8-2021

Development of an Encrypted Wireless System for Body Sensor Network Applications

Kendra Anderson
kander68@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Kendra Anderson entitled "Development of an Encrypted Wireless System for Body Sensor Network Applications." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Nicole McFarlane, Major Professor

We have read this thesis and recommend its acceptance:

Aly Fathy, Garrett Rose

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Development of an Encrypted Wireless System for Body Sensor Network Applications

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Kendra Anderson

August 2021

DEDICATION

To my family.

Acknowledgments

First, I would like to express my sincere gratitude to Dr. Nicole McFarlane for her guidance, support, and patience. Her deep knowledge and passion for the field has always encouraged and motivated me. Not only has she been my advisor during my master's program, but she has been an incredible mentor and role model.

I would like to thank my committee members: Dr. Aly Fathy and Dr. Garrett Rose for their insightful questions and comments of my thesis. I would also like to thank Dr. Fathy for his support throughout my academic career so far, and for providing equipment for measurements. I would like to thank Dr. Garrett Rose as well for his support with the probe station, and to Ryan Weiss, who helped me multiple times with the probe station.

I am deeply grateful for my family, who always offered their unwavering support and belief in me. I would like to especially thank my grandmother, Betty Taylor, my mother, Cindy Anderson, and my father, Wally Anderson, for their unconditional love and unending support.

I would like to thank my MLAB colleagues and all the people I have worked with during my time at UT: Bennett Waddell, Matthew Smalley, Ava Hedayatipour, Shaghayegh Aslanzadeh, Aminul Haghue, Zakaraya Hamdan, Kelli Determan, Farshid Tamjid, and other lab-mates in MK540 and MK538. I will always cherish my memories of long hours working on projects, eating lunches, hallways talks, and more. I would like to extend a special thank you to two of my co-authors, Farshid Tamjid and Ava Hedayatipour, for their valuable assistance in different projects.

Abstract

Wireless body area networks (WBAN), also called wireless body sensor networks (WBSN), consist of a collection of wireless sensor nodes used to monitor and assess various human physiological conditions, which can then be used by healthcare professionals to help them make important healthcare decisions. They can be used to prevent disease, help diagnosis a disease, or manage the symptoms of a disease. An extremely important aspect of WBAN is security to protect a patient's healthcare information, as a hacker could potentially cause fatal harm. Current security measures are implemented in software at the MAC layer and higher, not in the physical layer. Previous research demonstrated a chaotic encryption cipher to add a layer of security in the physical layer. This cipher exploits different properties of the Lorenz chaotic system to encrypt and decrypt digital data. Decryption involved synchronizing two chaotic signals to recover original data by sharing a state between the transmitter and receiver. In this thesis, we further develop the encryption system by implementing wireless capabilities. We use two approaches: the first by using commercially available wireless microcontrollers that communicate using Bluetooth Low Energy, and the second by the design and fabrication of a dual-band low noise amplifier (LNA) that can be used in a receiver for WBANs collecting data from implantable and on-the-body sensors. For the first approach, a custom Bluetooth Low Energy profile was created for streaming the analog encrypted signal, and signal processing was done at the receiver side. For the second approach, the LNA operates at the Medical Implant Communication System (MICS) band and the 915 MHz Industrial, Scientific, and Medical (ISM) band simultaneously through dual-band input and output matching networks.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Body Sensor Networks

Wireless sensor networks (WSN) have been a huge advancement in technology and have transformed the modern world. Wireless sensor networks consist of a collection of sensor nodes that are used to monitor and detect information, and then send that information to the outside world. Sensor nodes usually consist of a sensor, a microcontroller, a radio transceiver, a battery, and possibly external memory. They are used in various applications such as agriculture, traffic monitoring, fitness and wellness, military, medical, and social networking. Some specific application examples are vineyard monitoring, bridge monitoring, and animal monitoring [1]. Wireless body sensor network (BSN), also referred to as body area network (BAN), is a type of wireless sensor network focused on applications with the human body. Sensor nodes are placed around, on, or inside of the body to monitor physiological conditions. The collected data can be used by medical professionals to make important healthcare decisions. BSNs are typically used to track a patient over time in order to observe and detect any adverse event in real time, allowing real time decision making and intervention [2].

