

# **Bluetooth Low Energy transmitted real-time sensor data visualization on Android and System-on-Chip analysis**

**Bachelor paper 2**

FH Campus Wien  
ITTK

**Submitted by:**

Erika Wood

**Student ID**

c1410475072

**Specialization:**

Telecommunications

**Submitted on:**

June 30, 2018

**This research project was conducted within the scope of the  
Austrian Marshall Plan Foundation Scholarship**

Ich erkläre, dass die vorliegende Bachelorarbeit von mir selbst verfasst wurde und ich keine anderen als die angeführten Behelfe verwendet bzw. mich auch sonst keiner unerlaubter Hilfe bedient habe.

Ich versichere, dass ich diese Bachelorarbeit bisher weder im In- noch im Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Weiters versichere ich, dass die von mir eingereichten Exemplare (ausgedruckt und elektronisch) identisch sind.

Datum: 30. Juni, 2018

Unterschrift: *Aika Hood*

## Abstract

Advanced Self-Powered Systems of Integrated Sensors and Technologies (ASSIST) research at the SF Nanosystems Engineering Research Center (NERC) advances health informatics and biomedical engineering. ASSIST research focuses on developing nanotechnology-based systems to empower wearable monitoring platforms composed of embedded battery-free sensing, human body energy harvesting and wireless communication interfaces that aim to improve understanding of health and environmental exposure related adverse health responses. The ASSIST Health and Environment Tracker (HET) system testbed is a sensor system monitoring physiological and ambient parameters. The goal of this Bachelor thesis is to provide real-time data visualization and data aggregation support to the HET project by implementing an Android user interface enabling Bluetooth Low Energy (BLE) communication technology for low power wireless data transfer of sensor data transmitted by the HET monitoring platform. The BLE compliant Texas Instrument (TI) System-on-Chip (SoC) CC2451 currently integrated in the wearable HET system operating on limited lithium battery power supply provides power optimized components to allow energy efficient operations. In an effort to improve wearability and usability a custom SoC platform is being developed by the ASSIST research team to significantly reduce power consumption ultimately enabling ultra-low power operations for battery-less sensing. A comparative analysis of the respective TI and ASSIST SoC components presented in this paper highlights the advancements towards ultra-low power wireless communication to overcome impeding development barriers imposed on self-powered body sensor systems.

## Zusammenfassung

Advanced Self-Powered Systems of Integrated Sensors and Technologies (ASSIST) Forschung am SF Nanosystems Engineering Research Center (NERC) bringt Fortschritte in den Bereichen Gesundheitsinformatik und biomedizinisches Ingenieurwesen. ASSIST Forschung konzentriert sich auf die Entwicklung von nanotechnologiebasierenden Systemen, um tragbare batteriefreie Sensorplattformen zu ermöglichen, die aus Komponenten zur Energiegewinnung aus Körperwärme sowie drahtlosen Kommunikationsschnittstellen bestehen, mit dem Ziel, ein besseres Verständnis über den Zusammenhang zwischen Gesundheit und krankheitserregenden Umwelteinflüssen zu gewinnen. Die ASSIST Health and Environment Tracker (HET) Systemtestplattform dient zur Messung von physiologischen Parametern sowie Umgebungsparametern. Ziel dieser Bachelorarbeit ist die Implementierung einer Android Benutzeroberfläche, die Bluetooth Low Energy (BLE) als Kommunikationstechnologie zwecks energiesparender, drahtloser Datenübertragung einsetzt, um Datenvisualisierung in Echtzeit sowie Datensammlung von Sensordaten, die über die HET Überwachungsplattform gesendet werden, zu ermöglichen. Der derzeitige auf der tragbaren HET Plattform integrierte BLE konforme Texas Instruments (TI) System-on-Chip (SoC) CC2541 arbeitet mit begrenzter Energieversorgung, die mittels einer Lithiumbatterie zugeführt wird und stellt energieoptimierte Komponenten zur Verfügung, um energieeffiziente Arbeitsschritte zu ermöglichen. Das ASSIST Forschungsteam arbeitet an der Entwicklung einer SoC Plattform, die eine beträchtliche Reduzierung des Energieverbrauchs ermöglicht, um letztendlich Operationen mit extrem niedrigen Stromverbrauch für einen batterielosen Betrieb zu erzielen. Ein in dieser Arbeit präsentierter Vergleich der Komponenten des TI SoCs sowie der ASSIST SoC Entwicklung

streicht die Fortschritte Richtung drahtloser Kommunikation mit extrem niedrigen Energieverbrauch hervor, um in Verbindung mit selbstversorgenden Körpersensoren-systemen stehende Entwicklungsbarrieren zu überwinden.

## List of abbreviations

ADC	Analog-to-digital converter
ADPLL	All Digital Phase Locked Loop
AMQP	Advanced Message Queuing Protocol
ASIC	Application Specific Integrated Circuit
ASSIST	Advanced Self-Powered Systems of Integrated Sensors and Technologies
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
BT	Bluetooth
CBMS	Cold-Boot Management System
COTS	Commercial off-the-shelf
CPU	Central Processing Unit
DMA	Direct Memory Access
DPM	Digital Power Management
EH	Energy Harvesting
FDM	Frequency Division Multiplexing
FIFO	First In First Out
GAP	Generic Application Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
GPIO	General Purpose Input Output
HCI	Host Controller Interface
HET	Health and Environment Tracker
I/O	Input/Output
I2C	Inter-Integrated Circuit
IEEE	
IMEM	Instruction Memory
IR	Infrared
IRQ	Interrupt
ISM	Industrial, scientific and medical radio band
ISM	Industrial, scientific and medical
L2CAP	Logical link control and adaptation layer protocol
LCU	Lightweight Control Unit
LE	Low Energy

LPC	Low Power Controller
MSK	Minimum-Shift Keying
NCSU	North Carolina State University
NERC	Nanosystems Engineering Research Center
NSF	National Science Foundation
OMSP	OpenMSP430
OOK	On/Off Keying
PDU	Protocol Data Units
PHY	Physical Layer
PM	Power Monitor
PMC	Power Management Controller
PMU	Power Management Unit
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RX	Receiver
SFR	Special Function Register
SFTP	SSH File Transfer Protocol
SIG	Special Interest Group
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSH	Secure Shell
TEG	Thermoelectric Generators
TI	Texas Instruments
TX	Transmitter
UI	User Interface
ULP	Ultra-low-Power
UML	Unified Modelling Language
USART	Universal Synchronous and Asynchronous Receiver-Transmitter
UUID	Universally Unique Identifier
VPN	Virtual Private Network
WuRx	Wake Up Receiver

## Table of Contents

<b>ABSTRACT .....</b>	<b>III</b>
<b>ZUSAMMENFASSUNG .....</b>	<b>III</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>V</b>
<b>1 INTRODUCTION .....</b>	<b>8</b>
<b>2 THEORETICAL BACKGROUND.....</b>	<b>9</b>
<b>3 SoC COMPARISON .....</b>	<b>17</b>
3.1. System architecture overview .....	17
3.2. ULP related features.....	20
3.3. Comparison.....	22
<b>4 METHODOLOGY AND IMPLEMENTATION .....</b>	<b>23</b>
4.1. Requirements specification .....	23
4.2. Design specification.....	25
4.3. Implementation .....	26
<b>5 RESULTS &amp; DISCUSSION.....</b>	<b>37</b>
<b>6 RELATED WORK.....</b>	<b>39</b>
<b>7 CONCLUSION AND FUTURE WORK .....</b>	<b>40</b>
<b>LIST OF FIGURES .....</b>	<b>41</b>
<b>LIST OF LISTINGS .....</b>	<b>42</b>
<b>LIST OF TABLES.....</b>	<b>43</b>
<b>REFERENCES .....</b>	<b>44</b>

# 1 Introduction

## Motivation

The purpose of this research project is to support the Health and Environment Tracker (HET) project of the NSF Engineering Research Center for Advanced Self-Powered Systems of Integrated Sensors and Technologies (ASSIST) at North Carolina State University (NCSU) in Raleigh [ASSIST]. The ASSIST HET system testbed platform is one of the biocompatible sensor systems monitoring individual biomedical and environmental parameters. A wireless communication interface is provided to enable transmission of heterogeneous data composed of parameters that aim to improve understanding of health and environmental exposure related adverse health responses.

The implementation of an Android user interface establishing a Bluetooth Low Energy (BLE) wireless communication with the ASSIST HET wristband serves a dual purpose comprised of data aggregation support and real time data visualization.

Wireless communication is one of the major contributing factors of high energy consumption in wearable body sensor systems. A custom System-on-Chip (SoC) is being developed by ASSIST to significantly reduce power consumption. A comparative analysis of the commercially-off the-shelf (COTS) Texas Instruments (TI) SoC and the development achievements of ASSIST towards an ultra-low power (ULP) SoC highlights key custom elements and components designed to ultimately lower power consumption.

## Relevance and expected results

Applications establishing BLE communication with in vivo sensors enabling ubiquitous monitoring of physiological and ambient parameters represent a growing segment in the IoT market. Mobile applications in association with wearable devices constitute key components contributing to data aggregation efforts and enhancing user experience. Facilitating data aggregation in a joined effort for continuous data storage supports the ASSIST Testing and Data Analysis Thrust V team in their efforts to perform key data analytics advancing medicine driven by information extraction from collected data and pattern recognition bridging the correlation between environmental exposure and adverse health effects. Real time visualization of individual physiological parameters enhances user awareness of aforementioned causation and contributes towards effective health monitoring of biomedical sensor data received from wearable sensors based on ultra-low power wireless communication. In order to achieve ultra-low power wireless communication used to tether body sensor systems to data access points such as smartphones power reduction is one of the major challenges being tackled by scientists of Thrust IV. Researchers aim to create key hardware components in an ongoing development of a custom SoC designed to achieve overall operations in the submilliwatt range. This paper provides an overview of the key ULP components by comparing architectural SoC designs supporting ULP energy consumption for wireless data transfer of environmental and physiological health sensing parameters.



### **Methodological considerations**

The SoC comparison is based on the technical documentation provided by Texas Instruments and IEEE publications in relation to the development of an ultra-low power SoC. The DBIS database and the IEEE/IET Electronic online library is used to perform literature search. Abstracts of conference proceedings and publications containing relevant acronyms are scanned for appropriate content and full-text papers are obtained as referenced. Content-based classification of publications is performed to organize literature results. Relevant figures are referenced for illustrative purposes. The literature search is concluded with supplementary documentation provided by the framework websites. Book reviews are included in the scope of Software Development.

The implementation of BLE data aggregation support and data visualization follows the phases of the Software Development process coherent to the Waterfall Model as elaborated in [SDWM]. Product scope, hardware and system requirements as well as functional and non-functional requirements are outlined in the Software Requirements Specifications (SRS) document. System visualization is facilitated by using the Unified Modeling Language (UML) Design providing a high-level abstraction layer of the application [UML].

### **Structure of the paper**

The rest of this paper includes an elaboration of the theoretical background in Chapter 2 focusing on the HET 1.0 system testbed deploying an SoC by Texas Instruments. A comparative analysis of selective SoCs is presented in Chapter 3. The specifications of requirements and design and the subsequent implementation are outlined in Chapter 4. Implementation results are demonstrated in Chapter 5 along with a discussion reflecting upon the results. An overview of related work is presented in Chapter 6. Conclusion is given in the final Chapter including an outlook on future work regarding the Android HET ASSIST application.

## **2 Theoretical background**

ASSIST wearable sensor monitoring platforms are developed to track biomedical and environmental parameters. Five cross-disciplinary and collaborative ASSIST research Thrusts incorporating human and environmental factors are geared to develop highly effective technologies for harnessing and storing human body energy in the long term. This combined effort ultimately enables ultra-low power communication and computation and empowers nanotechnologies for sensing through integration in a wearable sensor system designed to provide comfortable and biocompatible health monitoring devices. The monitoring device used for this Android user interface implementation is part of the HET wearable sensor system comprised of a chest patch and a wristband [HET1].

The current wristband prototype consists of COTS components to simultaneously measure the level of ozone, the variability of heart-rate, motion in terms of three-axis acceleration, as well as ambient temperature and humidity values using a lithium polymer battery as a power source.

The long-term vision of the five integrated ASSIST research thrusts is to develop a unified battery-free system using self-powered technology harvesting human body heat and motion. Gradually integrating a unique set of power optimised sensors, continuous energy harvesting modules, a custom ultra-low power (ULP) radio are key power reducing strategies towards achieving self-powered operations.

Radio transmissions are performed using the BLE technology introduced with Bluetooth 4.0 coexisting with WiFi channels in the 2.4 GHz ISM spectrum [BLE]. BLE operations are performed on the physical advertising channel and the physical data channel. Fig. 1 [BLE4] shows 40 radio frequency (RF) channels separated by 2MHz that are available for allocation to the two physical channels.

