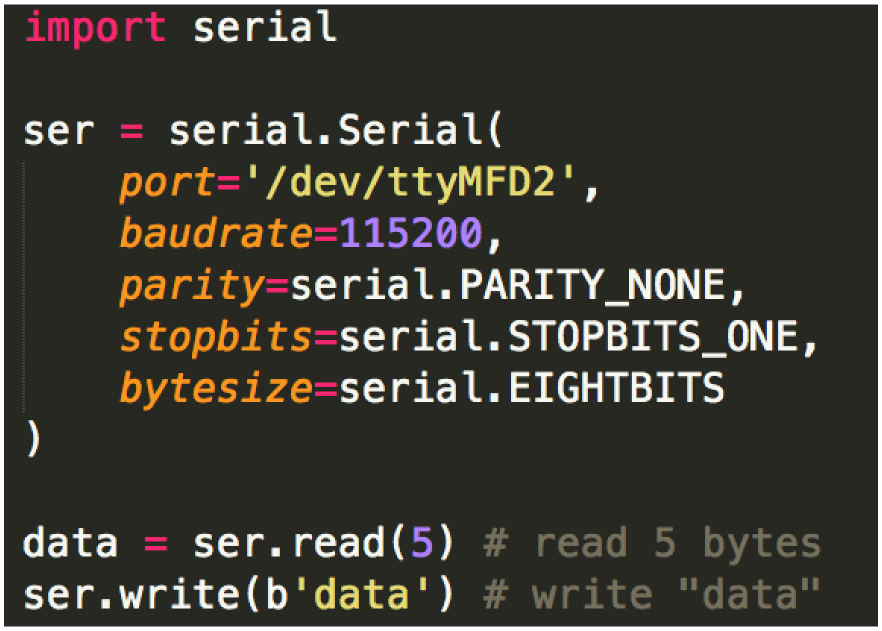


Figure 3.11: UART pins on GPIO Block

the API's read function is used to read from /dev/ttyMFD2 or the write function is used to write data to the serial port. Figure 3.12 shows a Python example that demonstrates PySerial.



```python
import serial

ser = serial.Serial(
    port='/dev/ttyMFD2',
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS
)

data = ser.read(5) # read 5 bytes
ser.write(b'data') # write "data"
```

Figure 3.12: PySerial Demo

3.5 Universal Serial Bus (USB)

The Intel Edison features a USB 2.0 OTG interface. The Hirose connector pins 3, 16, 18, and 20 are USB OTG ID pin, USB D+, USB D-, and USB VBUS respectively [15]. The Arduino breakout connects these pins to a micro USB port and a USB type-A port. There is a switch that selects which port to use. These are shown in Figure 3.13. SparkFun base block has only the micro USB port option. However, an adapter can be used, as shown in Figure 3.14, to connect devices with USB type-A connectors.

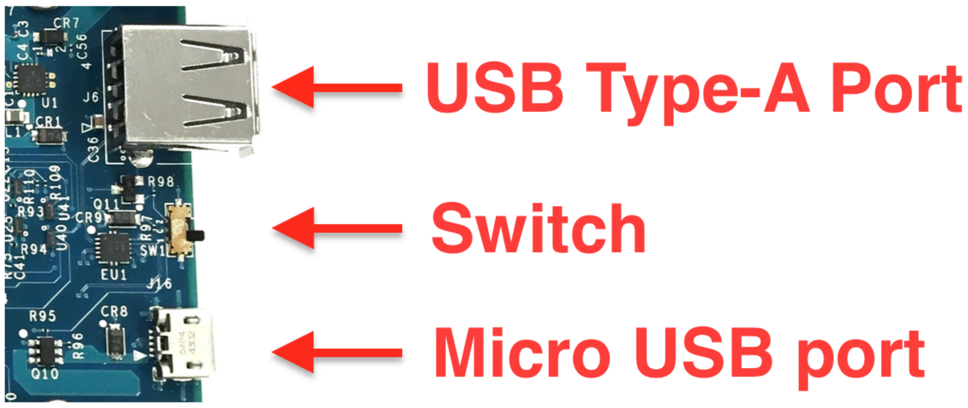


Figure 3.13: Arduino Breakout USB Ports

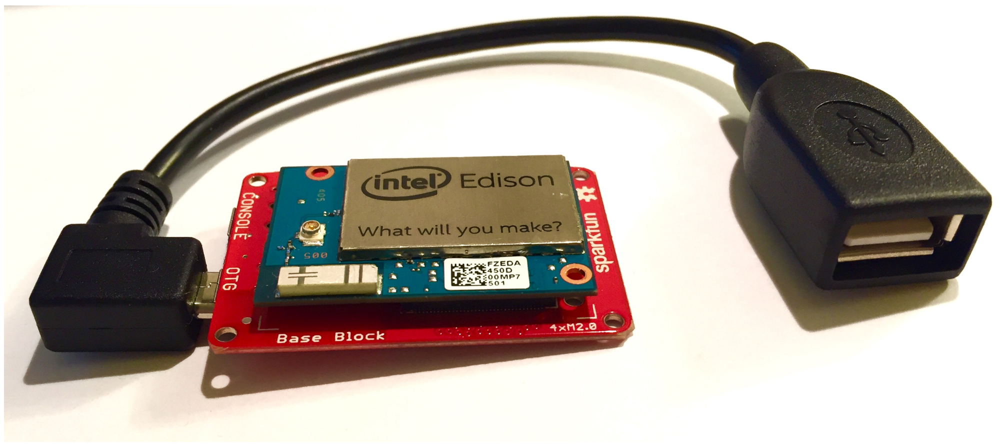
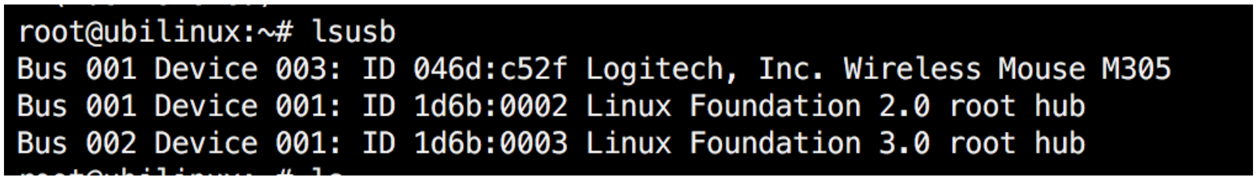


Figure 3.14: SparkFun Base Block USB Port

The USB interface accepts USB peripherals that are Linux-compatible. These include sound cards, mice, keyboards, and webcams. The Linux command, lsusb, can be used to display the accepted USB devices. Figure 3.15 shows an output of lsusb command entered while a USB mouse is connected. If the connected device is not listed, it may not be



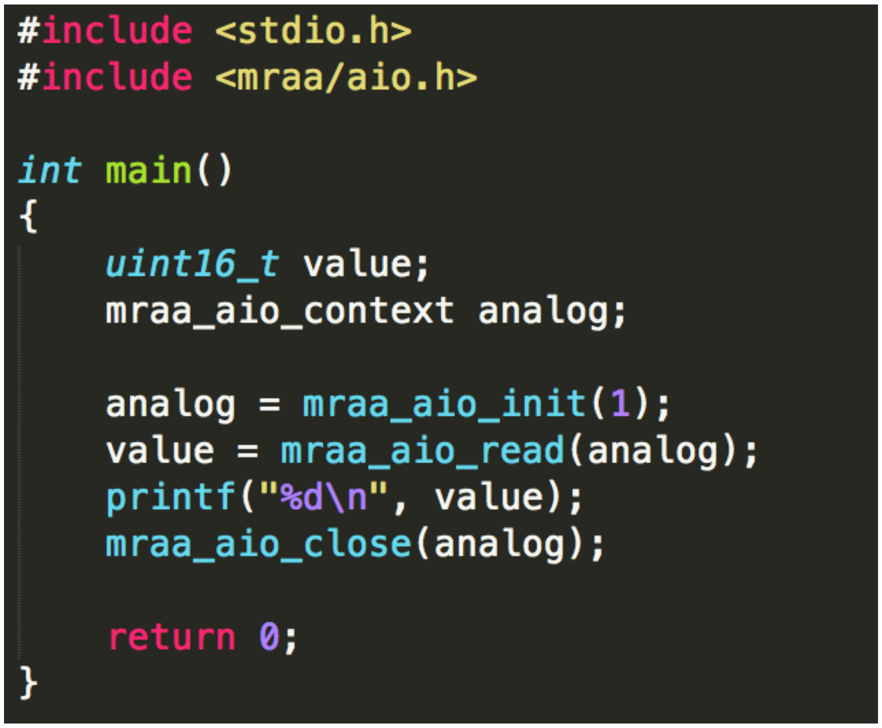
```
root@ubilinux:~# lsusb
Bus 001 Device 003: ID 046d:c52f Logitech, Inc. Wireless Mouse M305
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

Figure 3.15: lsusb Command Output

supported by Linux. Thus, it is important to ensure that the USB devices to use for the BAN platform are Linux-compatible. Not all device have drivers for Linux. Even if they have Linux drivers, they may not support some Linux distribution. For instance, a USB data acquisition device from National Instrument called USB-6009 has a Linux driver with dependencies that are RPM packages. Therefore, only the Linux distributions with RPM support, such as RedHat, SUSE, CentOS, and Fedora, can properly use the driver installer. Engineers may even need to write their own drivers if there is no supported driver.

3.6 Analog-to-Digital Converter (ADC)

An ADC is a device that converts a quantity given as voltage to a digital number. In order to read analog signals, digital systems require ADCs. The Intel Edison module does not include an integrated ADC to accept analog signals. Instead, an ADC device must be connected to the Intel Edison via another interface. The Arduino breakout features an on-board ADS7951 ADC from Texas Instrument. The on-board ADC communicates with the Intel Edison via the Linux Industrial I/O subsystem. The MRAA library provides an API for analog inputs that abstracts the communication between the Intel Edison module and the on-board ADC. Similar to other MRAA APIs, the analog input API provides functions to initialize an analog input context, read values from the input, and de-initialize the context. Figure 3.16 shows a C code example that demonstrates the API.