1.1.1 Communication Techniques

There are different protocols that a wireless sensor node can use to communication data. The most common are Bluetooth Low Energy (BLE), Zigbee, and the IEEE 802.15.6 standard. The IEEE 802.15.6 is the newest standard for body area networks and is recommended for BAN applications; however, the technology for this standard is premature and under-developed.

Bluetooth Low Energy (BLE)

The Bluetooth Low Energy protocol was introduced in version Bluetooth 4.0, and is branded as "Bluetooth Smart." The main difference from classic Bluetooth is that it consumes less power, making it suitable for cell phones, wearable hardware such as fitness trackers, etc. At it's core, Bluetooth consists of the host and the controller. The host defines the upper layers of the protocol, and the controller defines the lower layers of the protocol, such as the radio and link manager layers. BLE sends radio signals at 2.4 GHz and uses Gaussian frequency shift modulation. There are 40 channels between 2.402 and 2.480 GHz, each with 2 MHz bandwidth, that BLE can operate and send data on. To avoid interference with other protocols that use the 2.4 GHz band, such as Wifi and Zigbee, BLE uses a frequency hopping technique, called frequency hopping spread spectrum (FHSS), to hop between channels if there is any interference.

Zigbee

Zigbee is a communication protocol with low power, low data rate, and short range, and is specially built for sensor networks. It supports up to 65000 devices in its network. Zigbee networks consist of a coordinator, routers, and end devices. A coordinator is the root of the network, and routers are intermediate nodes between the coordinator and end devices. Zigbee network topolgies include star, mesh, tree, and cluster tree. The star topology contains no routers. Mesh and tree topologies have routers; however, mesh requires every node to be connected to every other node except end devices. Unlike mesh topology, the routers are not

interconnected in the tree topology. Clustered tree topology means routers are connected to other routers to extend the network range.

Zigbee can operate at 915 MHz or 2.4 GHz. At the 915 MHz band, it has 10 channels between 902-928 MHz spaced 2 MHz apart; at the 2.4 GHz band, it has 16 channels between 2.4-2.4835 GHz spaced 5 MHz apart. To avoid interference with other protocols, Zigbee uses direct sequence spread spectrum (DSSS), which spreads the signal over a wider bandwidth.

802.15.6 Standard

The first standard specifically for body area networks (BAN) is the IEEE 802.15.6 standard [3], which supports low power, short-range, and highly reliable communication. This standard defines multiple frequency bandwidths for the physical layer, which includes human body communication (HBC: 21 MHz), narrowband communication (NB: 402-405, 420-450, 863-870, 902-928, 950-958, 2360-400, and 2400-2483.5 MHz), and ultra-wideband communication (UWB: bandwidth - 499.2 MHz, 10 center frequencies ranging from 3.5 MHz to 10 MHz). This standard was published in 2012, and research relating to this standard are still underdeveloped.

1.2 Previous Work

Previously, a cipher was developed using discrete components that utilized time-scaling chaotic shift keying [4, 5]. This system is based on the Lorenz function, which is a mathematical model defined as,

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= (\beta - z)x - y \\ \dot{z} &= xy - \rho z \end{aligned} \tag{1.1}$$

where σ, β, and ρ are real positive values, and only certain relations result in a chaotic system. A plot of the system is shown in Fig. 1.1. The system can be bounded when meeting certain conditions, and a small change in initial conditions results in significantly different outcomes and trajectories. These properties can be exploited for an encryption

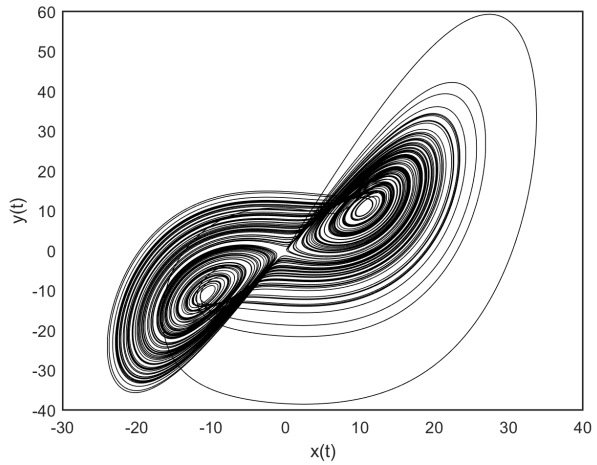


Figure 1.1: The Butterfly attractor from plotting x vs. y of the Lorenz function.