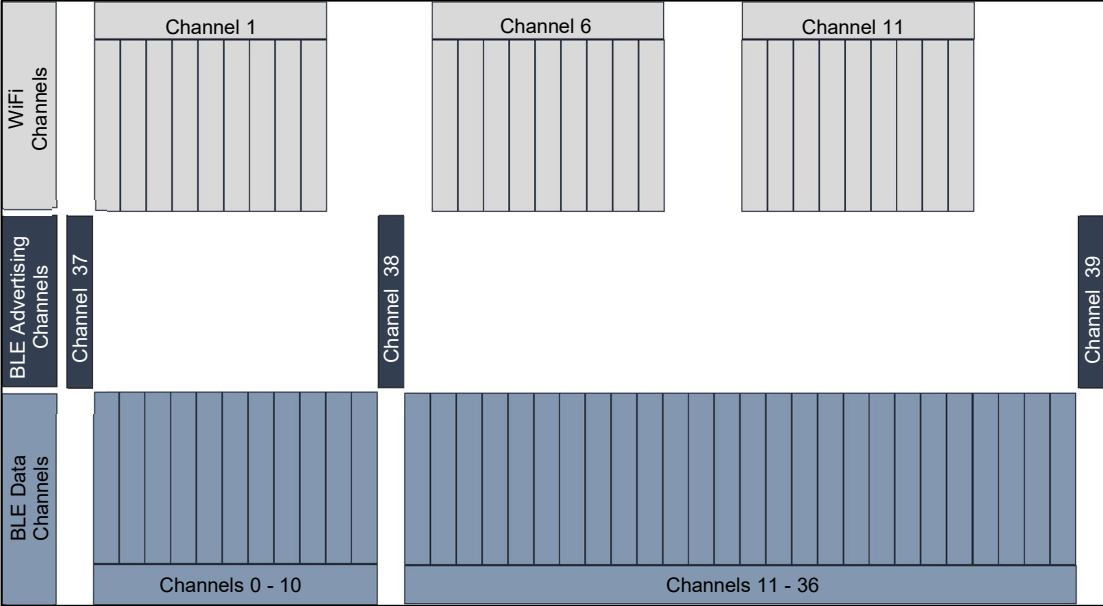


Fig. 1 BLE Frequency spectrum (see [BLE4])

The RF channels are spread in between wireless LAN channels to mitigate frequency interferences. Up to 37 RF channels can be used by the physical data channel. The remaining three RF channels 37, 38 and 39 positioned at index 0, 12 and 39 are dedicated advertising channels. Advertising channels are used to broadcast a device's presence and to perform device discovery and connection initiation to establish a BLE communication. The same RF channel needs to be used by BLE radio transceiver components of the BLE Physical Layer (PHY) illustrated in Fig. 2 [BLE5] to enable a BLE connection between devices.

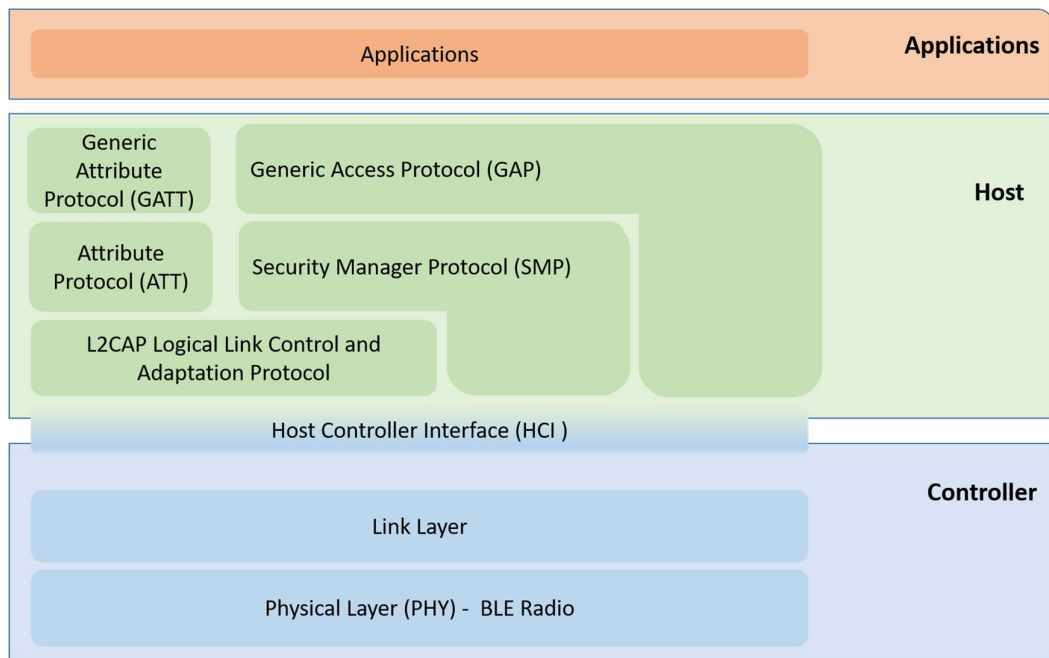


Fig. 3 BLE Layer architecture (see [BLE5]).

One physical channel is used at any time. The Link Layer of the BLE device looking for advertising devices listens on the RF channels assigned to the physical advertising channel during a device scan and in the connection initiating state as indicated in Fig 3 [BLE6]. The connection initiating link layer takes the master role of the connection.

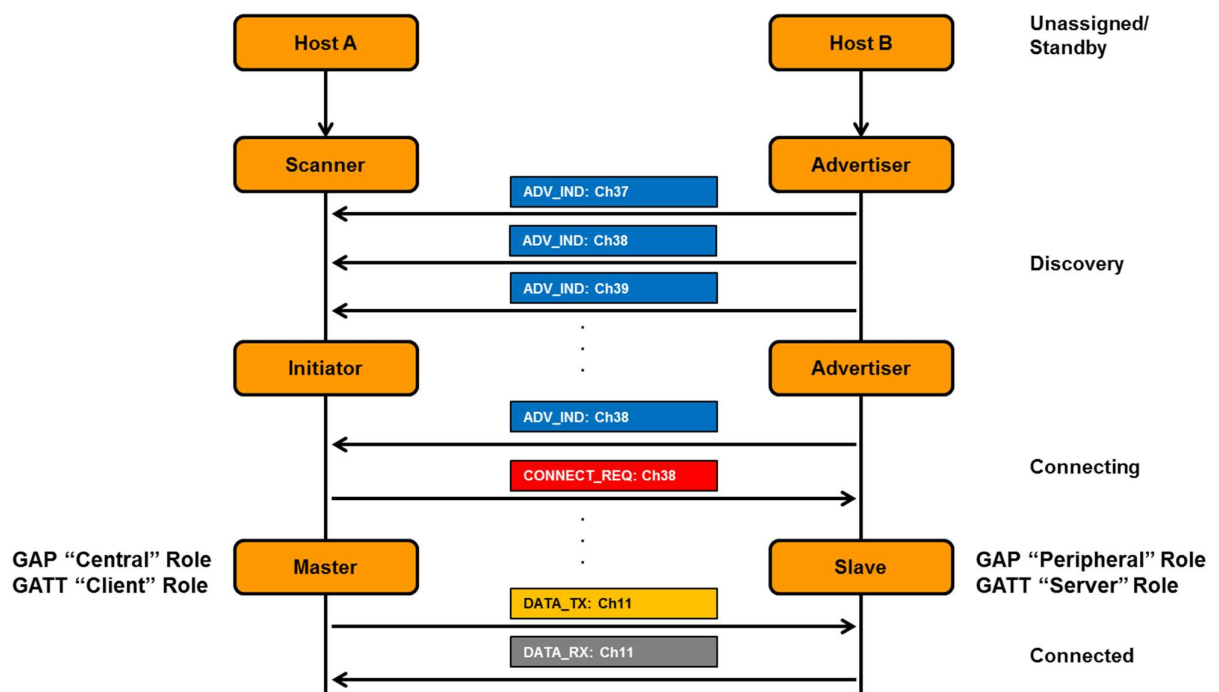


Fig. 2 Advertising and Data Channel communication flow [BLE6].

A channel is connected when devices are tuned to the same physical channel correlated by an access address and the link layers are synchronized to the frequency and timing. Synchronization between master and slave is controlled by connection event timing performed by the master starting a connection event by transmitting a data channel packet. Connection events are characterised by data packet transmissions on the physical data channel defined for each connection event by the master and slave. Data transmissions only occur during connection events. At least one master data packet is transmitted during a connection event that remains open during the transmission period that can consist of alternating sending and receiving sequences [BLE].

BLE enables low power consumption data transfers during very short connection events of a few milliseconds depending on various factors such as the Protocol Data Unit (PDU) size or required processing time. Contrary to the continuous streaming mode operation of classic Bluetooth BLE is designed to operate in sleep mode when not transmitting and wake up upon periodic connection initiation for small data packets transmissions as illustrated in Fig. 4 [BLE2].

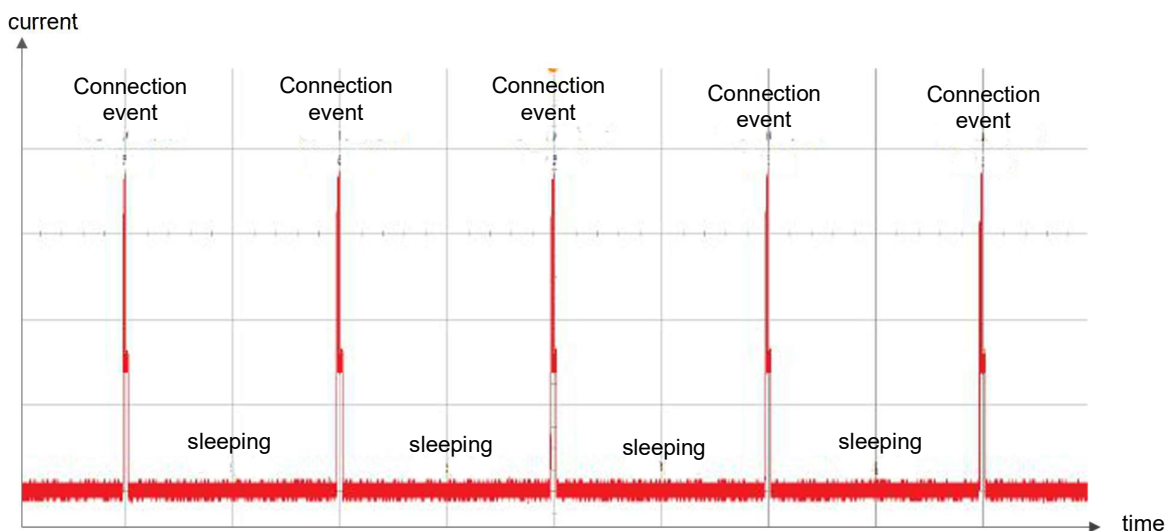


Fig. 4 Current consumption vs time during a BLE connection as in [BLE2].

At each start of a connection event also referred to as anchor point the slave listens to the packet transmitted by the central device. The master controls the connection event timing by scheduling the start of the first connection event. Subsequent anchor points are defined by the connection interval parameter of 1.25 ms multiples between 7.5 ms and 4 s. The slave can optionally skip listening to a number of consecutive connection events set by the value of the slave latency parameter depicted in Fig. 5 [BLE6] if there is no data to be sent.

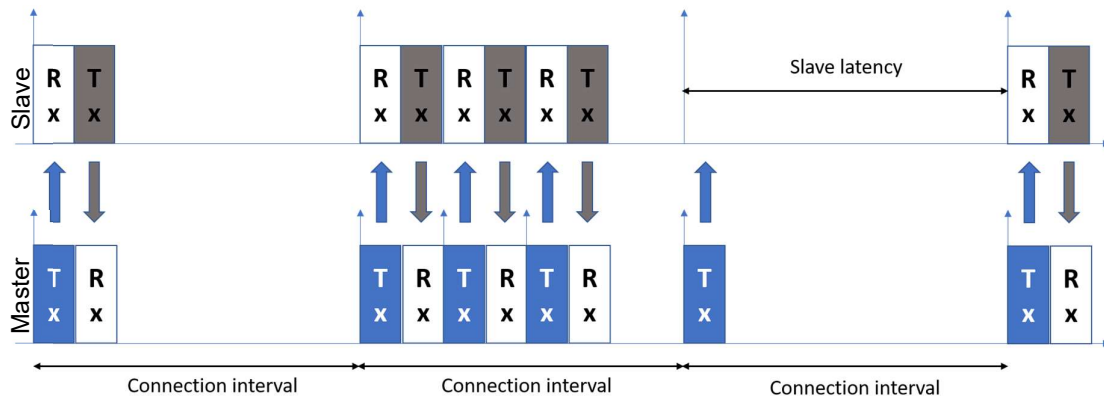


Fig. 5 Slave latency (see [BLE6]).

The slave latency value and the connection interval are efficient timing parameters contributing to increased power savings. Fig. 6 [BLE2]. below indicates a spike in power consumption during connection event initiation leveling out during the wake-up period. The amount of drawn current changes during a connection event with the actual transmission state being the peak current consumer.

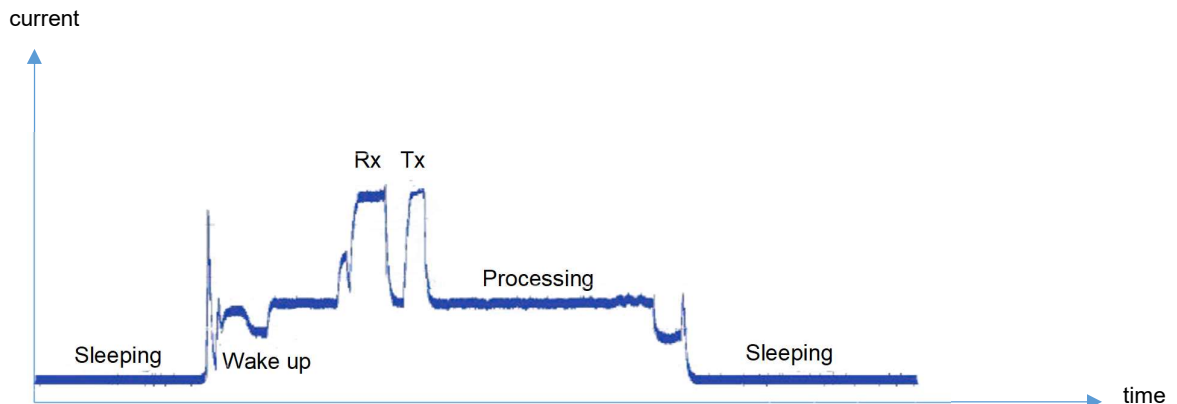


Fig. 6 Current Consumption vs time during a single Connection Event (see [BLE2]).

After processing of the PDU data the connection event can be closed by the device in the Link Layer master role or the slave role when the packet header MD bit of the data channel PDU indicating more data is set to zero.

In addition to the master and slave roles defined by the Link Layer the protocols integrated in the Host layer of the BLE architecture further distinguish behavioral roles illustrated in Fig. 7. The Generic Access Protocol (GAP) defining how devices interoperate allows connection functionality and access to Link Layer operations. GAP provides a connection-oriented Periphera/Central role pair for bidirectional communication and the Broadcaster/Observer connection-less role pair to implement unidirectional communication.