```c
#include <stdio.h>
#include <mraa/aio.h>

int main()
{
    uint16_t value;
    mraa_aio_context analog;

    analog = mraa_aio_init(1);
    value = mraa_aio_read(analog);
    printf("%d\n", value);
    mraa_aio_close(analog);

    return 0;
}
```

Figure 3.16: MRAA Analog Input Demo

The BAN platform uses a compact alternative. SparkFun manufactures an ADC board, shown in Figure 3.17, for the Intel Edison. It features an ADS1015 ADC. Since the MRAA

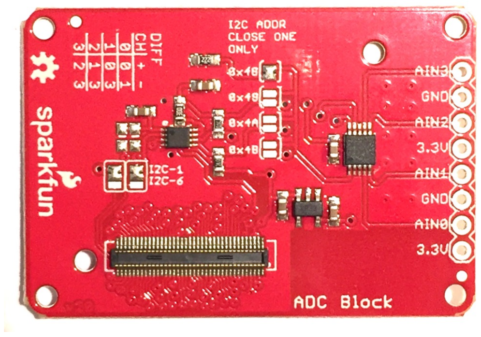
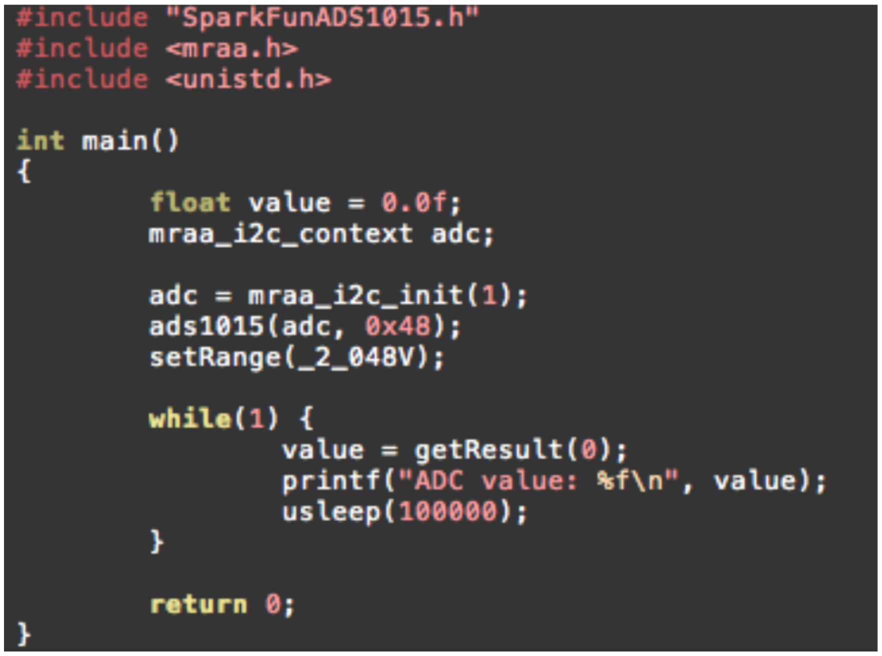


Figure 3.17: SparkFun ADC Board

library is built for ADS7951, the analog input API does not work with the SparkFun ADC board. In order to enable rapid development capability on the ADC, a C library is developed. The C library is based on SparkFun's C++ library and includes functions that abstracts the I2C communication between the ADC and the Intel Edison. The library still depends on the MRAA library for its I2C API. An MRAA I2C context must be declared and initialized. Then, the C library's functions make the Intel Edison to communicate with the ADC via an I2C bus to set the ADC's voltage range and to read data from the ADC. Figure 3.18 shows a C example that demonstrates the C library.



```
#include "SparkFunADS1015.h"
#include <mraa.h>
#include <unistd.h>

int main()
{
        float value = 0.0f;
        mraa_i2c_context adc;

        adc = mraa_i2c_init(1);
        ads1015(adc, 0x48);
        setRange(_2_048V);

        while(1) {
                value = getResult(0);
                printf("ADC value: %f\n", value);
                usleep(100000);
        }