algorithm. Two chaotic systems can be synchronized and their trajectories matched, known as chaotic synchronization, which can be exploited for decryption algorithms. Chaotic shift keying (CSK) is the encryption/decryption algorithm that uses these two properties, and the system equations are,

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2 - x_1) \quad & \dot{z}_1 &= \sigma(z_2 - z_1) \\ \dot{x}_2 &= (\beta(m) - x_3)x_1 - x_2 \quad & \dot{z}_2 &= (\beta_0 - z_3)x_1 - z_2 \\ \dot{x}_3 &= x_1 x_2 - \rho x_3 \quad & \dot{z}_3 &= x_1 z_2 - \rho z_3 \end{aligned} \tag{1.2}$$

In these equations, the transmitter states are x_1, x_2, and x_3 and the receiver states are z_1, z_2, and z_3. β is the modulator. x_1 is the encrypted signal and the shared state with the receiver.

The CSK algorithm is vulnerable to the return map attack, where the local minimum and maximum with respect to time can be monitored to discover time-varying characteristics. To protect against this attack, a time scaling factor, λ(x,m), is added to create a time-scaling chaotic shift keying algorithm. The message m is a digital signal that can be 0 or 1. The system equations are,

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2 - x_1)\lambda(x, m) \\ \dot{z}_1 &= \sigma(z_2 - z_1)\lambda(z, 0) \\ \dot{x}_2 &= ((\beta(m) - x_3)x_1 - x_2)\lambda(x, m) \\ \dot{z}_2 &= ((\beta(m) - z_3)x_1 - z_2)\lambda(z, 0) \\ \dot{x}_3 &= (x_1 x_2 - \rho x_3)\lambda(x, m) \\ \dot{z}_3 &= (x_1 z_2 - \rho z_3)\lambda(z, 0) \end{aligned} \tag{1.3}$$

where,

$$\lambda(x, m) = \left\{ \begin{array}{ccc} \lambda_m & if & d_x = 0 \\ \lambda_{1-m} & if & d_x = 1 \end{array} \right\} \tag{1.4}$$

where $d(x)$ is the decision engine function, which uses a series of logic gates to perform a λ selection and is a function of the message signal, time, and the states of the system.

The full circuit is shown in Fig. 1.2. An algorithm is used to extract the original information signal, which uses periodic averaging, thresholding, and the shared state. More details regarding this system can be found in [4].

1.3 Organization

In Chapter 2, a literature review on wireless sensor nodes, chaotic ciphering, and multi-band low noise amplifiers. Chapter 3 details the system implementation of the wireless chaotic encryption cipher. Chapter 4 details the design, fabrication, and results of the dual-band LNA. Finally, Chapter 5 concludes the thesis and presents possible future work.