The Broadcaster constantly sends advertising packets embedding accessible data without allowing connections while the Observer passively collects and processes the broadcasted data. The Observer doesn't intend to send any data therefore there is no need for the ability to initiate a connection. The GAP Central on the other hand corresponds to the Link Layer master role and has the ability to connect to the power optimized GAP Peripheral corresponding to the Link Layer slave role. The Peripheral encapsulates device and connection specific information in the read-only GAP Service accessible to all connected devices. The GAP service is a mandatory Generic Attribute profile (GATT) based service. GATT is a framework establishing the exchange of data transported over the stateless transport protocol Attribute Protocol (ATT). Interacting devices can adopt either a GATT client or server role that are completely independent from the roles defined by GAP. The GATT client requests data structured in the GATT database of the GATT server. The GATT server responds to client requests accordingly. Communication initiated by the server can be configured by the client enabling notifications to automatically receive new data values from the GATT server database [BLE].

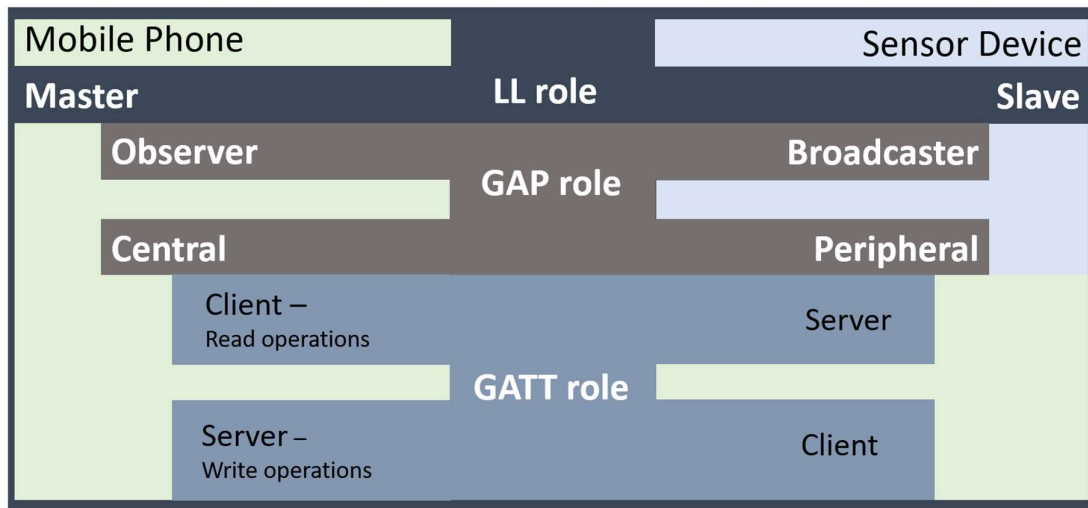


Fig. 7 BLE roles (see [BLE]).

The GATT database is a structured data collection depicted in Fig. 8 [BLE] holding attribute values. Attributes are adressable data containing the actual data and a corresponding data description. A unique 16-bit attribute handle is assigned to address each attribute on a GATT server. The attribute type is defined by a globally unique 128-bit number referred to as Universal Unique Identifier (UUID) as indicated in Table 1 [BLE] representing an example of an attribute’s constitution.

Attribute handle	Attribute type	Attribute value	Attribute permissions
0x0008	“Temperature UUID”	“Temperature value”	“Read only, no authorization, no authentication”

Table 1 Attribute (see [BLE]).

GATT organizes individual attributes into a strict data hierachy referred to as Generic Application Profiles as shown in Fig. 8 [BLE]. Conceptually related attributes are grouped into services. Services contain Characterisitcs holding at least two attributes. The characteristic descriptor attribute provides metadata about the characteristic and its value and the characteristic value attribute contains the actual data value.

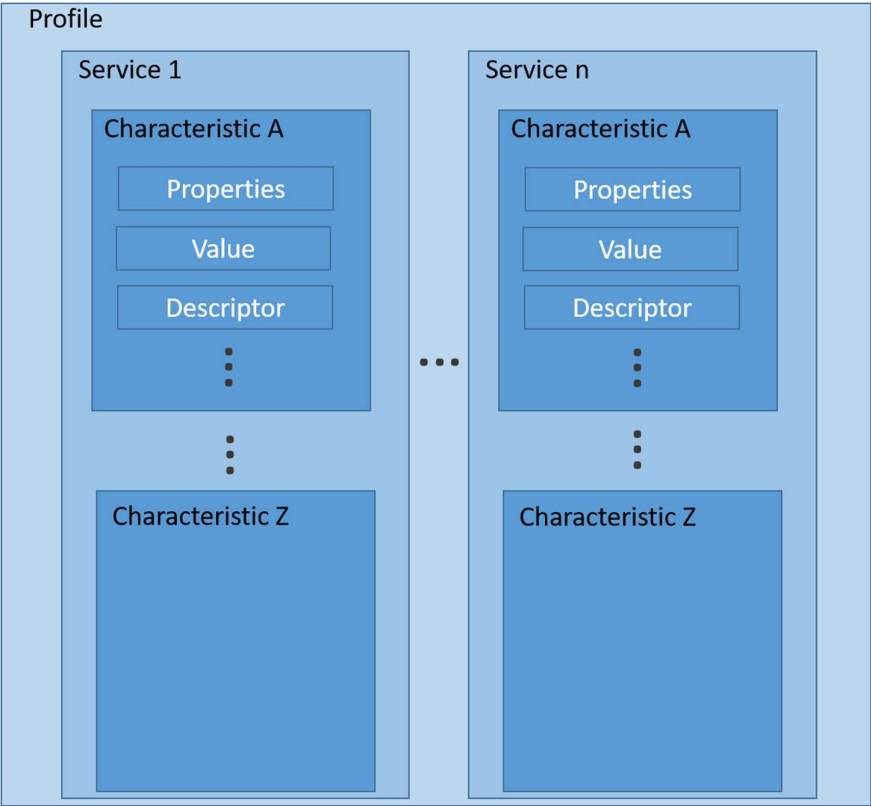


Fig. 8 BLE GATT database (see [BLE]).

Profiles provide an abstraction interface between the application and the BLE protocol stack. The Host Controller Interface (HCI) as shown in Fig. 2 enables communication between the Host and Controller blocks of the BLE protocol stack. The L2CAP Logical Link Control and Adaptation Protocol on top of the HCI layer provides multiplexing, data packet segmentation and reassembly functionalities for data exchange between the Host and Controller elements. The data encapsulation services provided by the L2CAP layer transporting data between the upper and lower layers permits logical end-to-end data communication. The Security Manager (SM) on top of L2CAP defines security features for secure data exchange [BLE].

Implementing the BLE stack on a BLE compliant SoC acting as the BLE peripheral enables wireless low power communication between the Android device and the sensor device.



## 3 SoC Comparison

### 3.1. System architecture overview

#### TI CC2541

The TI CC2541 is a COTS BLE 4.0 compliant SoC optimized for ultra-low power consumption integrated in the HET wristband prototype [T11]. Featuring a proprietary industry RF radio it enables continuous BLE data streaming of sensor readings through appropriate peripherals depicted in Fig. 9 [T11]. 21 of the 23 GPIO pins can be configured as peripheral I/Os. A peripheral I/O mapping is provided demonstrating the available pins for the required peripheral function such as interfacing with sensors. The simplified block diagram in Fig. 9 below shows the main SoC components.

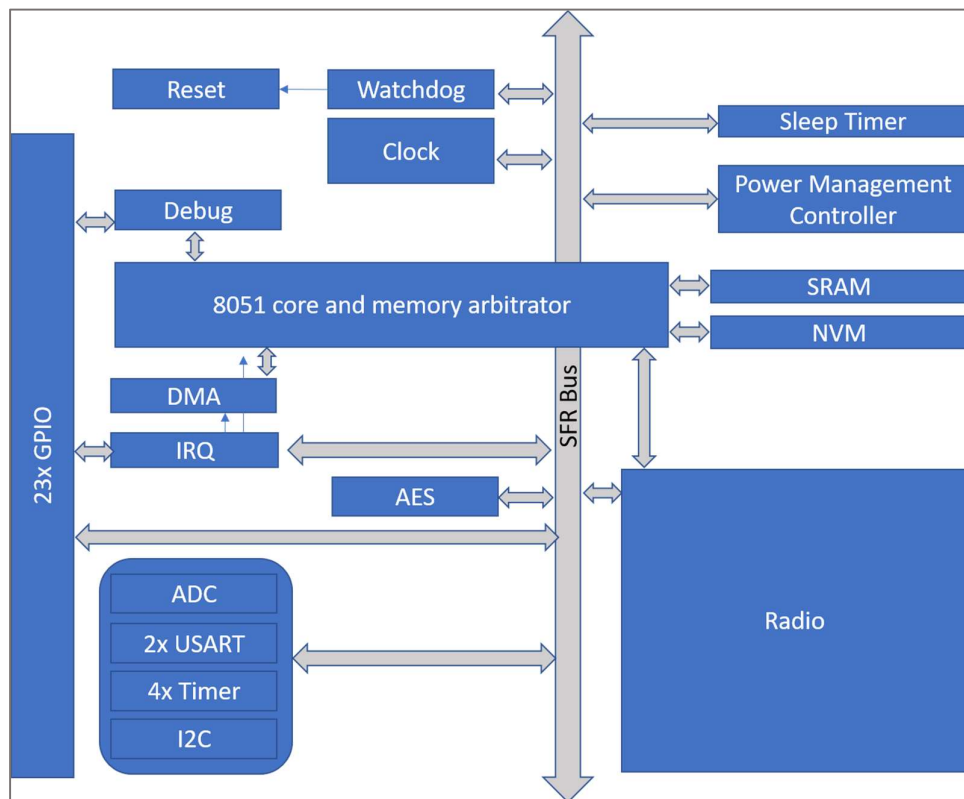


Fig. 9 Simplified TI CC2541 SoC block diagram (see [T1]).

Port 0 provides up to eight input pins for ADC configuration to convert analog input. The multiple operation modes ADC includes eight input channels with an additional input channel for temperature sensing, supports battery measurement and is capable of single conversions and sequence conversions triggering the DMA without any CPU interaction to store up to 14-bit conversions in memory. A decrease in power consumption can be

achieved by utilising the DMA controller keeping the 8051 CPU in low-power mode by moving data between memory and peripherals such as the ADC or the USART [T11].

Two USART serial communication interfaces provide either an asynchronous mode to perform UART operations or the synchronous SPI mode allowing a master-slave communication relationship. The I2C module provided by the SoC CC2541 is another serial communication interface enabling master/slave operations for synchronous data transfer over a two-wire I2C serial bus. An I2C master waits for the I2C bus to be free before initiating the data transfer process including the generation of an interrupt signaling the CPU intervention to start the IRQ service routine. The I2C interrupt P2INT belongs to the same Interrupt Priority Group as the ADC and the Timer T1. Each timer has an assigned interrupt vector [T11].

The 16-bit Timer 1 consists of a 16-bit counter, features five capture-or-compare channels, operates in three different modes and can be configured to generate Infrared (IR) signals for remote control capabilities and IR Learning. Timer 1 together with Timer 3 are used to generate the IR signals. Timer 3 and Timer 4 are 8-bit timers offering two capture/compare channels each. These four modes 8-bit counter timers offer more granulated prescaler values for clock-tick frequency division than Timer 1. Like Timer 1 the maximum clock frequency is 32MHz. There are two high-frequency and two low frequency oscillator clock sources available. The system clock can either be driven by the 32-MHz crystal or by the power efficient 16-MHz RC oscillator. Either one of the low frequency oscillators can be used to set the 32-kHz clock rate for the 15-bit Watchdog Timer and the 24-bit Sleep Timer which continues running without interruptions. The sleep timer sets the low-power mode periods during which the system clock is shut down. During this time the Sleep Timer ensures that timing is maintained for the 40-bit Timer 2, which keeps time for the BLE Link Layer controlling the RF state [T11].

The integrated BLE compliant radio transceiver is accessed through API calls to the TI BLE stack as direct application access to the RF core controlling the radio modules is only permitted when operating the radio in proprietary mode for data transmission. Various data transfer rates are available for GFSK and MSK modulation formats ranging from 250 kbps to 2 Mbps. FIFOs are used for data transport between the MCU and the radio and the DMA can be set up for data transfer between memory and the radio. Radio access is provided by the SFR bus connecting the CPU and the two channel DMA controller with peripherals and memory. The MCU memory spaces are mapped to memory-mapped registers, a non-volatile 245kb Flash program memory and an 8kb SRAM memory block which retains its content in all power modes. Five modes of operation are available to manage power consumption [T11].

## ASSIST ULP SoC

Research Thrust IV integrates innovative power management technologies critical to self-powered wearable systems to enable energy harvesting sensing, sensor data computation, data storage and wireless communication [HET4]. The system block diagram in Fig. 10 [HET4] below depicts a highly integrated self-powered SoC architecture developed by the Thrust IV research team.