        return 0;
}
```

Figure 3.18: ADC Library Demo

CHAPTER 4

Sensors

The BAN platform base model includes 9DOF IMU motion sensors, an ECG sensor, a microphone, and a heart rate sensor. This chapter describes the sensors and their interfaces to the BAN platform.

4.1 9 Degree-of-Freedom Inertial Measurement Unit (9DOF IMU)

The BAN platform provides capability to develop systems including motion tracking and posture monitoring. Such capability is enabled by the 9DOF IMUs at the sensor nodes. A 9DOF IMU is an electronic device that measures the acceleration in three axes, the angular rotation in three axes, and the magnetic field in three axes. Each measurement type is in three degrees of freedom because it includes data in three different axes. Thus, it has nine degrees of freedom in total. The 9DOF IMUs have accelerometers, gyroscopes, and magnetometers.

The device used in the platform is the SparkFun 9 Degrees-of-Freedom block, shown in Figure 4.1. The board includes a Hirose connector, which enables simple solderless connection to an Intel Edison module. The device has an on-board LSM9DS0 chip, which communicates with the Intel Edison through the I2C bus [16]. Each sensor on the LSM9DS0 chip is configurable to a wide range of sensitivity values and sample rates. The sensitivity describes the gain of the sensor [17]. The accelerometer's sensitivity value can be set to ±2, ±4, ±6, ±8, or ±16 g. The sensitivity value is explained as follows. The sensor values are measured when its axis of interest is pointing to the center of the Earth and when it

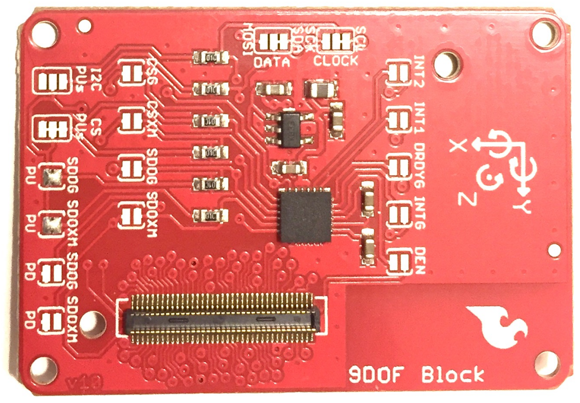


Figure 4.1: SparkFun 9 Degrees of Freedom Block

is pointing to the opposite direction. The difference between these values is divided by 2. The resulting value is the sensitivity value. The accelerometer's supported sample rates range from 3.125 Hz to 1600 Hz. The gyroscope's sensitivity value can be set to 245, 500, or 2000 degrees per second. It supports sample rates ranging from 95 Hz to 760 Hz. The magnetometer's sensitivity value can be set to 2, 4, 8, or 12 gauss. It supports sample rates ranging from 3.125 Hz to 100 Hz.

Direct interaction with the sensor may not be intuitive. In order to enable the sensors, set up the sensitivities and the sample rates, and read the sensor data, the Intel Edison needs to write to or read from the LSM9DS0 chip's registers. For instance, setting up the sample rate for the accelerometer is done via writing values to the four most significant bits of the LSM9DS0 chip's CTRL_REG1_XM register. In order to enable fast development with the 9DOF IMU device, a C library is developed. The library is based on SparkFun's C++ library for the device. The C library utilizes the MRAA library to interact with the sensor via MRAA's I2C API. It includes functions that abstracts the low-level I2C communication to configure sensors and read data. With these functions, the library provides a very simple and intuitive API. A programming guide document is created to describe the API and guide users.

4.2 Electrocardiogram (ECG)

The BAN platform is capable of monitoring the patient's heart activity, enabled by a compact 3-lead ECG module. An ECG is the heart's electrical activity data collected with electrodes placed on the surface of skin [18]. An ECG delivers a lot of information about the heartbeat, size of the heart chambers, the heart rate, muscle tissue damage, drug effects, and many others.

The BAN platform uses a compact 3-lead ECG module that is about the size of a credit card. The ECG module is EG01000 from Medlab shown in Figure 4.2. The module connects

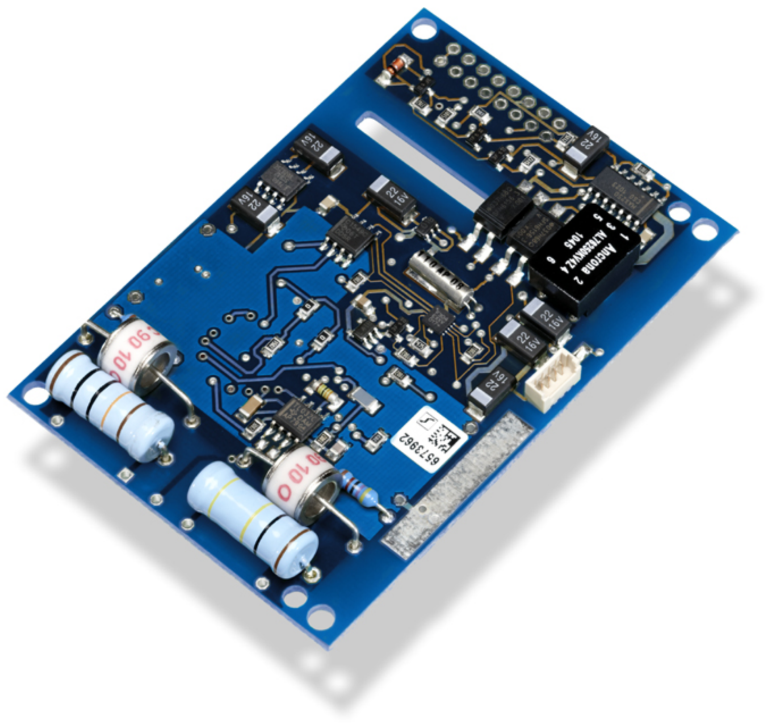


Figure 4.2: ECG Module

to the Intel Edison using the UART interface at 9600 baud rate with 8-bit data, one stop bit, and no parity. The serial connection is bidirectional because the ECG module continuously streams data and receives commands. The data stream includes wave sample points, pulse values, “lead off” notification, and data type markers, which separate the different types of data.

The data can be transmitted in three different rates: 300 Hz, 100 Hz, and 50 Hz. The transmission rate is selected by sending a command to the ECG module. “S0” command chooses 300 Hz. “S1” command chooses 100 Hz. “S2” command chooses 50 Hz. There are other available commands. “5” and “6” commands turn on a 50 Hz notch filter and a 60 Hz notch filter respectively. “M” command switches to simulation mode. “N” command switches back to the normal mode, in which real data is collected from three ECG leads. In order to test the communication between the ECG module and the Intel Edison, the “M” command is sent in order to generate simulated wave sample data, which is recorded in a csv file on the Intel Edison. The data in the csv file is plotted as shown in Figure 4.3.

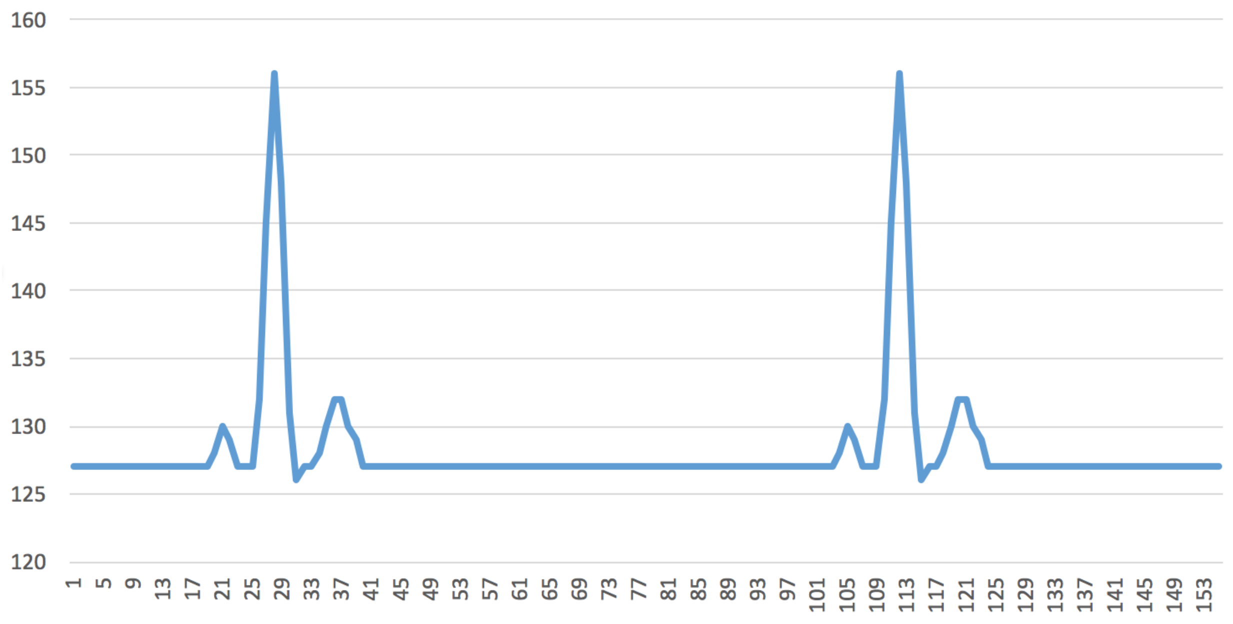


Figure 4.3: Simulated ECG Trace

4.3 Microphone

The BAN platform includes a microphone that is connected via the platform's USB sound card. With a microphone, the BAN platform is capable of hosting features like voice activity detection, speech recognition, and speaker recognition. In a capstone design course, a group of students were provided with the BAN platform and have developed a behavioral monitoring system that uses a microphone to monitor the user's interaction with others. Another group has been working on a mental wellness IoT system, which detects the user's sighs to determine the user's stress level.

The sensor node with the microphone is shown in Figure 4.4. The BAN platform features a clip-on omnidirectional condenser microphone. The users can simply clip it on their clothes near their mouths. The microphone initially features a 3.5 mm TRRS (tip, ring, ring, sleeve) connector, which is not compatible with the USB sound card used in the BAN platform. Therefore, the connector is replaced with a TRS (tip, ring, sleeve) connector as shown in Figure 4.4.

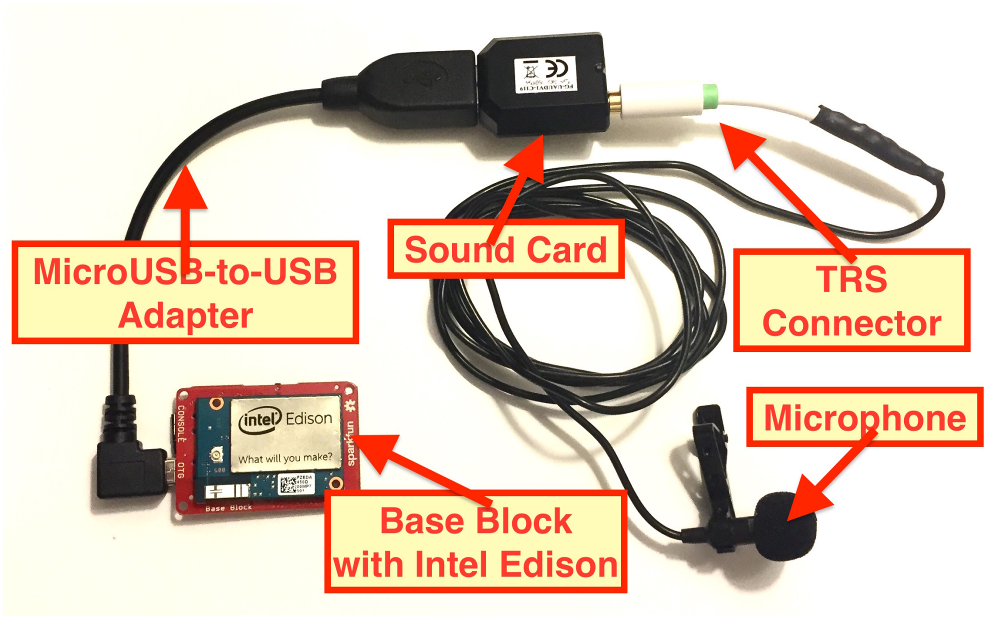
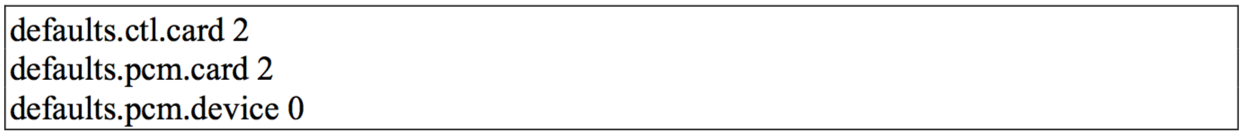


Figure 4.4: Sensor Node with Microphone

A SparkFun base block is required in order to have a microphone set up. The base block features a USB OTG via a micro USB port. A micro USB-to-USB adapter is used, as the sound card has a USB male connector. The sound card used is Linux-compatible so that it is recognized by the Linux system. Once the sound card is connected, the compatibility can be verified by entering lsusb command, which displays all USB devices. The output should include a device named “C-Media Electronics, Inc.”.

In order to interact with the USB sound card, the Intel Edison uses the Advanced Linux Sound Architecture (ALSA), which provides audio functionality to the Linux operating system [19]. In most cases, Linux systems should be able to record and play audio without any configuration. However, the USB sound card on the Intel Edison will not work without a configuration file because it is not the default device. The .asoundrc file and /etc/asound.conf are the configuration files for ALSA drivers. The BAN platform includes a .asoundrc file, shown in Figure 4.5, to set up the USB sound card as the audio device for playback and recording.