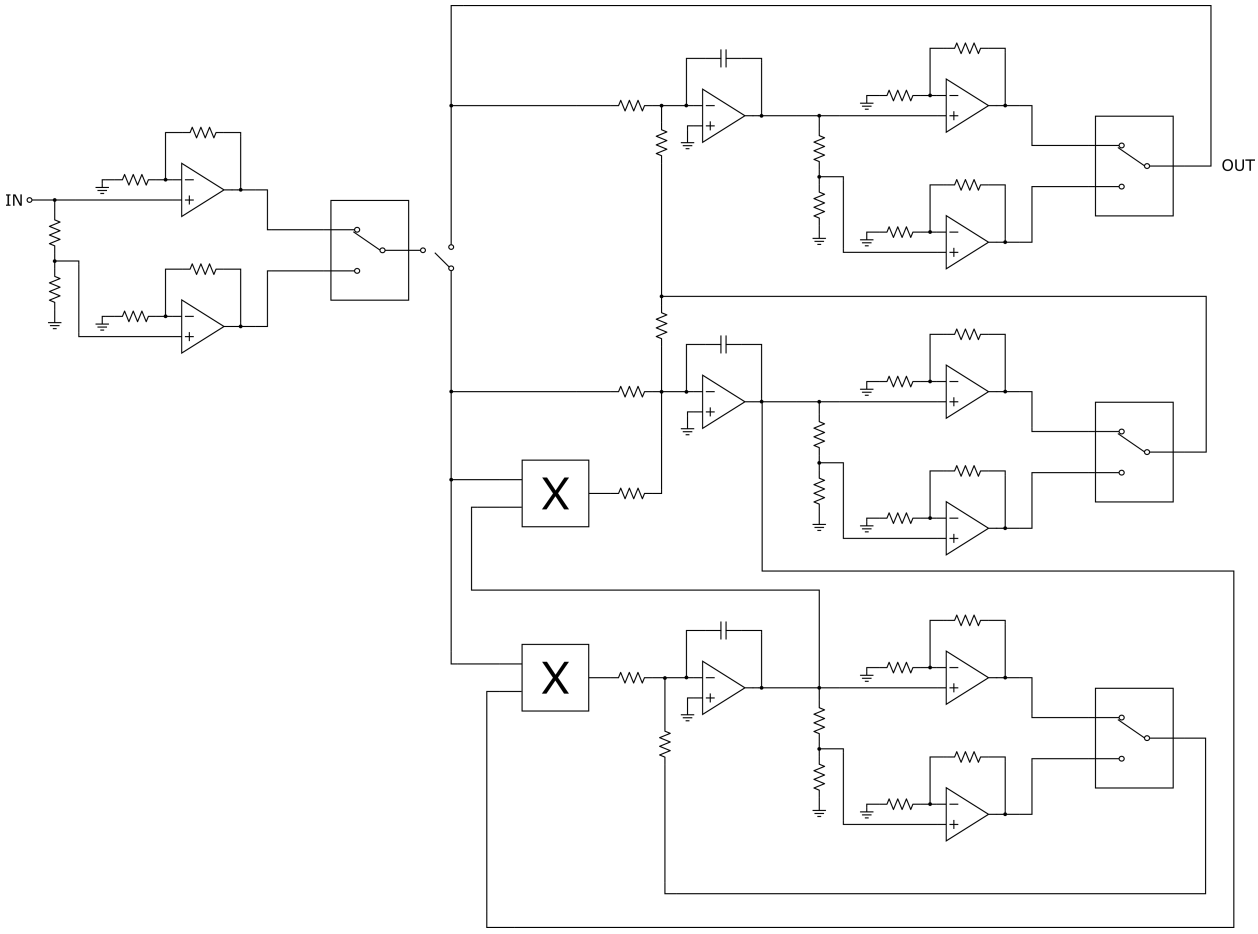


Figure 1.2: TS-CSK Circuit.

Chapter 2

Literature Review

2.1 Wireless Body Area Network

Even though wireless body area network technology is in a primitive stage, it is a hot topic in research and numerous papers are dedicated on either one aspect of a sensor node, such as the transceiver or sensor, or the entire functioning sensor node is presented to use in WBANs. Sensor nodes consist of the sensors, a micro-controller, a radio transceiver, and usually a battery.

A wearable sensor node was demonstrated in [6] that used electrocardiography (ECG) and photolenthysmography (PPG) sensor modules, an energy harvesting module, a low-power microcontroller, and a Bluetooth Low Energy module. They used a photovoltaic energy source to extend the battery life of the node. In [7], the presented sensor node contains an accelerometer, temperature sensor, and pulse sensor. The core microcontroller is an ATmega328P, and a Bluetooth Low Energy module is used to transmit data to a smartphone. The sensor node contains a flexible solar panel as a power source for energy harvesting, and stores energy is a supercapacitor to extend battery life. Experimental results confirm that the sensor node can operate autonomously for 24 hours, as long as it gets some sunlight everyday. A zigbee-based wireless sensor network is presented in [8], using sensor nodes that consist of a physiological parameter sensor (heart rate), a MSP430 microcontroller, and a Zigbee transceiver. The system demonstrates the potential of remote healthcare monitoring. A sensor node for glucose monitoring is presented in [9]. The node consists

of an optical glucose sensor, an energy harvesting unit, an energy storage element, and a wireless microcontroller that uses Bluetooth Low Energy.

These examples are just a few found in literature about wireless sensor nodes and/or wireless sensor networks. However, the only security measures that the systems take are within the communication protocols. Most encryption efforts outside of the communication protocol, if any, are done through software at the MAC layer or higher.