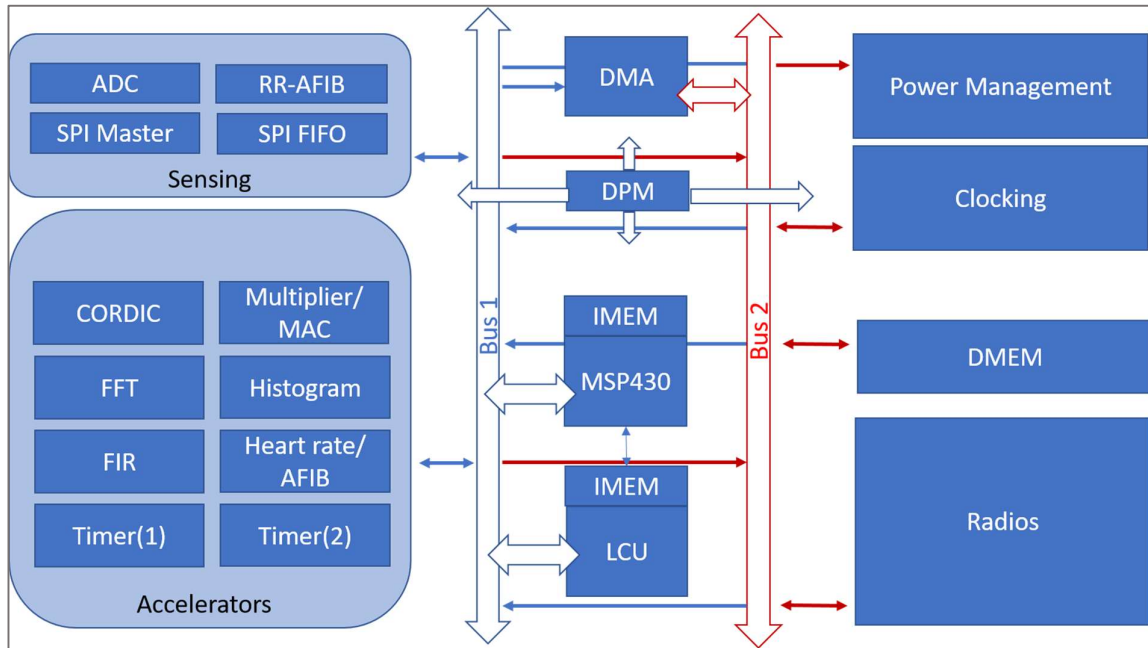


Fig. 10 ASSIST SoC system block diagram (see [HET4]).

It shows a Power Management Unit (PMU) integrating energy harvesting from ambient energy sources such as thermoelectric generators (TEG). The low output voltage supply generated by TEGs is boosted up to the required voltage by the Boost Converter to enable system operations. Before system operations are performed the digital power management (DPM) unit monitors available voltage levels of an off-chip storage capacitor charged by the Boost Converter, controls power options for peripherals and changes system modes based on voltage thresholds. The 8-bit ADC can be used for voltage sampling to monitor for operating mode thresholds while receiving data over the SPI Interface. SPI sensor communication is triggered by a Timer 1 system interrupt [HET4].

The SoC provides two independent general purpose timers with Capture/Compare capabilities. Each timer can interrupt the integrated Lightweight Control Unit (LCU) used for SoC data management and node control during off-mode periods of the OpenMSP430 (OMSP) [OMSP] [HET7]. LCU and OMSP have access to the clocking module for SoC reset and clock frequency configurations. Three system clock sources can be configured to drive the system clock from the on-chip clocking unit. An external clock, an on-chip low power 31.25kHz crystal oscillator or an integrated ultra-low power custom designed programmable all digital phase locked loop (ADPLL) receiving its reference from the on-chip oscillator. The ADPLL frequency ranges between 187.5 kHz and 500 kHz [HET4].

A free running 4GHz ring oscillator can be used to provide a center frequency of 3.994 GHz of a 500MHz bandwidth for the Ultra-Wideband (UWB) transmitter of the integrated asymmetric ULP RF transceiver including a low power wakeup receiver (WuRx) to perform wireless high-data rate transmissions [HET4].

On-chip data transfer is managed by the DMA. Bus 2 is used by the DMA to avoid using the OMSP/LCU controlled Bus 1 when moving data between peripheral blocks and on-chip memory blocks. The memory blocks is comprised of a 4kb data memory, a 2kb memory block dedicated each to a radio TX buffer, LCU instructions and OMSP instructions [HET4].

## **3.2. ULP related features**

### **TI CC2541**

The standard 8051 instruction set used for the TI CC2541 CPU, however due to the instruction cycle memory fetch alignment the standard 12 clocks cycle can be reduced to a single clock cycle increasing power consumption improvements due to increased speed of execution [T11]. Caching instructions by enabling flash prefetching can further increase energy savings as faster access to flash memory is granted. Changing the flash memory space to accommodate SRAM into the CODE memory space results in further power savings by enabling code execution from the ultra-low power SRAM. Mapping parts of DATA memory into the XDATA memory space is another aspect for improving energy consumption by enabling the DMA controller to transfer data between 8051 memory blocks. Accelerating the speed of moving data blocks between memories is achieved by the architectural ULP enhancing introduction of two data pointers as opposed to using only one data pointer. Memory access points are provided by the memory arbitrator managing CPU and DMA access. Utilizing the DMA controller allows the CPU to stay in low-power modes managed by the Power Management Unit. Additional power consumption aspects are the short transition times between power modes. ULP power mode is entered when clock gating is used and the voltage supply to digital modules as well as the oscillators driving the system clock are turned off to avoid leakage and dynamic power consumption. During active mode the system clock can derive its clock source from the high frequency RC Oscillator resulting in further reductions of power consumption, however the 32-MHz crystal oscillator is required when operating the RF transceiver [T11].

### **ASSIST SoC**

The configurable system clock of the ASSIST SoC is driven by the ultra-low power ADPLL clock generator consuming only 300nW by eliminating the divider to significantly reduce power consumption [HET10]. The ADPLL receives its reference from the 31.25-kHz crystal oscillator requiring only 29nW in its low power state [HET4]. The digital components operating in sub-threshold run on the 0.5V rail provided by the PMU supplying 1.2V and 0.5V power rails regulated by a single-inductor multiple-output DC-DC converter for power delivery. The power consumption of the power-gateable memory arrays can drop down to 0.35V. Sub-threshold operations are enabled by power-gateable memories consisting of eight transistor (8T) SRAM bitcells with reduced static and dynamic power consumption and

performing read before write operations. The two channel DMA efficiently performs data transfer between memories and peripherals on Bus 2 reducing power consumption by unburdening the OMSP or LCU. The LCU is the default controller for Bus 1, however, the controller configuration can be changed by the LCU to assign the main bus controller function to the OMSP. The ULP optimized OMSP 16-bit RISC microcontroller is ASIC suitable enabling energy-efficient processing for sub-threshold operations. ULP operations for wireless RF communication and high-data rate transmissions are enabled by utilizing asymmetric RF communication, a custom designed low power wakeup receiver (WuRx) operating on a narrowband RF downlink and the application of OOK modulation decreasing the power consumption of the UWB transmitter and the WuRx. A motion detection sample application using the described SoC architecture for digital data processing of motion sensor data received through SPI and wirelessly streaming the data over the integrated UWB demonstrates a total of 6.45  $\mu\text{W}$  power consumption [HET4].

A total power consumption of 507 nW is achieved with a different SoC architecture integrated in a ULP system-in-package (SiP) as illustrated below in Fig. 11 [HET5].

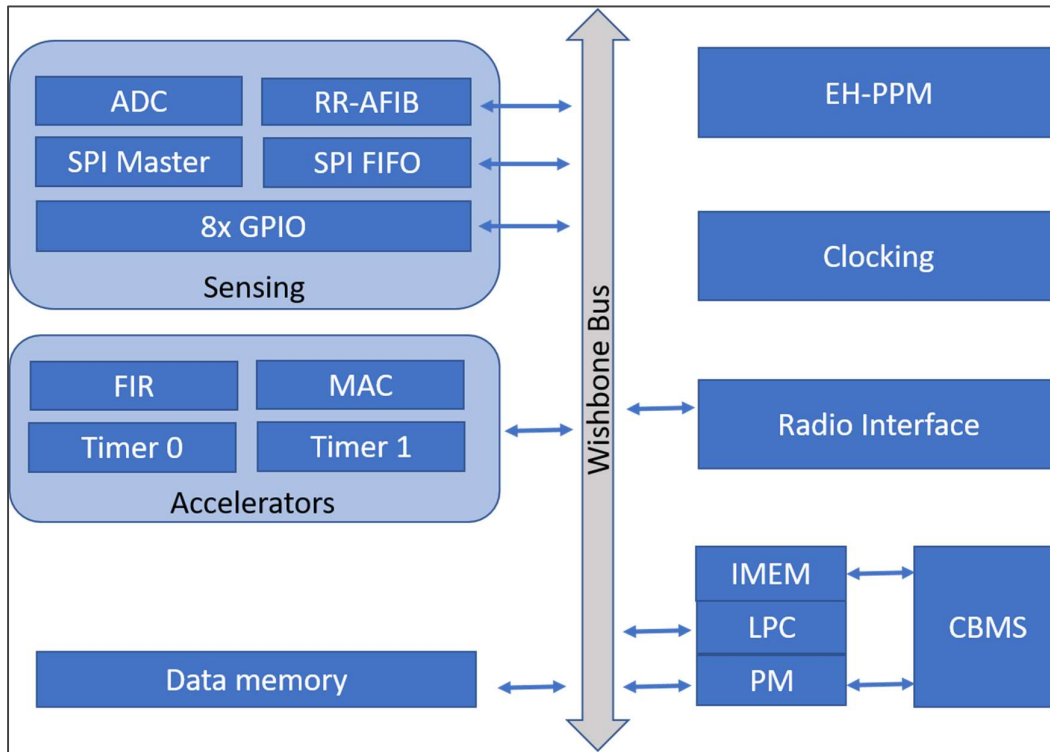


Fig. 11 ASSIST 507nW SoC block diagram (see [HET5]).

The total value includes power consumptions of the on-chip components SPI, GPIO, Timer, IO, MCU and Radio interface used for an application tracking shipping-integrity. Fig. 11 shows the main building blocks of the ULP SoC including an energy harvesting unit powering the SoC, off-chip sensors as well as the in-package 1 Mb/s FSK transmitter and NVM. The SoC is cold-booted from the NVM by the power monitor (PM) component using the cold-boot management system (CBMS) for optimal NVM integration. The PM and the CBMS make up the control block together with the RI, a custom low power controller (LPC)

and the instruction memory (IMEM). The power saving modes of the 8T bitcell SRAM significantly reduce the power consumption of the control block.

### 3.3. Comparison

Major achievements in power consumption reduction measurements result from the proprietary SoCs design proposed by ASSIST [HET8] [HET9]. The integration of a low power 16-bit RISC OMSP430 as suggested in Fig. 10 or a custom low power controller as in Fig. 11 are one of the contributing power reduction factors compared to the enhanced 8051 microcontroller used by the TI CC2541 SoC.

Communication between the 8051 microcontroller unit and DMA with peripherals and memory is accomplished through an SFR bus as opposed to the two 16-bit busses architecture applied to the ASSIST design as in Fig. 10 or the 16-bit Wishbone bus architecture used for circuit communication as depicted in Fig. 11.

Memory access through the SFR bus of the CC2541 is facilitated via a memory arbitrator. The architectural design of the ASSIST SoC does not include a separate memory arbitration component. The memory arbitrator access points of the CC2541 can map to physical memories such as the on-chip flash memory or the SRAM block. Data transfer between physical memories can be performed by a five channel DMA controller integrated on the CC2541 and the two channel DMA of the ASSIST SoC architecture as shown in Fig. 10.

DMA triggers of the TI SoC can occur upon ADC conversions executed by the integrated ADC supporting 7-12 bits of resolution is capable of single and sequence conversions and operating in multiple modes provides eight input channels as well as temperature sensing capabilities. The ASSIST SoC design shown in Fig. 11 provides a 12-bit ADC. Data obtained via the proposed 8-bit four channel ADC of the ASSIST as illustrated in Fig. 10 can serve as input to the DPM unit.

The CC2541 integrates a Power Management Controller (PMC) to provide management of available power modes. Power Management of the ASSIST SoC architecture is provided by the Power Management Unit shown in Fig. 10 and the Platform Power Manager in Fig. 11. The PPM is part of the SoC EH-PPM unit powering the SoC system and components of the SiP. The SoC system power is adjusted by the on-chip Power Monitor (PM) as shown in Fig. 11. In addition to power monitoring the PM includes a cold-boot capability to startup the system from the off-chip system-in-package non volatile memory component.

Communication between SiP and SoC components is enabled by custom digital SPI and GPIO interfaces. While the ASSIST SoC architectures show an integrated SPI component the TI CC2541 additionally provides an I2C module for serial communication. The digital I2C master and slave communication requires two of the 23 GPIO pins. Fig. 11 indicates 8 GPIO pins compared to the total number of the TI CC2541 GPIO pins.

The TI SoC designates a number of GPIO pins to provide timer functionality. The peripheral set of the TI SoC is comprised of a sleep timer, a watchdog timer and four timers with capture and compare capabilities. Two capture/compare timer modules are integrated in the ASSIST SoC as illustrated in Fig. 10 [HET10].

The internal system clock frequency used for the TI SoC timers can be generated by an internal clock source. Two high-frequency oscillators and two low-frequency oscillators are available for clock generation. The clock frequency used by the ASSIST SoC timers can be derived from an external clock source, the internal low frequency crystal oscillator or the integrated ULP ADPLL.

A 4GHz ring oscillator provides the center frequency for the integrated asymmetric ULP RF transceiver as shown in Fig. 4 using OOK modulation for data transmission at 187.5kbps as opposed to the GFSK and MSK modulation format supported by the TI SoC radio transmitter achieving data rates between 250 kbps to 2Mbps. The SiP depicted in Fig. 11 includes an FSK transmitter for up to 1Mb/s wireless data transfers.

## **4 Methodology and Implementation**

The implementation of the wireless communication user interface on Android applies methodological considerations in accordance to the Software Design Waterfall Model commencing with the requirements specification phase followed by the design phase and the implementation phase subsequently.

### **4.1. Requirements specification**

The requirements are specified in the SRS document and briefly outlined below.

#### **Product Scope**

The scope of this implementation is to provide data aggregation support to the ASSIST HET project and graphical visualization of physiological & environmental sensor data transmitted via BLE.

#### **Hardware Requirements**

The HET wrist watch developed by the ASSIST center shall provide the respective sensor data transmitted to a BLE enabled smartphone.

#### **System Interfaces**

Following interfaces are defined:

- Sensor device
- User interface

## Functional Requirements

Identified use cases are listed below and depicted in the use case diagram provided in Appendix A.