```
defaults.ctl.card 2
defaults.pcm.card 2
defaults.pcm.device 0
```

Figure 4.5: ALSA Configuration

4.4 Heart Rate Sensor

In addition to the ECG module, the BAN platform includes a very compact plug-and-play heart rate sensor at one of the wrist sensor nodes. The sensor is an open source device called pulse sensor, which is shown in Figure 4.6. The sensor can be powered by a 3.3V pin of the SparkFun ADC board and the sensor output pin can be connected to an analog input pin of the ADC board. The sensor outputs pulse waveforms rather than digital signals indicating heartbeats. The data collected on the Intel Edison is shown as a plot in Figure 4.7.

The peaks indicate heartbeats. The peaks are detected as follows. A flag is set to 0 if

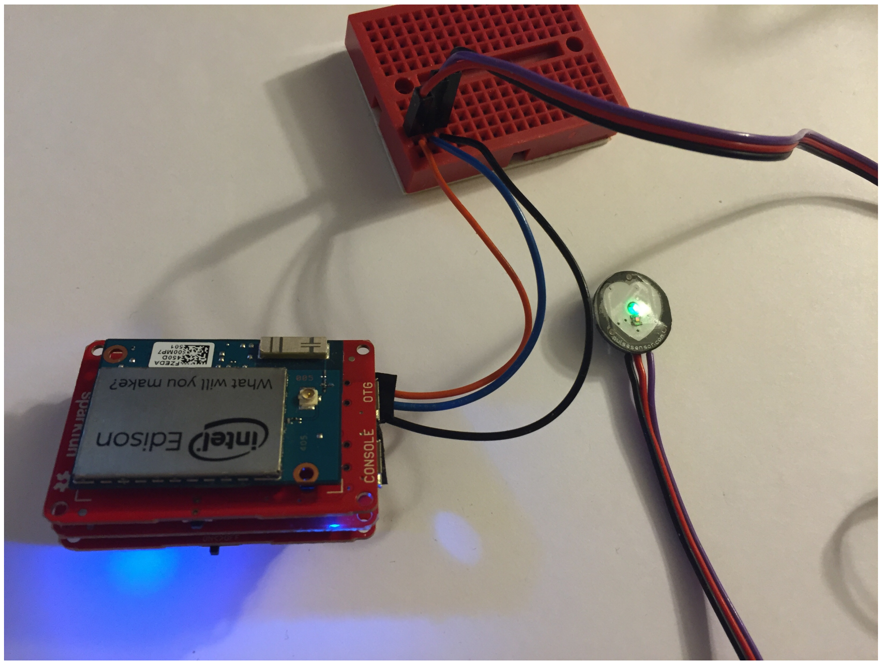


Figure 4.6: Heart Rate Sensor Setup for Testing

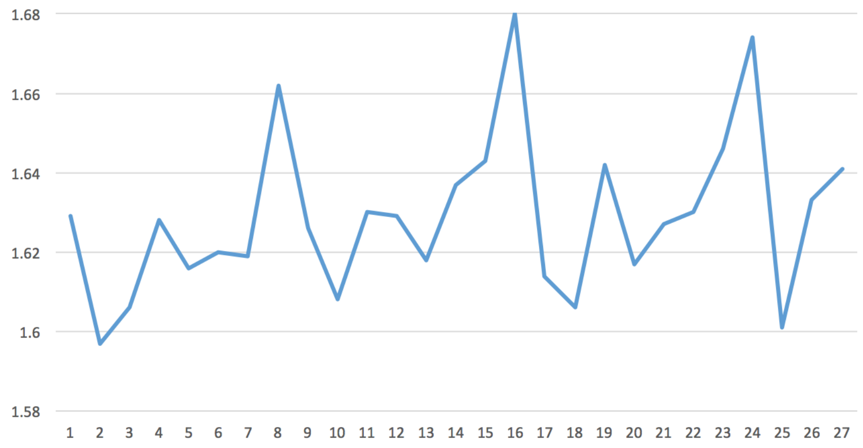


Figure 4.7: Heart Rate Sensor Data Plot

the ADC reading is below a certain threshold and is set to 1 otherwise. When the signal crosses the threshold from below, the flag changes from 0 to 1. This change is understood by the software as a heartbeat. Every time a heartbeat is detected, the software timestamps it. One minute divided by the difference between the timestamps of two adjacent peaks is the heart rate at the moment.

CHAPTER 5

Networking

With its integrated WiFi and Bluetooth Low-Energy, Intel Edisons can connect to each other. The sensor nodes in the BAN platform are connected as shown in Figure 5.1. The BAN platform features on-board data processing and is self-contained. The connection to a router is optional and the Internet access is only needed when the data is to be transmitted to clinics or the system is accessed remotely for debugging.

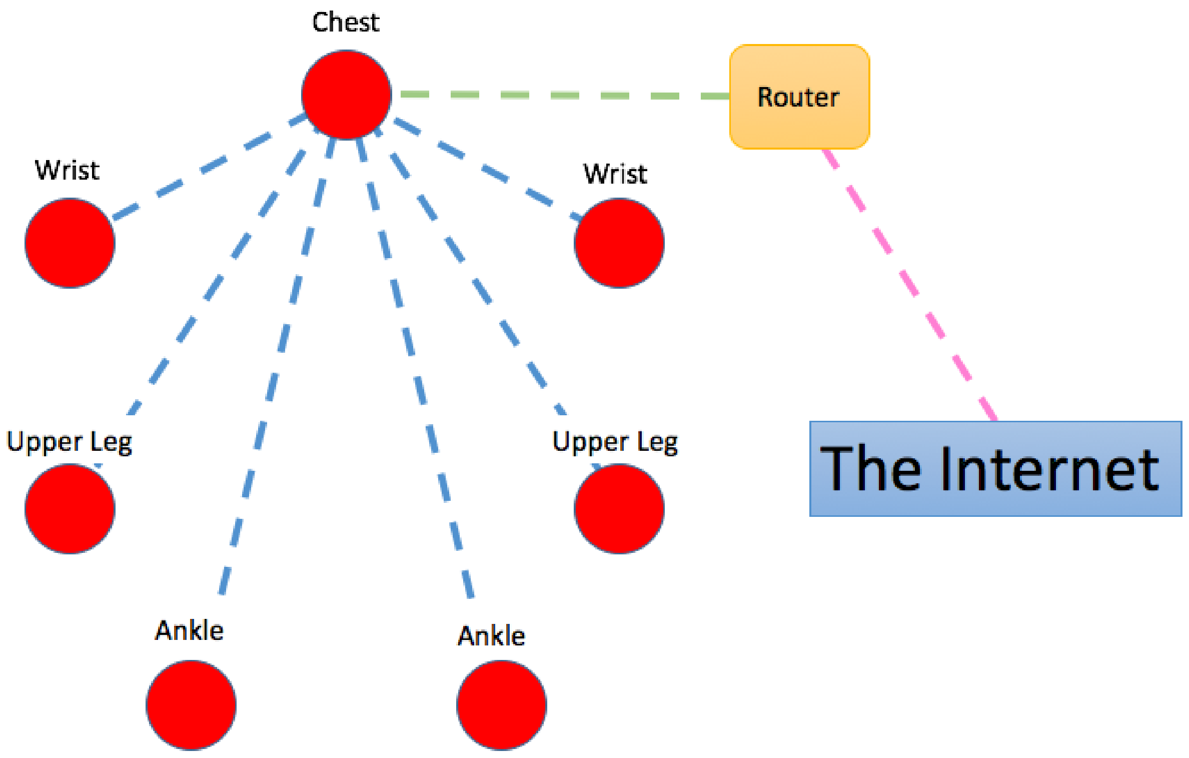


Figure 5.1: Network Diagram

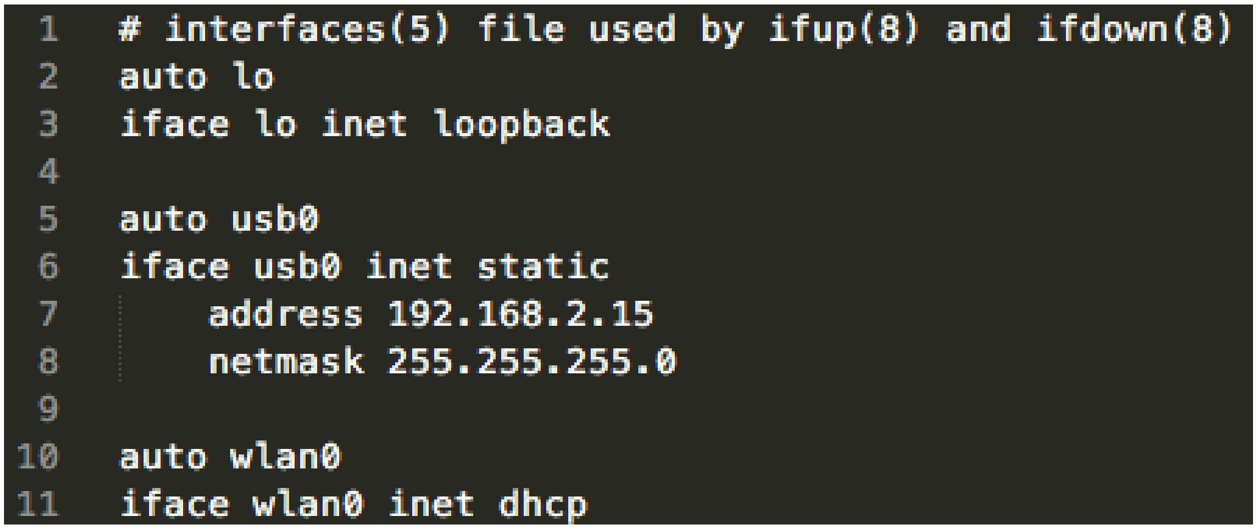
Although Bluetooth Low-Energy may be more energy efficient, WiFi is chosen for sensor node networking because experiments found the Intel Edison's WiFi interface to be more robust than its Bluetooth Low-Energy interface. The communication between sensor nodes is based on Linux sockets. Similar to a telephone, a Linux "socket represents endpoints in a line of communication" [20]. Transmission control protocol (TCP) is used at the transport layer. The communication is secured by TLS, which is a cryptographic protocol used at the application layer. The secure communication with TLS is described in section 5.2.

Some data collected by sensors at each sensor node is fused with the data collected at other sensor nodes. In order to have precise sensor fusion, time synchronization is necessary since the data must be timestamped with clocks set to the same time. Time synchronization is performed using the network time protocol (NTP). Section 5.3 will describe the time synchronization in more detail.

5.1 WiFi and Network Configuration

The method to connect to a network differs depending which Linux image is used. The chest node uses the Ubilinux while other nodes use Intel's official Yocto embedded Linux. The network configuration on the Ubilinux is identical to Debian and Ubuntu. On the Ubilinux, /etc/network/interfaces file contains the network configuration information and this can be modified to choose how the system is connected to a network. First, the system chooses whether it connects with static Internet protocol (IP) addressing or dynamic host configuration protocol (DHCP) addressing based on the configuration specified in the file. Figure 5.2 shows an example of the file's content. Lines 5 to 8 is an example of static IP addressing. The USB interface is given a static IP address of 192.168.2.15 and netmask of 255.255.255.0. On the other hand, the wireless local area network (WLAN) interface is given an IP address based on DHCP, as shown in lines 7 and 8.

The chest node connects to a router, thus manual addressing needs to be done carefully to avoid address overlaps. However, having a static IP address make it easier to remotely

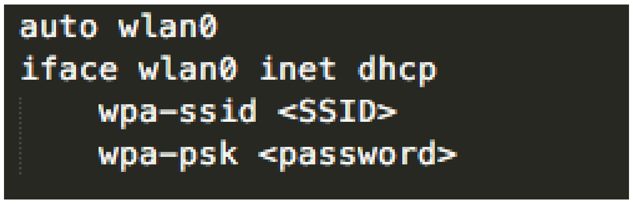