Combining chaos to wireless sensor networks, most work found in literature focuses on software chaotic encryption algorithms. In [10], a fast reaching finite time synchronization approach is verified in numerical simulation for chaotic systems, and its application to medical image encryption is explored. Secret keys are generated from synchronized chaotic systems, and an adaptive terminal sliding mode tracking approach is used to synchronize the chaos at the receiver and transmitter ends. A block encryption algorithm is presented in [11] that uses chaotic mapping along with other different types of mapping. Lower power consumption make it potentially suitable for wireless network applications.

2.2 Security

The two types of encryption are symmetric key encryption and asymmetric key encryption. Symmetric-key ciphers use the same private key for encrypting and decrypting data. Although it is a fast method of encryption, it depends on the sender exchanging the key with the receiver [12]. Asymmetric-key ciphers have one public key used for encryption and one private key used for decryption, where the private key is only known by the receiver [13]. One common way of hacking the cipher is through brute force method, where all of the possible combinations are tried until the right key is found to read the encrypted data. Other methods are through side-channel attacks, where the attacks exploit a system error in the cipher, or cryptanalysis, where a flaw in the encryption algorithm is exploited [12].

Cryptography is a fundamental part of all online communication and modern day computers. Various different algorithms exist that are pivotal to modern day technology and communication, utilizing asymmetric-key encryption. Once large quantum computers exist, many of these cryptosystems are expected to fail [13]. Classic computers relay on a binary

physical state of zero or one, called a bit; quantum computers relay on a particle's quantum state, known as a qubit. Qubits do not have a defined state, but rather a superposition of multiple states that simultaneously exist and are entangled together. Some companies have already made strives towards successful quantum computing. For example, IBM made a 5-qubit processor in 2016, and have continued upgrading the qubit count since then. Google, announced a 72-qubit processor [14]. Since quantum computing is on the horizon, security methods other than symmetric and asymmetric security are gaining importance. This is because quantum computing has the capacity to break cryptography keys using brute force method where it was not previously possible before using modern day computers. Chaotic ciphering is an alternative method of encryption that can implemented with low power electronics.

Communication systems that utilize chaotic encryption are scarce in literature. While efforts have been made to implement a chaotic communication system in digital and analog electronics, most implementations are software-based. Even rarer are the chaotic circuit implementations that have been constructed or fabricated. Because a chaotic system is very sensitive to initial conditions, most of the demonstrated circuits in literature had problems with accuracy due to mismatch and variance in component parameters.

In 1993, circuit implementations of the Lorenz chaotic system presented and demonstrated for the use of encryption for communication applications [15]. Two approaches were presented. First, the data was masked with a chaotic signal, and the receiver regenerated the mask to subtract it from the received signal and recover the data. Second, the coefficients of the chaotic system were modulated in the transmitter and the receiver detected the synchronization error. Since then, some circuit designs have been presented in FPGAs, using discrete components, and fabricated at the IC level.

One of the first experimentally verified chaotic encryption ICs is demonstrated in [16] where they implement a monolithic chaotic oscillator based on Chua's model for chaotic systems. The design incorporated a nonlinear resistor. A multiscroll chaotic oscillator is presented in [17] by using floating gate MOSFETs to implement the nonlinear function, and a later paper explores the limitations of this design [18]. Another experimentally verified cryptography IC is found in [19] that operates at the baseband level and uses a Lorenz based

chaotic system. The design implements differential circuits that implement mathematical functions to build the chaotic system. In [20], a chaotic oscillator is presented that uses a four-dimensional model of chaos from combining the Lorenz and Stenflo equations. The conditions for synchronization are defined, which can be exploited for decryption. In [21], a double-scroll chaotic system is implemented using OTAs. In [22], two designs of the Lu chaotic oscillator were presented.

One work in literature that incorporates chaotic-based encryption with bio-medical devices is presented in [23]. The work presents a symmetric encryption method that avoids exchanging the keys wirelessly. Instead, a chaotic system is used to generate pseudo-random keys from the preset initial conditions in the transmitter and receiver. Although this implementation uses chaotic systems, it is still considered a type of symmetric key ciphering that has a key length of 128 bits and is potentially breakable through quantum computing.

FPGA implementations of various chaotic-based systems have been presented in literature. An advantage of FPGAs is that there is no mismatch between the transmitter and receiver, which has been a problem in different implementation platforms due to the chaotic system's sensitivity to initial conditions. The only noise depends on noise sensibility [13]. For example, [24] presents an FPGA implementation of the Lorenz's chaotic generator, and [25] presents an FPGA implementation of Chua's chaotic system. In [26], a wireless transmission system is presented that uses chaotic encryption with an A5/1 algorithm on an FPGA platform. It uses a SIM300 module to realize wireless transmission. Even though this system is chaotic-based, the chaotic systems are not synchronized and the wireless module does not stream any data.