- Use case: Start BLE  
The system must allow the user to start BLE including the scan for the BLE wrist device, connecting to the found device, discovering the BLE services, reading and writing BLE characteristics and enable notifications.
- Use case: Visualize real time data  
The system must allow the user to view a graphical display of real time data.
- Use case: Stream data  
The system must allow to stream data autonomously to a dedicated server.
- Use case: Upload data file  
The system must allow the user to upload a csv file of raw data to a dedicated server.
- Use case: Delete data file  
The system shall allow the user to delete stored csv files.
- Use case: Disconnect  
The system shall allow the user to disconnect the BLE device.

## Non-functional Requirements

The following system constraints apply:

- *Security & Privacy*  
The scope of the ASSIST research project does not include any security nor privacy related considerations.
- *IDE*  
Android Studio Version 2.3 is the preferred development environment for implementation. The minimum Android API level is 24 Nougat.
- *Usability*  
Ease of learning is facilitated through an intuitive and structured user interface.

## Requirements Prioritization

The MoSCoW prioritization scheme is applied defining the following use case priorities:

- Must
  - Start BLE
  - Visualize real time data
  - Upload data file
- Should
  - Stream data
- Could
  - Delete data file
  - Disconnect
- Won't
  - User Authentication
  - Security

Must and Should use cases are elaborated in Appendix B.



## 4.2. Design specification

### Communication design

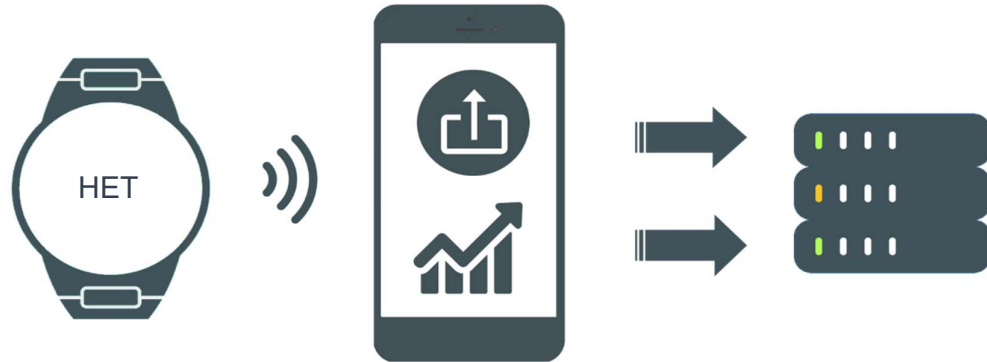


Fig. 12 Communication flow

Fig. 12 demonstrates the communication flow from the BLE sensor device to the dedicated server via the Android device. As outlined in the functional requirements the implementation of the HET app includes two communication paths. The BLE communication between the sensor device enables the exchange of BLE data. The communication between the Android phone and the dedicated server enables the transfer of BLE transmitted sensor data. The server port 22 receives SFTP file uploads and the Forwarding Port 5672 is activated for data streaming with RabbitMQ.

### UML modelling

The class diagram attached in Appendix C shows the main classes enabling the implementation of the use cases defined in the functional requirements. The MainActivity.java class holds functions to bind to the BLE service class and starts it as a background service. The service class provides BLE functions to the user application and contains BLE callback functions indicating the results of the respective BLE operations. Received sensor data is displayed in real time directly in the MainActivity class. The Chart2Activity class is used to render static charts using fragment classes. In addition to the chart rendering functionality sensor data is stored in csv files and can be uploaded to the dedicated server using the SSHActivity.java class. Data streaming functions are provided by the background service ESPservice.java started by the MainActivity and fed with data by the BLE service class. Event classes such as EventFileName are implemented for communication purposes between activities.

### User interface design

Appendix D includes user interface (UI) mock ups to illustrate the design supporting an intuitive user experience. The start screen displays the BLE connection sequence indicated in numerical order subsequently enabling the kickstart button and characteristic notification switches. Graphical user buttons for additional chart renderings and file uploads allow the user to navigate to the respective screens. The chart screen displays tabs for each fragment

including corresponding icons and allows swiping motions to switch between fragments. A bottom navigation bar provides labelled icons to navigate to the start screen. The file upload icons take the user to the upload screen providing toast messages to indicate the transmission progress and automatically returns to the start screen upon completion of the file transfer. An overflow menu on the top right provides the possibility to disconnect the BLE connection and delete stored csv files.

## 4.3. Implementation

### 4.3.1. Introduction

As illustrated by the mock ups of Appendix D the UI design of the implementation of the HET app includes data visualization and data transmission. The following components are deployed to implement the use cases of the HET application:

- **SciChart**  
The SciChart API is a high-performance Android charting library used to render real time data charts [SCI].
- **RabbitMQ**  
RabbitMQ is a message broker software supporting the Advanced Message Queuing Protocol (AMQP) using the assigned port number 5672 [RMQ].
- **Cisco AnyConnect Secure Mobility Client**  
The Cisco AnyConnect provides a VPN connection to the NCSU network [CISCO].
- **Termius**  
The Termius app is a SSH/SFTP client supporting Port Forwarding [TERM].
- **JSch**  
The Java Secure Channel (JSch) API allows SFTP file transfer. [JSCH].
- **EventBus**  
The open-source library by greenrobot enables communication based on the Publisher/Subscriber design pattern [EVENT].

#### Flow chart

The flow chart depicted in Fig. 13 shows the user interaction sequence enabling data visualization and data transmission. When the application is started the system checks if Bluetooth is turned on and access to the device location is granted. Once the Start button is pressed the BLE connection process is initiated and the Kickstart button is enabled as soon as BLE services are discovered. Pressing the Kickstart button enables the characteristic 3 notification switch. After the notification flag for characteristic 3 has been successfully set the characteristic 4 switch is enabled. As soon as notifications for both characteristics have been registered BLE sensor values are received and displayed in real time on the start screen of the application.

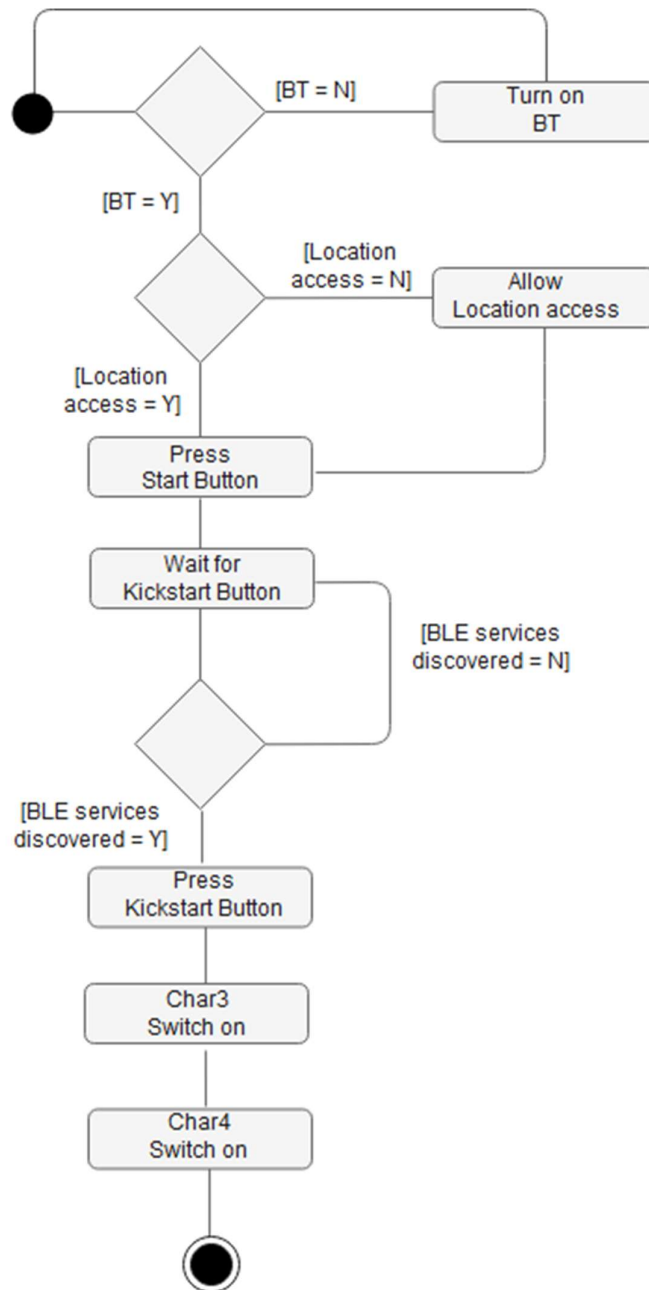


Fig. 13 Flow chart user interaction.

#### 4.3.2. BLE Peripheral device

The firmware implementation of the BLE sensor device is performed by the HET ASSIST research group. The current peripheral implementation does not include any BLE security features such as encrypting the link between the peripheral device and the central device. The Android phone receives byte values from the BLE peripheral sensor device. The HET watch BLE implementation defines one HET service containing three relevant characteristics listed below.

- Characteristic 1 is used to kickstart characteristic 3 and characteristic 4 by writing the HEX value 0x01.
- Characteristic 3 consists of 10 bytes holding four different ozone values of 2 bytes each and the value for the battery.
- Characteristic 4 is composed of 14 bytes holding three WAV, accelerometer, temperature and humidity values.

Fig. 14 indicates the composition of the characteristics 3 and 4 as elaborated above.

uuid - char3																						
oz1	oz2	oz3	oz4	battery																		
1   2	3   4	5   6	7   8	9   10																		
uuid - char4																						
wav1	wav2	wav3	blank	x	y	z	temp1	temp2	humid1	humid2												
1   2	3   4	5   6	7	8	9	10	11	12	13	14												

Fig. 14 Characteristic 3 and characteristic 4

### 4.3.3. BLE Central device

The BLE data transmission of the characteristics illustrated in Fig. 14 is enabled by implementing key BLE functionalities.

#### 4.3.3.1. Start Bluetooth

The BLE support introduced with Android API level 18 allows a BLE central device to scan for nearby peripheral devices, discover BLE services, establish and disconnect a BLE communication and transmit BLE specific information. Listing 1 shows the permissions needed to be declared in the manifest file of the application to enable BLE functionalities.

```
1 <uses-permission android:name="android.permission.BLUETOOTH" />
2 <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Listing 1 BLE permissions.

BLE specific functions are implemented in the BLEservice class extending the application component Service. The setOnClickListener of the Start Button in the MainActivity.java includes the startBluetooth() function instantiating an Intent to bind to the BLEservice class. A bound service allows the MainActivity class to interact with received responses from the service class. The MainActivity instantiates and initialises the BroadcastReceiver overriding the onReceive() method with a switch case for specific messages sent by the broadcast update of the BLE service class.

```

1 private final BroadcastReceiver mBleUpdateReceiver = new BroadcastReceiver() {
2     @Override
3     public void onReceive(Context context, Intent intent) {
4         final String action = intent.getAction();
5
6         switch (action) {
7             case BLEservice.ACTION_BLESTARTED:
8                 if (mServiceConnected) {
9                     mBLEservice.scan();
10                }
11                break;
12
13             case BLEservice.ACTION_BLESCAN_CALLBACK:
14                 mBLEservice.connect();
15                break;
16            .
17            .
18            .

```

Listing 2 onReceive() method of the BroadcastReceiver.

The onResume() method of the MainActivity in Listing 2 registers the Broadcast receiver. The BroadcastReceiver object is passed to the registerReceiver() method along with an Intent filter for Intents sent in the sendBroadcast() method by the BLE service class.

```

1 @Override
2 protected void onResume() {
3     super.onResume();
4     final IntentFilter filter = new IntentFilter();
5     filter.addAction(BLEservice.ACTION_BLESTARTED);
6     .
7     .
8     filter.addAction(BLEservice.ACTION_DATA_RECEIVED);
9     registerReceiver(mBleUpdateReceiver, filter);
10 }

```

Listing 3 onResume() method of the MainActivity.java class.

The first broadcast is sent after successfully obtaining a BluetoothAdapter from the initialized BluetoothManager – a procedure called in the onServiceConnected() method of the MainActivity as shown in Listing 4.

```

1 mBluetoothManager = (BluetoothManager)
2 getSystemService(Context.BLUETOOTH_SERVICE);
3
4 mBluetoothAdapter = mBluetoothManager.getAdapter();

```

Listing 4 Bluetooth adapter initialization.

This broadcast update causes the scan procedure to be called on the BLEservice class object of the MainActivity. The scan function in Listing 5 filters for the BLEdevice containing the BLE HET service.

```
1 mLEScanner.startScan(filters, settings, mScanCallback);
```

Listing 5 startScan() method call.

The callback function of the startScan() function in the BLEservice class gets the BLEdevice and broadcasts that it found the desired HET device. Upon receipt of the BLE scan callback broadcast the connectGatt() function is called on the BLE device initializing the BluetoothGatt object in the BLEservice class as shown in Listing 6.

```
1 mBluetoothGatt = mLeDevice.connectGatt(this, false, mGattCallback);
```

Listing 6 BluetoothGatt object initialization.

The BLE Gatt callback broadcasts the connection state to the Broadcast Receiver, which initiates the discoverServices() in Listing 7. This method is called on the BluetoothGatt object upon receipt of the broadcast update when a connection has been established .

```
1 mBluetoothGatt.discoverServices();
```

Listing 7 discoverServices() method call.

The result of the discoverServices() method invokes the onServicesDiscovered callback. The callback gets three characteristics from the HET BluetoothGattService based on their UUIDs as shown in Listing 8.

```
1 BluetoothGattService mService =  
2 gatt.getService(BLEdefinedUUIDs.Service.HET_SERVICE);  
3  
4 mChar1Characteristic =  
5 mService.getCharacteristic(BLEdefinedUUIDs.Characteristic.CHAR1);
```

Listing 8 getCharacteristic() method call.

Once the respective characteristics have been assigned the Broadcast Receiver is informed that services have been discovered. The Broadcast Receiver then enables the Kickstart button which writes a 0x01 value to the characteristic 1 shown in Listing 9 to allow the subscription to characteristic 3 and characteristic 4 notifications.