```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto usb0
iface usb0 inet static
    address 192.168.2.15
    netmask 255.255.255.0

auto wlan0
iface wlan0 inet dhcp
```

Figure 5.2: /etc/network/interfaces

access the node because the address is already known. DHCP takes care of the address overlap issue, but the IP address must be discovered in order to remotely access the node.

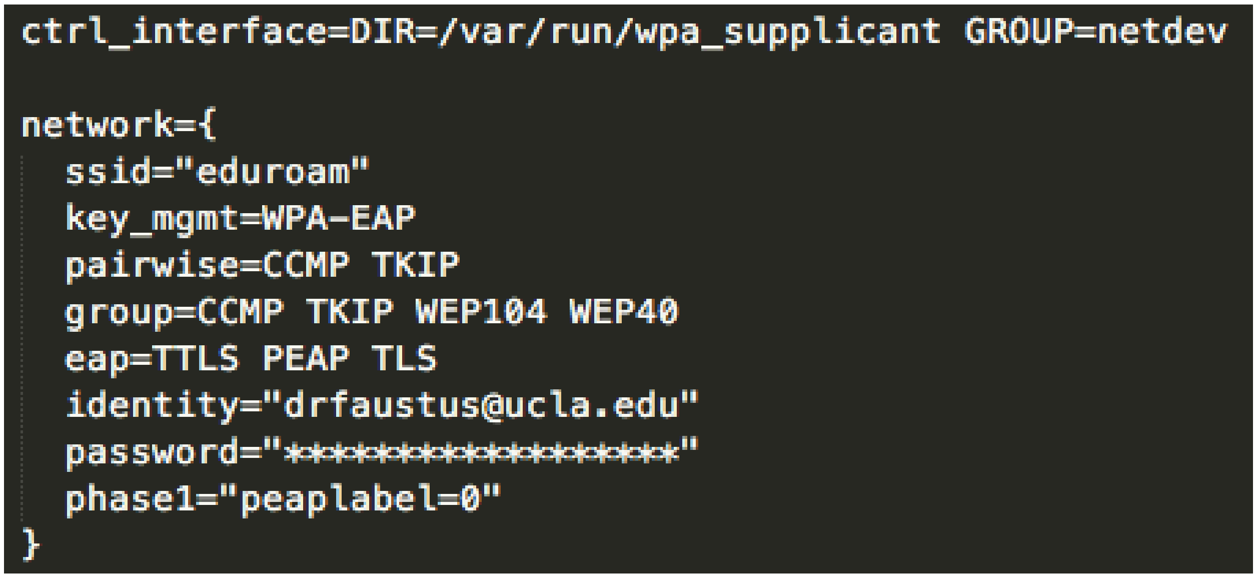
Connecting to a network is done as follows. In the portion of /etc/network/interfaces where WLAN interface configuration is specified, the service set identifier (SSID) and the password of a network can be specified as shown in Figure 5.3. For security purposes, it may not be desirable to enter the password directly. A utility called wpa_passphrase generates a WPA PSK from an ASCII password for a given SSID. Entering "wpa_passphrase ⟨SSID⟩ ⟨password⟩" will print the PSK. The PSK can substitute for ⟨password⟩ in Figure 5.3. After the network configuration is done, the WLAN driver must restart. This can be done by a simple command, "ifup wlan0".



```
auto wlan0
iface wlan0 inet dhcp
    wpa-ssid <SSID>
    wpa-psk <password>
```

Figure 5.3: SSID and Password

The Intel's official Yocto embedded Linux includes a simple utility to set up WiFi connection. The utility is called “configure_edison” and it allows user to change the device name and password, upgrade the firmware, and change the network settings. When entered with “--wifi” option, configure_edison begins scanning and then prints all available networks after the scan is done. Users can simply choose the network and enter the password. Then, the utility modifies /etc/wpa_supplicant/wpa_supplicant.conf file to add the network information. The file is then used by wpa_supplicant for network configuration. Figure 5.4 shows the file content after a user joined Eduroam network using configure_edison utility. The password is replaced with *’s in the figure.

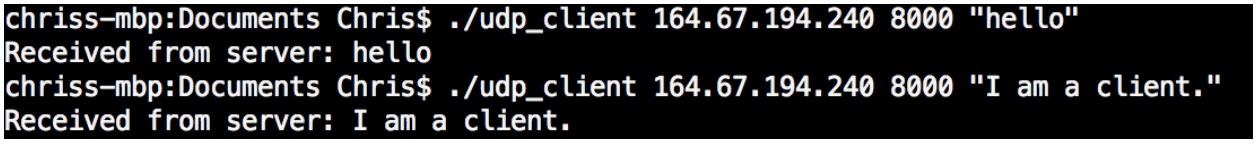


```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev

network={
  ssid="eduroam"
  key_mgmt=WPA-EAP
  pairwise=CCMP TKIP
  group=CCMP TKIP WEP104 WEP40
  eap=TTLS PEAP TLS
  identity="drfaustus@ucla.edu"
  password="******************"
  phase1="peaplabel=0"
}
```

Figure 5.4: wpa_supplicant.conf

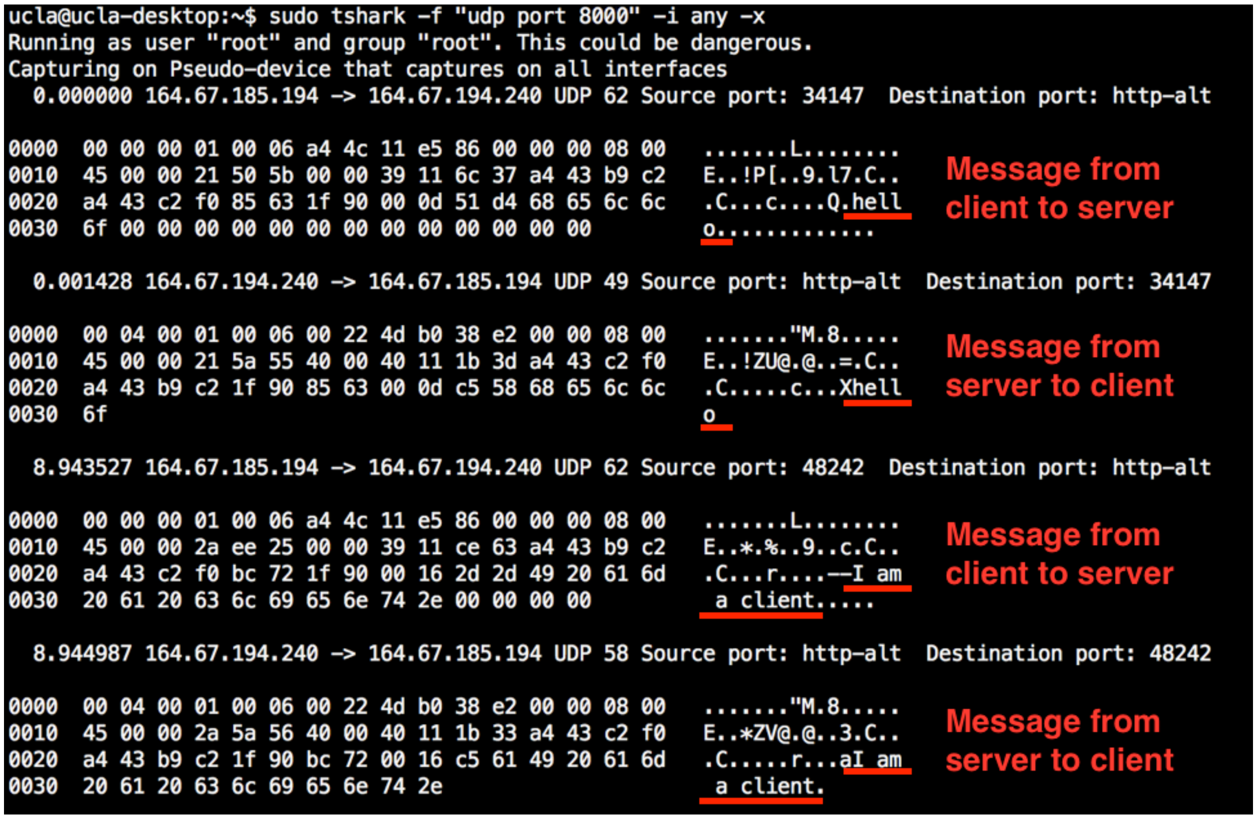
5.2 Secure Communication

The BAN platform collects the user’s health and activity data. Thus, the collected data are private information and hence must be transmitted securely. However, Linux sockets do not ensure security. It does not provide data encryption or authentication. The following experiment is performed to demonstrate the security vulnerability. A user datagram protocol (UDP) server is set up on the Linux machine whose address is 164.67.194.240. A UDP client is running on another machine sending messages to the server, which echoes the received messages back to the client. This is shown in Figure 5.5. A network protocol analyzer called TShark is used to capture packets and display their contents. As shown in Figure 5.6, the messages transmitted between the server and the client are clearly visible by TShark.