Implementations of chaotic-based systems can also be found using discrete components. In [27], a Lorenz-based discrete circuit is implemented using a master transmitter and slave receiver. In [28], a Lorenz design consists of analog multipliers, operational amplifiers, and passive elements as discrete components. It improves the Lorenz system by using an active control method so that the synchronization error system can be stabilized from the origin.

2.3 Multi-Band LNAs

Multiple strategies exist for designing an LNA that operates in two or more frequency bands. One strategy is to design an LNA for each desired frequency band and then put the LNAs in parallel with one another. While this offers the advantage of optimizing the performance of each LNA, it comes at the expense of increased power, area, and receiver complexity. Another strategy is to use wideband LNAs that cover a range of frequencies. Wideband LNAs are used for both multi-band and multi-mode receivers where the receivers can satisfy more than one standard protocol. Also, the ultra-wide band (UWB) frequency band is from 3.1 to 10.6 GHz, which support high data rate and bandwidth at low range, low energy levels [29]. For these reasons, many research efforts have been made to develop and improve wideband LNA design. Two of the biggest considerations when designing a wideband LNA are wideband input matching and linearity. Wideband input matching can be done with a common gate (CG) stage or a common source (CS) stage as the input stage. The common gate stage is relatively easier to to get wideband input matching; however, it has lower gain than its CS counterpart and degrades the noise figure (NF) of the whole circuit [30]. In a conventional CG LNA stage, the input matching network consists of an input inductor L_1 and the input transistor gate-source capacitance C_{gs1}, which create a resonance at,

$$\omega_0 = \frac{1}{2\pi\sqrt{C_{gs1}L_1}} \tag{2.1}$$

when neglecting load impedance. A low Q-factor results in wideband impedance matching [31], and the resistive 50 Ω matching is set by the transconductance of the input transistor,

$$g_m = \frac{1}{R_s} = \frac{1}{50} \tag{2.2}$$

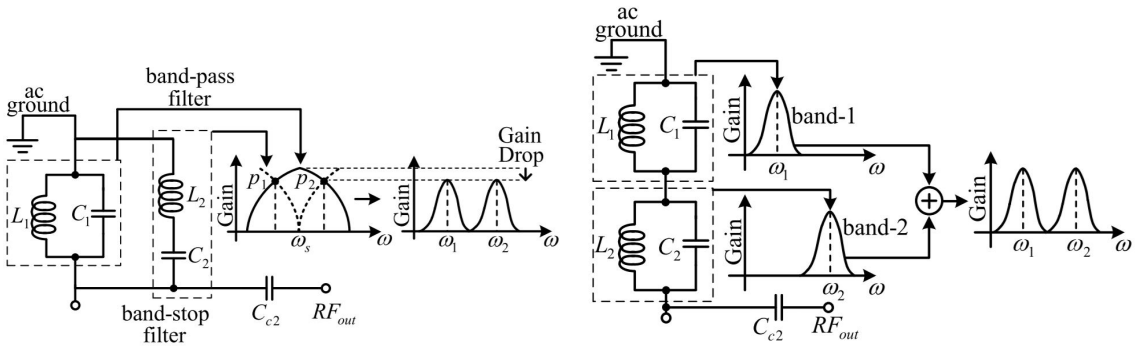
assuming channel length modulation is neglected [32]. A CS stage has higher gain and better NF than a CG stage, but it is harder to achieve wideband input matching. Typically, a CS stage is used with inductive degeneration or resistive feedback. The input impedance of a CS stage with inductive degeneration is [32],

$$Z_{in} = \frac{g_m L_1}{C_{gs1}} + L_1 s + \frac{1}{C_{gs1} s} \tag{2.3}$$

where g_m is the transconductance of the input transistor, L_1 is the inductive degeneration, and C_{gs1} is the gate-source capacitance of the input transistor. For input matching, the real term is set equal to 50 Ω and an input matching network is required to cancel the reactance of Eq. 2.3 at the frequency range of interest. A CS stage with resistive feedback is similar to the CG stage in that the resistive 50 Ω matching is set by Eq. [32]. However, the resistor degrades noise figure by contributing the input noise. Because wideband LNAs allow interference from unwanted covered frequeny ranges, linearity is especially important and can affect overall performance. One common way to improve linearity is by using differential circuits to cancel out the even order harmonics [32]. Another common method is the multiple gated transistors method (MGTR), which adds a transistor in parallel with the main transistor and biases it in subthreshold in order to cancel out the second order transconductance parameters of the two transistors and therefore improve linearity. Examples of this method are found in [33] and [34].