```
1 mChar1Characteristic.setValue(byteVal);
2 mBluetoothGatt.writeCharacteristic(mChar1Characteristic);
```

Listing 9 writeCharacteristic() method call.

The next step is to enable notifications by putting the switch for the characteristic 3 in the on-position. This sets the characteristic notification flag to true, sets the BluetoothGattDescriptor value to enable notification and passes the set Client Characteristic Configuration Descriptor UUID to the writeDescriptor() method called on the BluetoothGatt object as shown in Listing 10.

```
1 mBluetoothGatt.setCharacteristicNotification(mChar3Characteristic, value);
2
3 mCccd.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
4 mBluetoothGatt.writeDescriptor(mCccd);
```

Listing 10 writeDescriptor() method call.

Enabled notifications trigger the onCharacteristicChanged() callback when new values are received.

The received bytes are assigned to a byte array, buffered into a string variable holding the respective HEX value shown in Listing 11 and a broadcast update informs the MainActivity that new data has been received.

```
1 public void onCharacteristicChanged(BluetoothGatt gatt,
2                                     BluetoothGattCharacteristic characteristic) {
3
4     String uuid = characteristic.getUuid().toString();
5     switch (uuid){
6         case CHAR3UUID:
7             final byte[] data = characteristic.getValue();
8             StringBuffer buffer = new StringBuffer();
9             for(int i=0; i < data.length; i++){
10                 buffer.append(Character.forDigit((data[i] >> 4) & 0xF, 16));
11                 buffer.append(Character.forDigit((data[i] & 0xF), 16));
12             }
13
14             String result = buffer.toString();
15             mChar3ValueFull = result;
```

Listing 11 onCharacteristicChanged() method.

#### 4.3.3.2. Visualize real time data

The received data is retrieved as shown in Listing 12 and processed in the BroadcastReceiver of the MainActivity class.

```
1 String char4valueFull = mBLEService.getMChar4ValueFull();
```

Listing 12 Data value assignment.

The substring() method is called on the received HEX value to extract the respective values such as ozone related values. These string values are then parsed to Integers and added to an ArrayList. The corresponding ArrayList object is passed to a function that updates the data visualization chart. The chart surface is added to the LinearLayout as shown in Listing 13.

```
1 surface = new SciChartSurface(this);  
2 chartLayout.addView(surface);
```

Listing 13 Adding a chart surface to the LinearLayout.

The SciChartBuilder helper class is used to initialize chart objects such as Axes or DataSeries objects as demonstrated in Listing 14 below.

```
1 final IAxis xAxis = sciChartBuilder.newNumericAxis()  
2     .withAxisTitle("Ozone")  
3     .build();  
.  
.  
6 lineData = sciChartBuilder.newXyDataSeries(Integer.class, Integer.class)  
7     .withFifoCapacity(fifoCapacity)  
8     .build();
```

Listing 14 Chart objects initialization.



The `lineData` object receives the new data points stored in the `ArrayList` holding the new data values in a `TimerTask` running a `SciChart UpdateSuspend` method to append each new data point to the `lineData` object demonstrated in Listing 15.

```
1 TimerTask updateDataTask4 = new TimerTask() {
2     private int x = 0;
3     @Override
4     public void run() {
5         UpdateSuspend.using(surface, new Runnable() {
6             @Override
7             public void run() {
8                 lineData.append(x, value.get(x));
9                 surface.zoomExtents();
10                ++x;
11            }
12        }
13    }
14 }
```

Listing 15 Appending new chart data values.

The `SciChartBuilder` helper attaches the `lineData` to the `RenderableSeries` object as shown in Listing 16.

```
1 final IRenderableSeries lineSeries = sciChartBuilder.newLineSeries()
2     .withDataSeries(lineData)
3     .withStrokeStyle(ColorUtil.LightBlue, 2f, true)
4     .build();
```

Listing 16 `RenderableSeries` initialization.

Then the `RenderableSeries` instance is added to the chart surface shown in Listing 17.

```
1 surface.getRenderableSeries().add(lineSeries);
```

Listing 17 Adding data to `RenderableSeries`.

The procedure outlined above renders the new data values on the chart surface.

#### 4.3.3.3. Stream data

Simultaneously new data values are streamed to the dedicated server using the message broker software `RabbitMQ` supporting the network protocol `AMQP`. `AMQP` enables the communication between publisher, consumer and `AMQP` message broker. The `RabbitMQ` Java Client Library requires the manifest declaration for Internet access to allow a connection establishment shown in Listing 18.

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

Listing 18 Internet access permission for `RabbitMQ`.

Two apps enable the connection establishment:

- Cisco AnyConnect  
VPN tunneling is established using the Cisco AnyConnect app to gain access to the NCSU network.
- Termius  
The server host access is configured to run a terminal session with SSH and activate a port forwarding rule for the RabbitMQ port 5672.

A RabbitMQ ConnectionFactory object is initialised and set with the host, port, username and password parameters in the onStartCommand of the ESP Service class.

Connection establishment is performed on a thread with a reconnection interval of 5 seconds. A new RabbitMQ connection object is instantiated in this thread using the ConnectionFactory object as demonstrated in Listing 19.

```
1 publishThread = new Thread(new Runnable() {
2     @Override
3     public void run() {
4         while(true) {
5             try {
6                 Connection connection = factory.newConnection();
7                 Channel channel = connection.createChannel();
8                 while (true) {
9                     String message = queue.takeFirst();
10                    try{
11                        channel.basicPublish("sasesp", "chatter", null,
12                                           message.getBytes());
13                        channel.waitForConfirmsOrDie();
14                    } catch (Exception e){
15                        queue.putFirst(message);
16                        throw e;
17                    }
18                }
19            } catch (InterruptedException e) {
20                break;
21            } catch (Exception e) {
22                try {
23                    Thread.sleep(5000);
24                } catch (InterruptedException e1) {
25                    break;
26                }
27            }
28        }
29    }
30 });
31 publishThread.start();
```

Listing 19 RabbitMQ publishThread.

The MainActivity starts the background service in the onCreate() method as shown in Listing 20.

```
1 Intent intent = new Intent(this, ESPService.class);
2 startService(intent);
```

Listing 20 Starting ESP Service.

The publishThread is started in the onStartCommand() method of the ESPService Class. The queue used in publishThread shown in Listing 19 is fed with new BLE data in the onCharacteristicsChanged() method of the BLEservice class.

#### 4.3.3.4. Upload to server

The onCharacteristicChanged() method additionally adds the new values to an ArrayList of Strings in the BLEservice class that is used to create a file for uploading the values to the server in a csv file format. Two Buttons are provided by the MainActivity to create the csv file for the respective characteristic putting the file path on the EventBus. The communication across components using the EventBus implementation is straight forward requiring the following steps:

- Event definition with POJOs (Plain Old Java Object) shown in Listing 21.

```
1 public class EventFilePath {
2     public final String StringValue;
3
4     public EventFilePath(String StringValue) {
5         this.StringValue = StringValue;
6     }
7 }
```

Listing 21 EventBus class.

- Publishing component:
  - Post an Event in the publishing component shown in Listing 22.

```
1 EventBus.getDefault().postSticky(new EventFileName(mFileName));
```

Listing 22 Event publishing.

- Subscribing component:
  - Register the subscriber in the onCreate() method of the receiving component as in Listing 23.

```
1 EventBus.getDefault().register(this);
```

Listing 23 Event subscription.

- Definition of event handling using the Subscribe annotation shown in Listing 24.

```
1 @Subscribe(sticky = true, threadMode = ThreadMode.MAIN)
2 public void onEvent(EventFilePath event) {
3     mFilePath = event.getMessage();
4 }
```

Listing 24 Event handling.

The SSHActivity.java class picks up the path string and executes an AsyncTask to upload the file to the dedicated server outlined in Listing 25.

The AsyncTask instantiates a JSch object to obtain a session object passing username, host and port parameters and setting the password. This session object is used to open and connect a channel and use an SFTP channel to upload the csv file by passing the file path and the destination folder.

After the file upload the channel and the session are disconnected.

```
1 JSch ssh = new JSch();
2 session = ssh.getSession(...);
3
4 channel = session.openChannel("sftp");
5 channel.connect();
6
7 ChannelSftp sftp = (ChannelSftp) channel;
8 sftp.put(mFilePath, "/home/Data/HET_Upload/");
9
10 channel.disconnect();
11 session.disconnect();
```

Listing 25 File upload procedure.

#### 4.3.3.5. Disconnect Bluetooth

The overflow menu provides an option to call the disconnect() method on the BluetoothGatt object shown in Listing 26.

```
1 mBluetoothGatt.disconnect();
```

Listing 26 disconnect() method call.

#### 4.3.3.6. Delete files

The second option of the overflow menu deletes the created csv files that are stored in the internal storage of the application as demonstrated in Listing 27 below.