```
chriss-mbp:Documents Chris$ ./udp_client 164.67.194.240 8000 "hello"
Received from server: hello
chriss-mbp:Documents Chris$ ./udp_client 164.67.194.240 8000 "I am a client."
Received from server: I am a client.
```

Figure 5.5: Server-to-Client Communication



```
ucla@ucla-desktop:~$ sudo tshark -f "udp port 8000" -i any -x
Running as user "root" and group "root". This could be dangerous.
Capturing on Pseudo-device that captures on all interfaces
  0.000000 164.67.185.194 -> 164.67.194.240 UDP 62 Source port: 34147  Destination port: http-alt

0000  00 00 00 01 00 06 a4 4c 11 e5 86 00 00 00 08 00   .......L........
0010  45 00 00 21 50 5b 00 00 39 11 6c 37 a4 43 b9 c2   E..!P[..9.l7.C..
0020  a4 43 c2 f0 85 63 1f 90 00 0d 51 d4 68 65 6c 6c   .C...c....Q.hell
0030  6f 00 00 00 00 00 00 00 00 00 00 00 00 00         o.............

  0.001428 164.67.194.240 -> 164.67.185.194 UDP 49 Source port: http-alt  Destination port: 34147

0000  00 04 00 01 00 06 00 22 4d b0 38 e2 00 00 08 00   ......."M.8.....
0010  45 00 00 21 5a 55 40 00 40 11 1b 3d a4 43 c2 f0   E..!ZU@.@..=.C..
0020  a4 43 b9 c2 1f 90 85 63 00 0d c5 58 68 65 6c 6c   .C.....c...Xhell
0030  6f                                                o

  8.943527 164.67.185.194 -> 164.67.194.240 UDP 62 Source port: 48242  Destination port: http-alt

0000  00 00 00 01 00 06 a4 4c 11 e5 86 00 00 00 08 00   .......L........
0010  45 00 00 2a ee 25 00 00 39 11 ce 63 a4 43 b9 c2   E..*.%..9..c.C..
0020  a4 43 c2 f0 bc 72 1f 90 00 16 2d 2d 49 20 61 6d   .C...r....--I am
0030  20 61 20 63 6c 69 65 6e 74 2e 00 00 00 00          a client.....

  8.944987 164.67.194.240 -> 164.67.185.194 UDP 58 Source port: http-alt  Destination port: 48242

0000  00 04 00 01 00 06 00 22 4d b0 38 e2 00 00 08 00   ......."M.8.....
0010  45 00 00 2a 5a 56 40 00 40 11 1b 33 a4 43 c2 f0   E..*ZV@.@..3.C..
0020  a4 43 b9 c2 1f 90 bc 72 00 16 c5 61 49 20 61 6d   .C.....r...aI am
0030  20 61 20 63 6c 69 65 6e 74 2e                      a client.
```

Message from client to server
Message from server to client
Message from client to server
Message from server to client

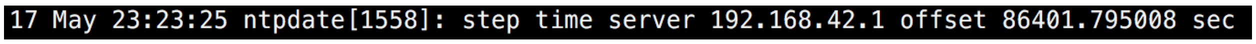
Figure 5.6: TShark Output

In order to secure the communication, TLS is used at the application layer. If the connection is secured by TLS, transmitted data between the server and the client is private because it is encrypted using symmetric cryptography. The server and the client negotiate and generate the encryption key, which is unknown to others. In addition to data encryption, TLS uses public-key cryptography for identity authentication.

5.3 Time Synchronization

The time synchronization of the sensor nodes is done with NTP. NTP is widely used for time synchronization among computers that are connected to the Internet and it is “scalable, self-configuring over multiple hops, and robust to failures” [21]. The NTP client and the server exchange packets, which are pairs of a request and a reply. The requests and replies are timestamped by the client and the server when they are sent and received. These timestamps are used to compute the round-trip delay and time offset. One packet exchange may not be enough to obtain accurate time. Several exchanges are made and then the computed delays and time offsets are put into a filter for statistical purpose.

The sensor nodes do not need to have an accurate clock. The need for time synchronization comes from the need for synchronized data timestamps within the BAN. Thus, the sensor nodes do not need to synchronize their clocks with time servers such as time1.ucla.edu. Instead, the sensor nodes can synchronize their clocks to the chest node’s clock. This is done by using ntpdate utility with the chest node’s IP address specified. Neither the Ubilinux nor the Yocto embedded Linux includes pre-installed ntpdate. On the Ubilinux, ntpdate can be downloaded and installed with an apt-get command. On the other hand, the Yocto embedded Linux must be provided with a package repository that contains ntpdate so that ntpdate can be downloaded and installed using OPKG, the package management for Yocto embedded Linux. The ntpdate also outputs the time offset once time synchronization is complete. Figure 5.7 is an example output, in which the server’s address is 192.168.42.1 and the time offset is 86401.795008 seconds, which is one day and 1.795008 seconds. In this experiment, the client clock was intentionally changed approximately one day earlier than the server clock prior to time synchronization.