Although not as common in recent literature, another approach to designing a multi-band LNA is use of CMOS switches to change the operating frequency. The switches are turned on and off to add or remove different components in the matching networks, changing the resonance frequency and therefore the frequency band operation by doing so. The advantages of this approach are that they LNAs can have optimal performance at each desired frequency bands while rejecting unwanted frequencies. The disadvantages are that the LNA is limited to one frequency band at a time, and the CMOS switch adds parasitics to the circuit that affect circuit operation in both on and off states. When the switch is on, it acts like a resistor whose value is dependent on the transistor width. Too high of a resistance may result in the matching network not being affected by the added components, and too low of a resistance may result in a lowered Q factor of the matching network [32]. When the switch is off, it acts like a capacitor, which will shift the resonance frequency. A few examples of this approach are found in [35], [36], and [37].

The use of more complicated matching networks can be used to create multiple resonance frequencies of the desired bands. This approach has the advantage of operating at the desired bands simultaneously and rejected unwanted frequencies at the expense of more complicated matching networks. This approach is more commonly used for dual band operation. For output matching, the two most common matching networks are a parallel combination of bandpass and bandstop filters and two bandpass filters connected in series. In the former, the bandpass filter allows a large frequency range to pass, and the bandstop filter cuts that range in the middle to create break the passband into two, as shown in Fig. 2.1a. In the latter, two separate bandpass filters create the two resonances at the desired frequency bands, as shown in Fig. 2.1b. Input matching networks can range from any combination of filters or matching network configurations such as the L-shape, T-shape, π-shape matching networks.



(a) OMN Approach 1 **(b)** OMN Approach 2

Figure 2.1: Output Matching Network (OMN) Approaches [38].

Chapter 3

Wireless Module

3.1 Background

This background is summarized from the TI BLE-5 User's Guide [39].

3.1.1 OSI Model

The OSI model stands for Open System Interconnection Model, and it represents a standard reference model for communication protocols. It illustrates how two devices communicate with one another with 7 different layers, as shown in Fig. 3.1. The top layer is the application layer, which contains different network applications that an end-user can use to produce data. The data is sent to the presentation layer, where it is converted to machine language, called translation. The bits that comprise the data get reduced, which is called compression. Then, the data gets encrypted and sent to the session layer. The session layer has multiple roles. This includes initializing, managing, and ending connections. It checks to see if the data is synchronized and re-synchronizes if needed. The data continues to the transport layer, which is responsible for segmentation/reassembly, flow control, and error control. Segmentation takes the data from the session layer and divides it into smaller units, called segments, to send to the network layer on the sender side; reassembly reassembles the segments from the network layer to be able to send to the session layer on the receiver side. The Source and Destination port numbers are added to the header. Flow control determines the amount of

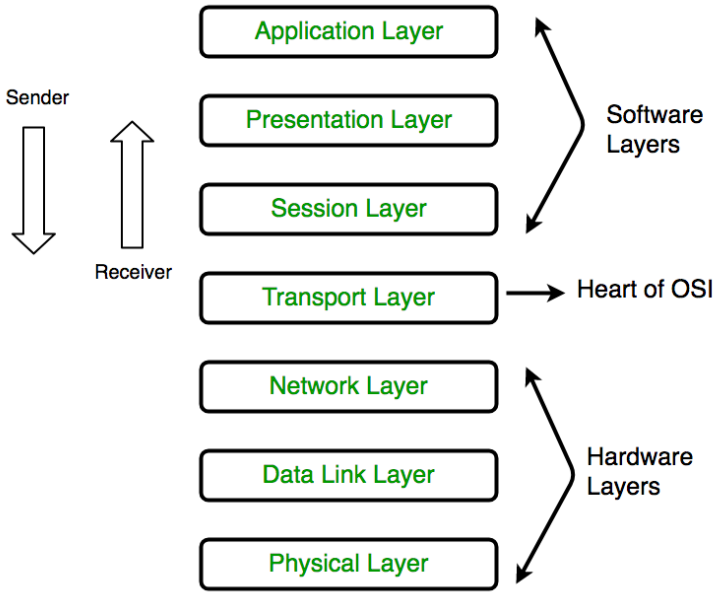


Figure 3.1: OSI Model. [40]

data being transmitted or received. Error control checks data for corruption and re-transmits data if there is an error.