```
1 File dir = context.getFilesDir();
2 String[] children = dir.list();
3 for (int i = 0; i < children.length; i++) {
4     new File(dir, children[i]).delete();
5 }
```

Listing 27 Deleting csv files.

## 5 Results & Discussion

The use case 'Deleting files' works as expected and frees up internal storage. Due to the requirement of uploading raw unstructured data as transmitted by the BLE peripheral device internal storage provides a sufficient storage solution. It allows the retrieval of stored files for upload to the designated server.

Alternatively, a SQLite database could be implemented providing structured data storage. However, considering that SQL queries are not verified at compile time and manual query changes would be required in the case of schema changes, the use of the Room Persistence Library is recommended [AAC]. Room is an object mapping library that provides an abstraction layer for SQLite eliminating repetitive code while embracing SQLite. The built-in observability support of Room allows for an extension of the persistence use case by wrapping query results in lifecycle-aware LiveData. This allows observing changes of data persisted in the Room database. LiveData is a class designed to hold data active observers can subscribe to so that they can be informed about data changes.

This concept could be applied to inform observers about new BLE data received in the BLE service class. The current implementation consists of a service class that performs BLE operations in the background and communicates data changes to the MainActivity that is bound to the BLE service class to receive broadcast updates. Upon receipt of the broadcast that new data has been received the Main Activity retrieves the new data from the BLE service class as elaborated in Chapters 4.3.3.1 and 4.3.3.2.

Delegating data retrieval and data processing to a repository class would keep the UI Controller free of business logic. LiveData holding ViewModel classes could instruct repositories to fetch data. The data changes can be reflected to the activity associated with the ViewModel and subscribed to LiveData changes as an observer. This architecture provides a clear separation of concern addressing the main concept of Android Architecture Components (AAC). Room, LiveData and ViewModels are the key Android Architecture Components. AAC is a collection of libraries supporting the design of testable and robust applications, managing the lifecycle of UI components and applying data persistence to drive the UI.

Persisting chart data in a ViewModel would particularly prove useful for the 'Visualize data' use case outlined in Chapter 4.3.3.2. The lifecycle awareness of ViewModels provide configuration change support by encapsulating UI related data. ViewModels are scoped to survive state changes of the components holding a reference to the ViewModel. That way charts would still be rendered after screen rotations due to the synchronous communication between View and ViewModel based on subscriptions to LiveData objects without the ViewModel knowing anything about the associated View.

The HET implementation takes advantages of subscriptions to enable asynchronous communication between application components. Instead of passing information using the `putExtra()` method for Intents the greenrobot EventBus is implemented as outlined in Chapter 4.3.3.4. Events are routed based on their class name to disseminate content-based information. The event bus managing the routing of events serves as a means of a logical communication channel which corresponds to the Publish-Subscribe software design pattern indirectly linking Publisher and Subscriber objects. This differentiates the Publish-Subscribe design pattern from the Observer design pattern [SDWM] inherent to LiveData Subject-Observer relationships that notify object status changes directly to observers. Persisting events is another distinguishing feature that EventBus is offering. As outlined in Chapter 4.3.3.4 the implementation of the subscriber has the option to set the annotation attribute `sticky` to true. Hence, event caching is enabled allowing subscribers to pick up the most recent specified event that was posted on the default event bus before the subscriber was instantiated and able to register for events. The `post(Object event)` method of the EventBus class implements a `List<Object>` to add events as a mechanism for event propagation [EVENT].

Queues are key elements of the message broker library RabbitMQ used to asynchronously stream BLE data to the designated server. As elaborated in Chapter 4.3.3.3 the BLEService class adds new values to the internal producer queue to stream data. The publishing thread is created in the ESPService class removing the first item of the queue and passing it to the publishing method of the channel. The channel is created on an AMQP connection object with an underlying TCP connection between the application and the broker. As indicated in Chapter 4.3.3.3 RabbitMQ listens on port 5672 supporting AMQP. The Termius application is used to open an SSH terminal session and enable port forwarding.

The JSch library is used to establish an SSH connection to transfer csv files over the SSH File Transfer Protocol (SFTP) as elaborated in Chapter 4.3.3.4. The JSch library offers a port forwarding feature that could be used to implement the 'Stream data' use case making Termius running in the background obsolete. Termius and the JSch library enable secure data transmissions between the dedicated server and the Android device. While the data transfer between the server and the Android device is secure, Chapter 4.3.2. indicates that the communication link between the BLE Android central device and the peripheral sensor device is not encrypted.

The low energy Secure Connections pairing model introduced with the legacy BLE 4.2 offers effective encryption security features enabling link and data encryption to mitigate identity tracking, man-in-the-middle (MITM) attacks and passive eavesdropping [BLE1]. The energy-efficient cryptography algorithm Elliptic Curve Diffie Hellman is applied for key generation [ECDH] during connection establishment supporting low energy consumption

during data transmission. The peak BLE radio power consumption occurs during the active transmission state of connection events as indicated in Fig. 6 of Chapter 2. Improved power efficiency can be achieved by reducing wireless communication duty cycles. The increased successive interval between intermittent connection events prolong low power mode durations as a result. Due to the short distance between the BLE sensor device and the Android device the transmitter output power can be significantly reduced to gain further power savings [HET2]. Another impact on power consumption reduction is the achieved by increasing the slave latency [BLE2]. This parameter allows the peripheral sensor device to skip connection events effectively increasing low power mode periods.

As indicated above software related adjustments of BLE parameters can be taken into consideration to impact low power features of BLE compliant SoCs providing hardware support for ultra-low power operations. Chapter 3 elaborates specific ULP features of the TI CC2541 currently integrated in the HET wrist watch and ongoing SoC developments of the ASSIST research thrust VI towards self-powered system integration. A comparison highlights key differences of the architectural SoC designs. Custom ULP components integrated in the energy-harvesting ASSIST SoC depicted in Fig. 10 significantly contribute to reducing the power consumption to 6.45  $\mu$ W. Fig. 11 illustrates the tight integration of the 507nW SoC components enabling sub  $\mu$ W operations while interfacing with off-chip SiP components allowing a diverse spectrum of IoT applications.

The development of custom components designed to meet the requirements of low energy performance and wireless communication defined by self-powered wearable ASSIST system platforms enable ultra-low-power operations that can't be achieved with COTS approaches such as the TI CC2541 SoC. While offering a range of ultra-low power features commercially available SoCs are required to provide a broad selection of functionality and a rich set of peripherals to enable the implementation of a wide range of use cases beyond IoT applications.

## 6 Related work

BLE compliant COTS SoCs are continuously optimized to make application designs operating on limited energy sources feasible. Comparative studies of selected SoCs manufactured by various vendors such as the Texas Instruments CC2540 presented in [SOC1] or Nordic Semiconductor SoCs evaluated in [SOC2] focus on BLE performance and power consumption. A systematic approach towards the analysis of peripheral parameters impacting BLE performances of applications supporting wireless communication between a smartphone and a TI CC2540 peripheral is provided in [SOC3].

Contrary to the quantification of BLE power consumption measurements of BLE SoCs presented in existing work this paper provides a comparison of the architectural SoC design and hardware components supporting ULP energy consumption for wireless data transfer of environmental and physiological health sensing parameters.

There has been considerable interest in BLE solutions dedicated to health-related wearable devices transmitting sensor data via wireless BLE communication. Major manufacturers address the increased demand of BLE solutions by developing low power BLE SoCs offering sample projects on Android BLE implementation. The BLE implementation of the

Android user interface presented in this work is based on the Cypress BLE demo project [CYP]. The sample implementation provided by Dialog is comparable to the Cypress sample project implementation. The Dialog Semiconductor SmartBond™ DA14681 Wearable Development Kit includes a wrist watch displaying heart rate and other health related parameters [DIAL]. The Dialog and the Cypress implementations create a BLE service class to enable BLE communication between the BLE peripheral and the Android device. The BLE implementation of the sample project included in the documentation of the Renesas RL78/G1D evaluation board suggests a different approach [REN]. A BLE wrapper class holding BLE requests and a BLE Wrapper Callback interface for request results are implemented to allow BLE communication without the use of a service class.

## 7 Conclusion and Future work

BLE enriches the world of wireless devices. The energy efficiency of this wireless communication protocol makes it ideal for establishing data transmissions between smartphones and wearable sensor devices operating on a restricted energy budget. Low energy radio technology in combination with custom ULP SoC components enable self-powered wearable sensing platforms running on human body harvested power sources. Cutting edge research on the development of an energy-harvesting custom SoC addressing the challenges of ULP energy constraints for self-powered wearable sensing results in substantial power reductions enabling operations in the sub  $\mu\text{W}$  range. The sensing parameters monitored by the wearable HET system aim to provide an understanding of the correlation between environmental exposure and related adverse health responses. Data visualization of these ambient and physiological parameters on an Android device enhances self-efficacy strategies for optimized health care and wellness management. The data aggregation functionality provided by the Android device in terms of data streaming and file uploading for longitudinal data storage further assists data analysis advancing medicine and health management.

Concepts of big data analytics could be addressed in future work with special focus on privacy concerns. An additional aspect to be covered is the implementation of security features preventing identity tracking, passive eavesdropping and MITM attacks and ensuring data integrity. The implementation of the Android User Interface could further be adapted to provide a comparison of the most widely adopted pattern approaches MVC, MVP and MVVM separating the application into logical components. The MVVM pattern could be presented to demonstrate and deep dive into the lifecycle aware Android architecture components introduced by Google.



## List of Figures

Fig. 1 BLE Frequency spectrum (see [BLE4]) .....	10
Fig. 2 BLE Layer architecture (see [BLE5]). .....	11
Fig. 3 Advertising and Data Channel communication flow [BLE6].....	11
Fig. 4 Current consumption vs time during a BLE connection as in [BLE2]......	12
Fig. 5 Slave latency (see [BLE6]). .....	13
Fig. 6 Current Consumption during a single Connection Event (see [BLE2]). .....	13
Fig. 7 BLE roles (see [BLE]). .....	14
Fig. 8 BLE GATT database (see [BLE]). .....	15
Fig. 9 Simplified TI CC2541 SoC block diagram (see [T1]). .....	17
Fig. 10 ASSIST SoC system block diagram (see [HET4]). .....	19
Fig. 11 ASSIST 507nW SoC block diagram (see [HET5]). .....	21
Fig. 12 Communication flow .....	25
Fig. 13 Flow chart user interaction. ....	27
Fig. 14 Characteristic 3 and Characteristic 4.....	28

## List of Listings

Listing 1 BLE permissions. ....	28
Listing 2 onReceive() method of the BroadcastReceiver.....	29
Listing 3 onResume() method of the MainActivity.java class.....	29
Listing 4 Bluetooth adapter initialization. ....	29
Listing 5 startScan() method call. ....	30
Listing 6 BluetoothGatt object initialization. ....	30
Listing 7 discoverServices() method call. ....	30
Listing 8 getCharacteristic() method call. ....	30
Listing 9 writeCharacteristic() method call. ....	31
Listing 10 writeDescriptor() method call. ....	31
Listing 11 onCharacteristicChanged() method. ....	31
Listing 12 Data value assignment.....	32
Listing 13 Adding a chart surface to the LinearLayout.....	32
Listing 14 Chart objects initialization. ....	32
Listing 15 Appending new chart data values. ....	33
Listing 16 RenderableSeries initialization.....	33
Listing 17 Adding data to RenderableSeries. ....	33
Listing 18 Internet access permission for RabbitMQ. ....	33
Listing 19 RabbitMQ publishThread. ....	34
Listing 20 Starting ESP Service.....	35
Listing 21 EventBus class.....	35
Listing 22 Event publishing.....	35
Listing 23 Event subscription.....	36
Listing 24 Event handling. ....	36
Listing 25 File upload procedure. ....	36
Listing 26 disconnect() method call. ....	36
Listing 27 Deleting csv files. ....	37

**List of Tables**

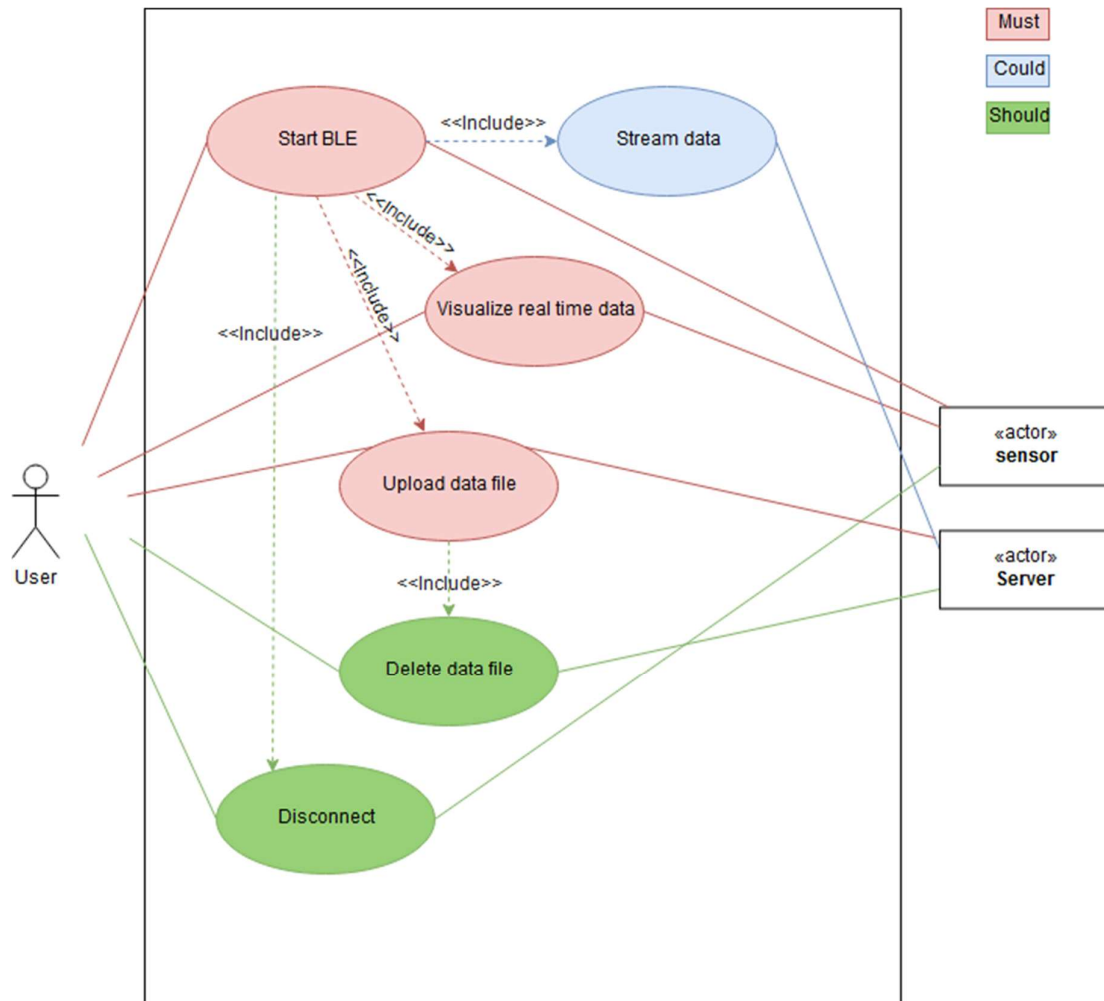
Table 1 Attribute (see [BLE]). ..... 15

## References

- [AAC] <https://developer.android.com/topic/libraries/architecture/>  
[Accessed: Feb 2, 2018].
- [ASSIST] <https://assist.ncsu.edu> [Accessed: Jan 31, 2018].
- [BLE] Bluetooth SIG, "Bluetooth Core Specification v 4.0", June 30, 2010.
- [BLE1] Bluetooth SIG, "Bluetooth Core Specification v 4.2", Dec 02, 2014.