```
17 May 23:23:25 ntpdate[1558]: step time server 192.168.42.1 offset 86401.795008 sec
```

Figure 5.7: ntpdate Output Example

CHAPTER 6

Supported Algorithms

Enabled by its Intel Atom dual-core processor, the BAN platform sensor node can host computationally intensive algorithms. These include signal processing and machine learning algorithms, which can be utilized for sensor fusion. A neural network package called NeuPy is installed on the chest node, enabling machine learning-based classification to monitor users' states such as motion and voice activity.

For the motion monitoring scenario, 9DOF IMU sensor data is collected on each sensor node and transmitted securely to the chest node. From the sensor data, features such as root mean square and zero-crossings are extracted. The extracted features are used to train a classifier. Once the classifier is trained, it can classify motion with features extracted from future data. The highlight of this is that all of these steps are performed on-board at the chest sensor node, without needing a separate processing device.

NumPy and SciPy packages are installed on the chest node to enable rapid development of systems such as the motion monitoring. NumPy and SciPy are Python packages that enable intuitive coding for numerical computation, signal processing, statistics, and optimization. These packages implement many of the commonly used Matlab functions, thus a developer can create a prototype in Matlab and port the Matlab code to Python using these packages. To demonstrate this, a Butterworth filter is used in Matlab to filter a signal and then the code is ported to Python. Figure 6.1 compares plots from the Matlab code and the Python code. The green lines in both plots are the unfiltered signal and the red lines are the filtered signals.

Experiments are performed to evaluate Neupy's performance which is then compared

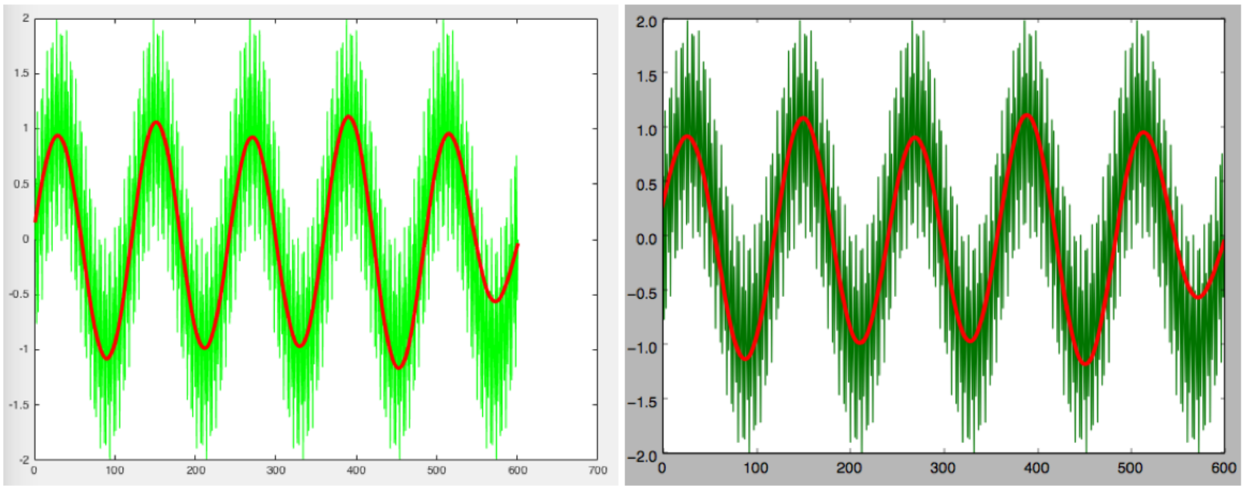


Figure 6.1: Plot from Matlab (left) and Plot from Python

to that of Matlab. First, the best performing algorithm was found for NeuPy. Example data sets were used to train neural networks with four different training algorithms. Each neural network was trained and its performance in terms of mean squared error (MSE) was evaluated. This was performed fifty times for each training algorithm. The MSE values were plotted as boxplots to visualize the algorithms' performance. Figure 6.2 shows the results of the experiments performed with two different data sets. As clearly shown, the conjugate gradient algorithm outperforms others. This is verified with more example data.

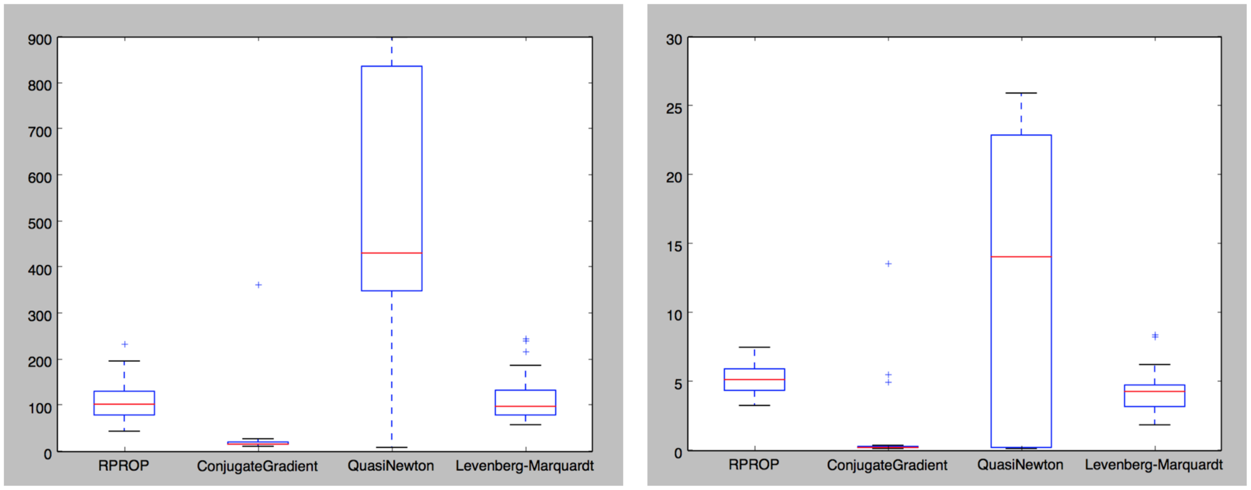


Figure 6.2: Comparison of Performance of NeuPy's Training Algorithms

Three example data sets from Matlab's Neural Network toolbox were used to compare the performance of training algorithms in Matalb's Neural Network toolbox and the conjugate gradient algorithm in NeuPy. As shown in Figure 6.3 and Figure 6.4, the performance of NeuPy's conjugate gradient is on par with those of Matlab's training algorithms.

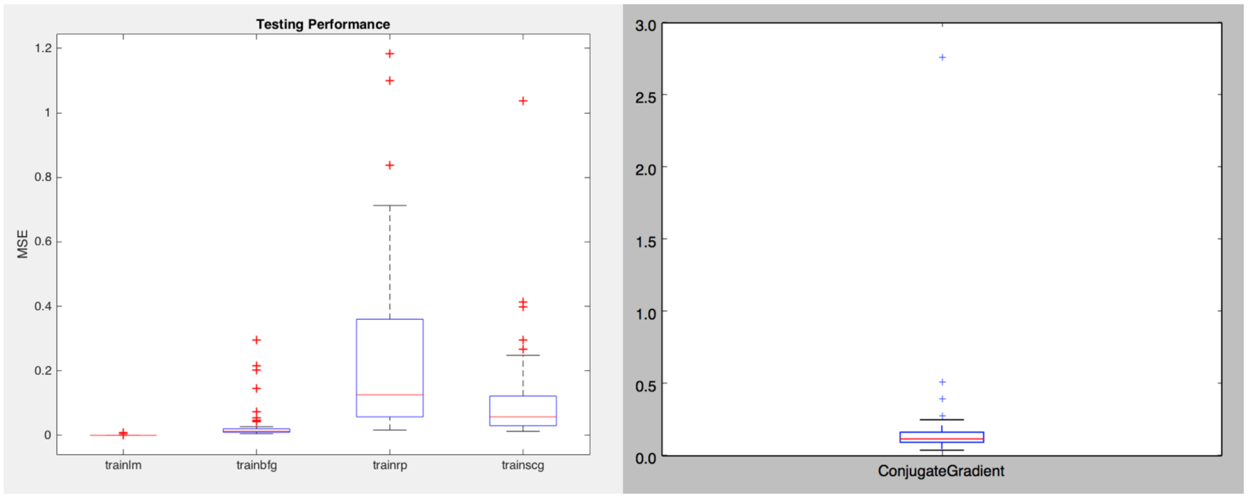


Figure 6.3: Training Algorithm Performance Comparison between Matlab (left) and NeuPy (right) on Data Set 1

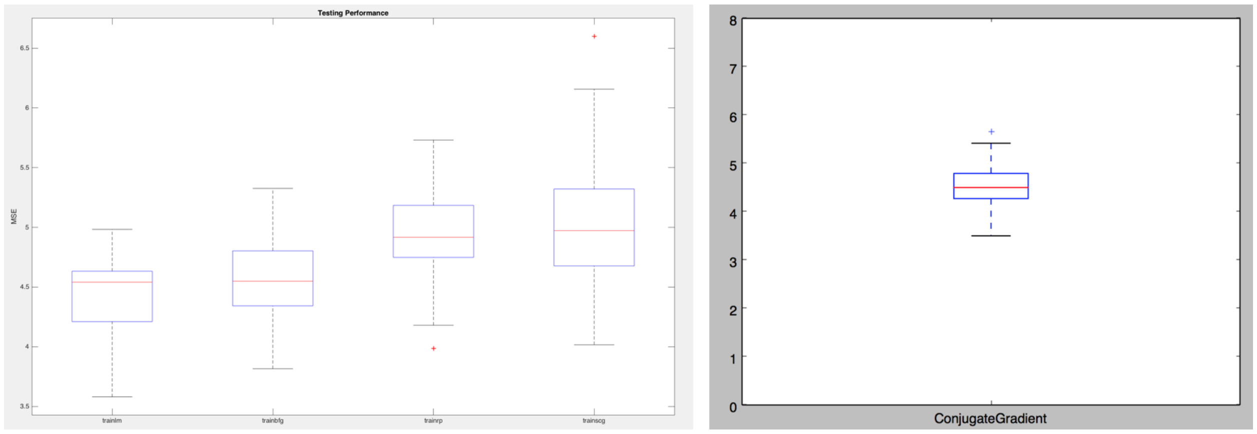


Figure 6.4: Training Algorithm Performance Comparison between Matlab (left) and NeuPy (right) on Data Set 2

CHAPTER 7

Education mission

The BAN platform development helped initiate an IoT curriculum at UCLA. A general-purpose Intel Edison-based IoT platform is also developed along with the BAN platform. These platforms were distributed to a freshman course, a senior capstone design course, and a graduate course. Tutorials on Intel Edison assembly, Yocto embedded Linux, networking, and external interfaces were developed to help students in these courses become familiarized with the Intel Edison and sensor/actuator interface development. In addition, tutorials on cloud-based data analytics, robotics, audio interface, and more in-depth BAN platform tutorials were developed and provided to specific groups of students depending on their project topics. The student projects include an IoT robot coordination system, vigilant robotics, smart wearables for workplace performance advancement, and an athletic performance advancement system. This chapter will introduce some of the projects.

The freshman course was offered in fall 2015. Due to the fact that these incoming freshmen came straight out of high school, most students had minimal background in engineering and programming. The IoT platform provided more motivating opportunities to learn about embedded systems and programming. A group of freshmen students built an IoT robot coordination system, in which robots securely exchange C code over SSH. The C code files include robot operations. The robots continuously check for the C code files' arrivals. Once the files are arrived, the robots compile and execute the code. The project involved robot assembly, circuit implementation, and software architecture design. This project was completed in ten weeks despite not having any prior programming or soldering experience.

Another robotics project was done by a group in the senior capstone design course. This group of three seniors designed a vigilant robot, which detects and follows suspicious or malfunctioning robots. The vigilant robot is shown in Figure 7.1. The robot includes the SparkFun ADC board and GPIO board. The same ADC and GPIO libraries developed for the BAN platform are used in this project to interface with the boards.

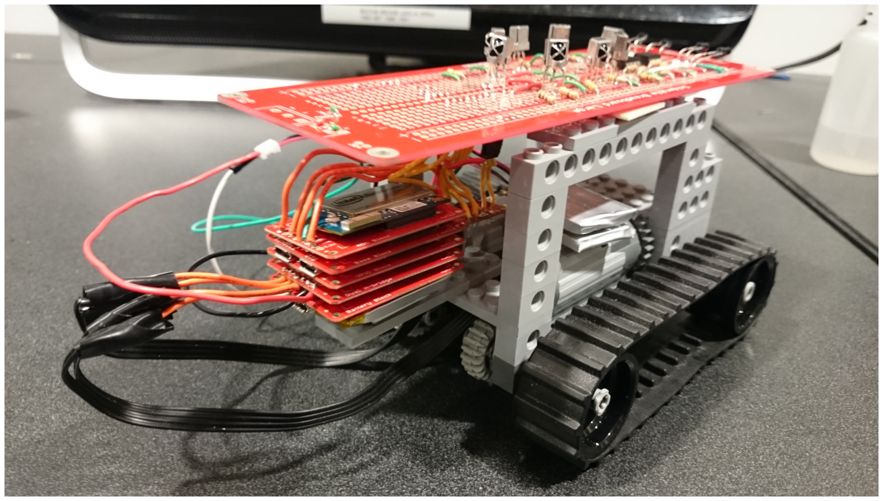


Figure 7.1: Vigilant Robot

Some of the other senior design projects are the smart wearables for workplace performance advancement (workplace performance in short), and the athletic performance advancement system (athletic performance in short). Both of these projects utilize the BAN platform. The workplace performance system includes posture and motion monitoring based on a neural network classifier trained with the 9DOF IMU data at the sensor nodes. With NTP, the sensor nodes have synchronized time to ensure precise relative data timestamps. It also includes neural network-based voice activity detection to monitor the user's interaction with others. Also, IR beacons and detectors are implemented to locate the user. Figure 7.2 shows a group member wearing the sensor nodes for system testing. The athletic performance system also includes motion monitoring based on a neural network classifier trained with data collected by the 9DOF IMUs worn by the user. Figure 7.3 shows the 9DOF IMU data collected from an arm curl. The sensor nodes in the system communicate with sockets and have TLS for secure connection. NTP is also utilized in this project for time synchronization.

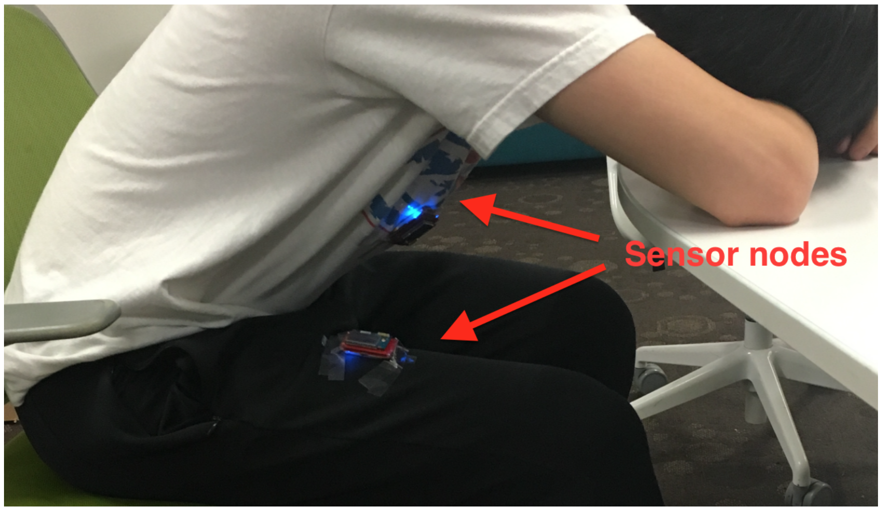


Figure 7.2: Smart Wearable for Workplace Performance Advancement

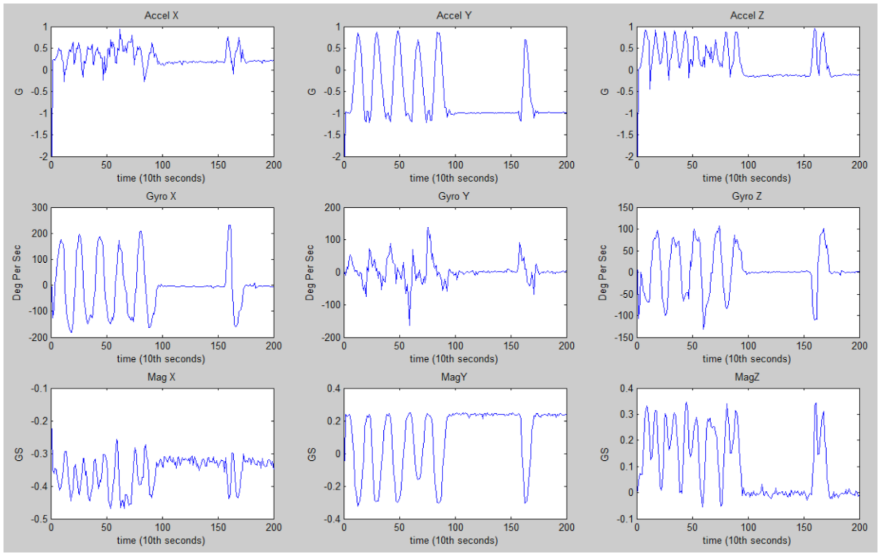


Figure 7.3: Athletic Performance Advancement System 9DOF Data from an Arm Curl

The IoT curriculum was very motivating for the undergraduate students and it was challenging enough for the graduate students. The students in the graduate course built autonomous delivery robots, drones, Bluetooth RSSI-based patient localization, and IR beacons. All of these projects in addition to freshmen projects and senior projects were very successful.

CHAPTER 8

Conclusion and Future Work