- [BLE2] Texas Instruments, "Measuring Bluetooth Low Energy Power Consumption", Application Note AN092, April 2012, Dallas, Texas.
- [BLE4] Kinetis BLE Toolbox Mobile Application User's Guide, NXP Semiconductors, Document Number: KBLETMAUG, Rev. 0, 09/2016.
- [BLE5] Texas Instruments, "CC2540 and CC2541 Bluetooth low energy Software Developer's Reference Guide", September 2009, Dallas, Texas.
- [BLE6] <http://microchipdeveloper.com> [Accessed: May 01, 2018].
- [CISCO] <https://www.cisco.com/c/en/us/products/security/anyconnect-secure-mobility-client/index.html> [Accessed: March 01, 2018].
- [CYP] <http://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-psoc-4-pioneer-kit?source=search&keywords=CY8CKit-042>  
[Accessed: Nov 01, 2017].
- [DIA1] <https://www.dialog-semiconductor.com/products/connectivity/bluetooth-low-energy/smartbond-da14581> [Accessed: April 03, 2018].
- [DIAL] <https://support.dialog-semiconductor.com/connectivity/product/da14681>  
[Accessed: April 03, 2018].
- [ECDH] G. de Meulenaer, F. Gosset, F. X. Standaert and O. Pereira, "On the Energy Cost of Communication and Cryptography in Wireless Sensor Networks," *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Avignon, 2008, pp. 580-585.
- [EVENT] <http://greenrobot.org/eventbus/> [Accessed: Feb 15, 2018].
- [HET1] James Dieffenderfer et al., "Low-Power Wearable Systems for Continuous Monitoring of Environment and Health for Chronic Respiratory Disease", *IEEE Journal of biomedical and health informatics*, Vol. 20, No.5, September 2016.
- [HET10] A. Klinefelter et al., "21.3 A 6.45 $\mu$ W self-powered IoT SoC with integrated energy-harvesting power management and ULP asymmetric radios," *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, San Francisco, CA, 2015, pp. 1-3.
- [HET2] Veena Misra et al., "Flexible Technologies for Self-Powered Wearable Health and Environmental Sensing", *Proceedings of the IEEE*, Vol. 103, No.4, April 2015.
- [HET4] Abhishek Roy, Alicia Klinefelter, Farah B. Yahya, Xing Chen, Luisa Patricia Gonzalez-Guerrero, Christopher J. Lukas, Divya Akella Kamakshi, James Boley, Kyle Craig, Muhammad Faisal, Seunghyun Oh, Nathan E. Roberts, Yousef Shakhsheer, Aatmesh Shrivastava, Member, IEEE, Dilip P. Vasudevan, David D. Wentzloff, Member, IEEE, and Benton H. Calhoun, Senior Member, IEEE "A 6.45 Self-Powered SoC With Integrated Energy-Harvesting Power Management and

- ULP Asymmetric Radios for Portable Biomedical Systems", IEEE Transactions on Biomedical Circuits and Systems, Vol. 9, No. 6, December, 2015.
- [HET5] Farah Yahya, Christopher J. Lukas, Jacob Breiholz, Abhishek Roy, Harsh N. Patel, NingXi Liu, Xing Chen, Avish Kosari, Shuo Li, Divya Akella, Oluseyi Ayorinde, David Wentzloff, Benton H. Calhoun, "A Battery-less 507nW SoC with Integrated Platform Power Manager and SiP Interfaces", VLSI Circuits, 2017 Symposium of the IEEE, Kyoto, Japan, June 5-8, 2017.
- [HET7] A. Roy, P. J. Grossmann, S. A. Vitale and B. H. Calhoun, "A 1.3 $\mu$ W, 5pJ/cycle sub-threshold MSP430 processor in 90nm xLP FDSOI for energy-efficient IoT applications," 2016 17th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, 2016, pp. 158-162.
- [HET8] Benton H. Calhoun, Brian Otis, "Energy harvesting and control for sensor node", US9882428, Jan 30, 2018
- [HET9] Benton H. Calhoun, Yousef Shakhsheer, Yanqing Zhang, Alicia Klinefelter, David D. Wentzloff, Nathan E. Roberts, Seunghyun Oh, "Ultra low power sensing platform with multimodal radios", US20180097538A1, April 5, 2018.
- [JSCH] <http://www.jcraft.com/jsch/> [Accessed: March 05, 2018].
- [OMSP] OpenMSP430 Architecture [Online].  
Available: <http://opencores.org/project,openmsp430> [Accessed: March 16, 2018].
- [REN] <https://www.renesas.com/en-us/products/microcontrollers-microprocessors/rl78/rl78g1x/rl78g1d.html> [Accessed: Feb 08, 2018].
- [RMQ] <https://www.rabbitmq.com/api-guide.html>, [Accessed: April 28, 2018].
- [SCI] <https://www.scichart.com/>, [Accessed: Feb 08, 2018].
- [SDWM] Ian Sommerville: Software Engineering, Pearson, 10th Edition, April 2015, Boston, Massachusetts, S. 30.
- [SOC1] Performance Analysis of an Bluetooth Low Energy Sensor System E. Mackensen, M. Lai and T. M. Wendt, "Performance analysis of an Bluetooth Low Energy sensor system," 2012 IEEE 1st International Symposium on Wireless Systems (IDAACS-SWS), Offenburg, 2012, pp. 62-66.
- [SOC2] A. Roy et al., "A 6.45  $\mu$ W Self-Powered SoC With Integrated Energy-Harvesting Power Management and ULP Asymmetric Radios for Portable Biomedical Systems," in IEEE Transactions on Biomedical Circuits and Systems, vol. 9, no. 6, pp. 862-874, Dec. 2015.
- [TERM] <https://play.google.com/store/apps/details?id=com.server.auditor.ssh.client&hl=en> [Accessed: 11-April-2018].
- [TI1] Texas Instruments, "CC2540/41 System-on-Chip Solution for 2.4GHz Bluetooth® low energy Applications User's Guide", April 2009, Dallas, Texas.
- [UML] OMG Object Management Group: UML Specification Version 2.5, May 2015.

## Appendix A – Use Case Diagram



## Appendix B – Use Case Descriptions

Use Case Name	Start BLE
Brief Description	Establish a BLE connection
Pre-Condition	BT enabled, devices not connected
Post-Condition	Devices connected
Actors	<ul style="list-style-type: none"> <li>- User, primary, active, human</li> <li>- Sensor device, secondary, passive,</li> </ul>
Basic Flow	<ol style="list-style-type: none"> <li>1. The user presses the Start button.</li> <li>2. The system checks, if Bluetooth is enabled. If it is not enabled, the system prompts the user to turn on BT.</li> <li>3. The system checks, if access to device location is permitted. If it is not enabled, the system prompts the user to allow access.</li> <li>4. The system scans for a device holding the HET service. If it is found the system connects to the device.</li> <li>6. The system discovers services &amp; characteristics.</li> <li>7. The system enables the Kickstart button.</li> </ol>
Alternate Flow	<ol style="list-style-type: none"> <li>1. The system does not find the required device.</li> <li>2. The system scans until a BLE device offering the requested service becomes available.</li> </ol>
Trigger	The user presses the Start button.

<b>Use Case Name</b>	<b>Visualize real time data</b>
Brief Description	Real time chart rendering of sensor data.
Pre-Condition	Notifications are not enabled.
Post-Condition	Real time data visualization on Start Screen.
Actors	<ul style="list-style-type: none"> <li>- User, primary, active, human</li> <li>- Sensor device, secondary, passive,</li> </ul>
Basic Flow	<ol style="list-style-type: none"> <li>1. The user presses the Kickstart button.</li> <li>2. The system enables the setting of notifications.</li> <li>3. The system enables the notifications switches.</li> <li>4. The user puts the notification switches in the on position.</li> <li>6. The sensor device transmits data to the system.</li> <li>7. The system displays the real time charts.</li> </ol>
Alternate Flow	The user disconnects the device.
Trigger	The user presses the Kickstart button.



<b>Use Case Name</b>	<b>Upload data file</b>
Brief Description	Upload of csv file containing sensor data to the dedicated server.
Pre-Condition	Transmission of sensor data.
Post-Condition	Successful file upload to the dedicated server.
Actors	<ul style="list-style-type: none"> <li>- User, primary, active, human</li> <li>- Sensor device, secondary, passive,</li> </ul>
Basic Flow	<ul style="list-style-type: none"> <li>3. The user presses the Upload button on the Start screen.</li> <li>4. The system creates a csv file of the respective sensor data.</li> <li>5. The system opens the Upload Activity.</li> <li>6. The user presses the Upload button on the Upload screen.</li> <li>3. The system establishes a secure connection to the dedicated server.</li> <li>4. The system uploads the csv file to the dedicated server.</li> <li>6. The system displays toast messages indicating the progress of the file upload.</li> <li>7. The system returns to the Start screen.</li> </ul>
Alternate Flow	The user disconnects the device.
Trigger	The user presses the Upload button.

<b>Use Case Name</b>	<b>Stream data</b>
Brief Description	Streaming data received from the sensor device to the dedicated server.
Pre-Condition	<ul style="list-style-type: none"> <li>• Transmission of sensor data.</li> <li>• Enabling Port Forwarding to the dedicated server using the Termius app in the background.</li> </ul>
Post-Condition	Successful data streaming to the dedicated server.
Actors	<ul style="list-style-type: none"> <li>- User, primary, active, human</li> <li>- Sensor device, secondary, passive,</li> </ul>
Basic Flow	<ol style="list-style-type: none"> <li>1. The user opens the Termius app in the background to establish a secure connection to the forwarding port of the dedicated server.</li> <li>2. The system streams the data to the server.</li> </ol>
Alternate Flow	The user disconnects the device.
Trigger	New sensor values received from the sensor device.

## Appendix C – Class diagram







































MainActivity	
context	Context
TAG	String
start_button	Button
kick_button	Button
tess4_button	Button
tess3_button	Button
graphlcon_button	Button
char3_switch	Switch
char4_switch	Switch
testArray	ArrayList<String>
mArrayOz1	ArrayList<Integer>
mArrayOz2	ArrayList<Integer>
mArrayOz3	ArrayList<Integer>
mArrayOz4	ArrayList<Integer>
mArrayWav1	ArrayList<Integer>
mArrayWav2	ArrayList<Integer>
mArrayWav3	ArrayList<Integer>
mArrayAccX	ArrayList<Integer>
mArrayAccY	ArrayList<Integer>
mArrayAccZ	ArrayList<Integer>
mFileCounter	Integer
mFileCounterDrive	Integer
mFileName	String
mCsv	String
lineData	XyDataSeries<Integer, Integer>
lineData2	XyDataSeries<Integer, Integer>
lineData3	XyDataSeries<Integer, Integer>
lineData4	XyDataSeries<Integer, Integer>
lineDataOZ1	XyDataSeries<Integer, Integer>
lineDataOZ2	XyDataSeries<Integer, Integer>
lineDataOZ3	XyDataSeries<Integer, Integer>
lineDataOZ4	XyDataSeries<Integer, Integer>
surface	SciChartSurface
surface2	SciChartSurface
mConnectState	boolean
mServiceConnected	boolean
mBLEService	BLEService
REQUEST_ENABLE_BLE	int
PERMISSION_REQUEST_COARSE_LOCATION	int
char3NotifyState	boolean
char4NotifyState	boolean
sharedPreferences	SharedPreferences
sharedPreferencesDrive	SharedPreferences
mServiceConnection	ServiceConnection
mBleUpdateReceiver	BroadcastReceiver
mMessageReceiver	BroadcastReceiver
onCreate(Bundle)	void
onCreateOptionsMenu(Menu)	boolean
onOptionsItemSelected(Menuitem)	boolean
onStart()	void
onRequestPermissionsResult(int, String[], int[])	void
onResume()	void
onActivityResult(int, int, Intent)	void
onPause()	void
onStop()	void
onDestroy()	void
startBluetooth(View)	void
searchBluetooth(View)	void
readDesc(View)	void
kick(View)	void
connectBluetooth(View)	void
discoverServices(View)	void
disconnect(View)	void
disconnect()	void
onEvent(EventInteger)	void
onEvent(EventESPcon)	void
updateChart2WithThreeArrays(ArrayList<Integer>, ArrayList<Integer>, ArrayList<Integer>, XyDataSeries, XyDataSeries, XyDataSeries)	void
updateTopChartWithFourArrays(ArrayList<Integer>, ArrayList<Integer>, ArrayList<Integer>, ArrayList<Integer>, XyDataSeries, XyDataSeries, XyDataSeries, XyDataSeries)	void
writeStringAsFile4(String, String, Context)	String
deleteCsvFiles()	void
saveCurrentFileCounter()	void
loadLastFileCounter()	Integer
loadLastFileCounterDrive()	Integer
addValueTestArray(String)	String











































BLEservice	
TAG	String
mBluetoothManager	BluetoothManager
mBluetoothAdapter	BluetoothAdapter
mLEScanner	BluetoothLeScanner
mLeDevice	BluetoothDevice
mBluetoothGatt	BluetoothGatt
mChar1Characteristic	BluetoothGattCharacteristic
mChar3Characteristic	BluetoothGattCharacteristic
mChar4Characteristic	BluetoothGattCharacteristic
mCccd	BluetoothGattDescriptor
mCccd4	BluetoothGattDescriptor
CccdUUID	String
Char1UUID	String
CHAR3UUID	String
CHAR4UUID	String
mChar4ValueFull	String
mChar3ValueFull	String
mChar4ValueInt	Integer
mArrayChar4IntChart3psoc	ArrayList<Integer>
mArrayChar4StrChart3psoc	ArrayList<String>
mArrayChar4StrRaw	ArrayList<String>
mArrayChar3StrRaw	ArrayList<String>
mCallbackdesc	int
ACTION_BLESTARTED	String
ACTION_BLESCAN_CALLBACK	String
ACTION_CONNECTED	String
ACTION_DISCONNECTED	String
ACTION_SERVICES_DISCOVERED	String
ACTION_DATA_RECEIVED	String
ACTION_DATA_READ	String
ACTION_DESC_CHAR3_READ	String
ACTION_DESC_CHAR4_READ	String
ACTION_DESC_READ	String
MSG_SAY_HELLO	int
mMessenger	Messenger
mBinder	IBinder
mLeScanCallback	LeScanCallback
mScanCallback	ScanCallback
mGattCallback	BluetoothGattCallback





Chart2Activity		
f	mOnNavigationItemSelectedListener	OnNavigationItemSelectedListener
m	onCreate(Bundle)	void
m	onCreateOptionsMenu(Menu)	boolean
m	onOptionsItemSelected(MenuItem)	boolean

ChartHumidityFragment		
f	mArrayStrHumPSoC	ArrayList<String>
f	mArrayStrHumPSoCtrans	ArrayList<String>
f	mArrayHum1	ArrayList<Integer>
f	mArrayHum2	ArrayList<Integer>
f	chartLayoutHum	LinearLayout
f	chartSurfaceHum	SciChartSurface
m	onCreateView(LayoutInflater, ViewGroup, Bundle)	View
m	onStart()	void
m	extractSubToIntArray(ArrayList<String>, ArrayList<Integer>, Integer, Integer)	void
m	extractSubToIntArray(ArrayList<String>, ArrayList<Integer>, Integer)	void
m	addToArrayStr(String)	void
m	createChart2(ArrayList<Integer>)	void
m	createChart3(ArrayList<Integer>, ArrayList<Integer>)	void

ChartTempFragment		
f	mArrayStrTempPSoC	ArrayList<String>
f	mArrayStrTempPSoCtrans	ArrayList<String>
f	mArrayTemp1	ArrayList<Integer>
f	mArrayTemp2	ArrayList<Integer>
f	chartLayoutTemp	LinearLayout
f	chartSurfaceTemp	SciChartSurface
m	onCreateView(LayoutInflater, ViewGroup, Bundle)	View
m	onStart()	void
m	extractSubToIntArray(ArrayList<String>, ArrayList<Integer>, Integer, Integer)	void
m	addToArrayStr(String)	void
m	createChart3(ArrayList<Integer>, ArrayList<Integer>)	void

SSHActivity		
 	STARTED_ASYNC_MSG	int
 	SESSION_DISCONNECTED_MSG	int
 	SESSION_CONNECTED_MSG	int
 	CHANNEL_DISCONNECTED_MSG	int
 	CHANNEL_CONNECTED_MSG	int
 	SFTP_ERR	int
 	TRYING_MSG	int
 	mBackgroundHandler	Handler
 	uploadSSH_button	Button
 	mCSVca	String
 	mFilePath	String
 	mFileName	String
 	csvTxt	String
 	context	Context
 	onCreate(Bundle)	void
 	onStart()	void
 	onEvent(EventFilePath)	void
 	onEvent(EventFileName)	void
 	onEvent(EventString)	void

ESPService		
  CONNECTIONSTATUS		int
  PUBLISHED		int
  NOTPUBLISHED		int
  factory		ConnectionFactory
  queue		BlockingDeque<String>
  publishThread		Thread
  currentTime		long
  latencyCount		short
  dateFormat		DateFormat
  deviceId		String
  mStatus		Integer
  context		Context
  mBackgroundHandler		Handler
  onBind(Intent)		IBinder
  onStartCommand(Intent, int, int)		int
  onDestroy()		void
  addToQueue(String)		void
  publishToAMQP()		void
  close()		void
  sendMessage()		void
  getmStatus()		Integer

EventFileName		
  StringValue		String
  getMessage()		String

Appendix D – User Interface Mock ups