This thesis presented an IoT BAN platform that provides rapid development capability and scalability. The computer-on-module and expansion boards are selected to enable flexible interfacing with sensors via GPIO, I2C, SPI, UART, USB, or ADC. For each interface, a software API is developed or thoroughly studied if it is already available. A base model of the BAN platform includes 9DOF IMUs, an ECG sensor, a microphone, and a heart rate sensor. The sensor interfaces were developed and tested.

The platform has room for improvement. The current version includes a few types of sensors. In the future, the base model can include more sensors such as an SpO2 sensor, a temperature sensor, and electromyography (EMG) sensors. Each sensor interface can be modular so that the developer can choose to include only the needed sensors. This will involve the development of consistent and modular APIs for the sensor interfaces. The availability of more sensor interfaces will make the BAN platform more general-purpose. Due to the modularity of the sensor interfaces, the developers will be able to create a stripped-down version that is specific to their need.

The platform's power consumption can be improved as well. The Intel Edison features a CPU and a microcontroller. A power management system can be designed for the platform based on that the microcontroller consumes a lot less power than the CPU. A possible power management is as follows. The CPU is in a sleep state while the microcontroller collects data. When the data needs to be processed by the CPU, the microcontroller wakes up the CPU and sends the data via UART connection between the CPU and the microcontroller. The data is available at /dev/ttymcu0 on the CPU.

The current version of the BAN platform uses a SparkFun battery block to supply power to each sensor node. It includes a micro USB LiPo charger. Users need to connect the sensor nodes to micro USB cables, which then need to be plugged into 5V USB ports in order to charge the batteries. Such inconvenience can be solved by replacing the USB chargers with Qi wireless chargers. The wireless charging capability will allow users to simply place the sensor nodes on wireless charging pads when they are not in use.

The BAN platform and the general-purpose IoT platform were successfully distributed to freshman, senior, and graduate courses with exceptional assessment records. These courses covered a wide range of topics and offered hands-on experience on the topics since the BAN platform development involved hardware, software, embedded systems, sensors, networking, security, Linux, machine learning, and biomedical system design.

The platform's ability to host intensive computation and successful neural network solution allowed students to build classifiers to monitor motion, posture, and voice activity without developing any external processing devices. The performance of neural networks trained on the platform was on par with that of neural networks trained on Matlab.

With its flexible interfaces, intuitive APIs, simple secure networking solution, and ability to host a high performance neural network, the BAN platform provides a rapid development capability. The IoT courses have shown that the students were able to develop fully functioning biomedical devices based on the BAN platform in a short amount of time.

APPENDIX A

Acronym List

9DOF IMU	9 Degree-of-Freedom Inertial Measurement Unit
ADC	Analog-to-Digital Converter
ALSA	Advanced Linux Sound Architecture
API	Application Program Interface
BAN	Body Area Network
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
ECG	Electrocardiogram
EMG	Electromyography
FTDI	Future Technology Devices International
GPIO	General Purpose Input Output
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IC	Integrated Circuit
IoT	Internet of Things
IP	Internet Protocol
IR	Infrared
LiPo	Lithium Polymer
MCU	Microcontroller Unit
MISO	Master In Slave Out
MOSI	Master Out Slave In

MSE	Mean Squared Error
NTP	Network Time Protocol
OS	Operating System
OTG	On-The-Go
PDA	Personal Digital Assistant
PSK	Pre-shared Key
PWM	Pulse Width Modulation
RSSI	Received Signal Strength Indication
RTOS	Real-Time Operating System
SCL	Serial Clock Line
SCLK	Serial Clock
SDA	Serial Data Line
SoC	System on Chip
SPI	Serial Peripheral Interface
SpO2	Peripheral Capillary Oxygen Saturation
SS	Slave Select
SSH	Secure Shell
SSID	Service Set Identifier
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TRRS	Tip Ring Ring Sleeve
TRS	Tip Ring Sleeve
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WSN	Wireless Sensor Network

References

[1] Intel. Creating applications with the mcu sdk for the intel edison board. Available at https://software.intel.com/en-us/node/545142.

[2] Intel. *Intel Edison Kit for Arduino Hardware Guide*, February 2015.

[3] SparkFun. I2c. Available at https://learn.sparkfun.com/tutorials/i2c.

[4] Bandar Alghamdi and Hacne Fouchal. A mobile wireless body area network platform. *computational science*, 5(4):664–674, July 2014.

[5] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *computer networks*, 52(12):2292–2330, August 2008.

[6] Paul Honeine, Farah Mourad, Maya Kallas, Hichem Snoussi, Hassan Amoud, and Clovis Francis. Wireless sensor networks in biomedical: Body area networks. In *Systems, Signal Processing and their Applications (WOSSPA), 2011 7th International Workshop on*, pages 388–391, May 2011.

[7] IEEE. Ieee 802.15 wpan task group 6 (tg6) body area networks. Available at http://www.ieee802.org/15/pub/TG6.html.

[8] Emil Jovanov, Aleksandar Milenkovic, Chris Otto, and Piet C. de Groen. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2(1):1–10, 2005.

[9] Joonyoung Jung, Kiryong Ha, Jeonwoo Lee, Youngsung Kim, and Daeyoung Kim. Wireless body area network in a ubiquitous healthcare system for physiological signal monitoring and health consulting. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 1(1):47–54, 2008.

[10] Chris Otto, Aleksandar Milenkovic, Corey Sanders, and Emil Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *Journal of mobile multimedia*, 1(4):307–326, 2006.

[11] E Monton, José F Hernandez, José Manuel Blasco, Thierry Hervé, Joseph Micallef, Ivan Grech, Andrea Brincat, and V Traver. Body area network for wireless patient monitoring. *Communications, IET*, 2(2):215–222, 2008.

[12] Intel. *Intel Edison Product Brief.*

[13] EmutexLabs. Ubilinux, July 2014. Available at http://www.emutexlabs.com/ubilinux.

[14] Intel. *MRAA: Low Level Skeleton Library for Communication on GNU/Linux platforms.* Available at http://iotdk.intel.com/docs/master/mraa/index.html.

[15] Intel. *Intel Edison Compute Module Hardware Guide*, January 2015.

[16] SparkFun. Sparkfun block for intel edison - 9 degrees of freedom. Available at https://www.sparkfun.com/products/13033.

[17] STMicroelectronics. *iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer*, August 2013.

[18] MG Naazneen, Sumaya Fathima, Syeda Husna Mohammadi, Sarah Iram L Indikar, Abdul Saleem, and Mohamed Jebran. Design and implementation of ecg monitoring and heart rate measurement system. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, 2(3):456–465, May 2013.

[19] ALSA Project. Advanced linux sound architecture (alsa) project homepage. Available at http://www.alsa-project.org/main/index.php/Main$_Page$.

[20] Warren Gay. *Linux Socket Programming By Example.* Que, Indianapolis, Indiana, 2000.

[21] G.J. Pottie and W.J. Kaiser. *Principles of Embedded Networked Systems Design.* Cambridge University Press, 2009.