The network layer is responsible for communication between networks and between hosts. It determines the best path for data to take, which is known as routing. It is also responsible for logical addressing, in which the sender and receiver's IP addresses are added to the header of the data unit. Data units in the network layer are called packets. To ensure the data is transferred correctly over the physical layer, the packets get sent to the data link layer. Through framing, the data link layer allows upper levels to access media and provides a way to transmit meaningful bits to the receiver. Media access control (MAC) addresses are added to the header of the data units, now called frames, through physical addressing. The data link layer re-transmits any damaged or lost frames and maintains a constant data rate from both the sender and receiver. When multiple devices are using the same communication channel, the data link layer will determine which device has control over the channel and for how long. Lastly, the physical layer converts the information bits into physical signals that can be transmitted over local media. The physical layer provides a clock for bit synchronization, defines the transmission rate, defines the device topology for a given network, and defines the direction of data flow. Data starts at the application layer of the sender side, flows down the layers, gets transmitted to the receiver side, and data flows back up from the physical layer to the application layer [40].

3.1.2 Bluetooth 5 Stack

Similarly following the OSI model, an overview of the BLE-5 stack can be seen in Fig. 3.2. Profiles and applications sit on top of the Generic Access Profile (GAP) and Generic Attribute Profile (GATT) layers. The Bluetooth stack consists of the host (software layers) and the controller (hardware layers). The controller, host, and applications can all be implemented in a single device, or the applications can reside on an external application processor (AP) such as a smart phone or laptop. The GAP layer controls the connection functionality of the device and interfaces with the application. The GATT layer provides a framework for using the Attribute Protocol (ATT). Attributes are the smallest unit of data that is communicated between Bluetooth devices. The Security Manager (SM)

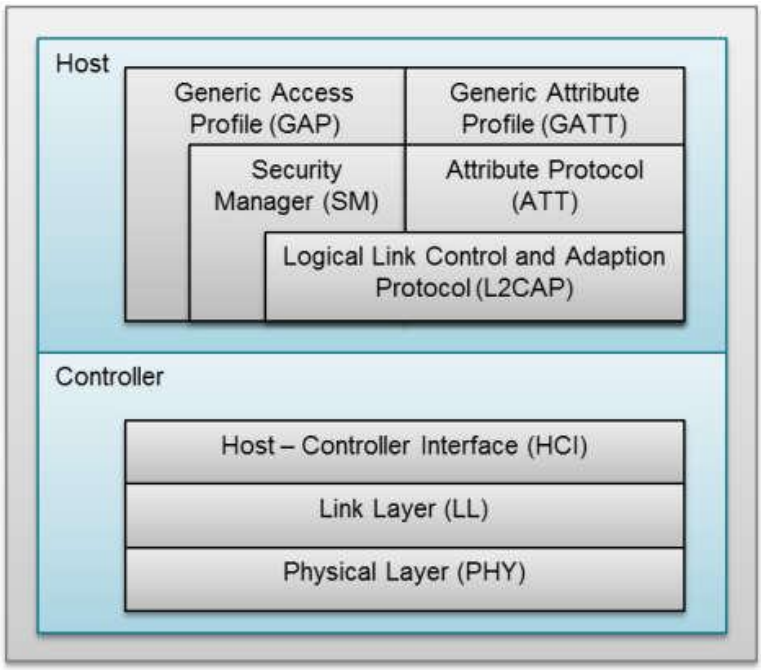


Figure 3.2: BLE 5 Stack. [39]

provides methods to securely exchange data. Logical Link Control and Adaption (L2CAP) provides data encapsulation services for data units traveling through the layers, and the Host-Controller Interface (HCI) is the interface between the L2CAP and Link Layer (LL).

Generic Access Profile (GAP)

The GAP layer is responsible for procedures for device connection (establishing, maintaining, and terminating a link) and device configuration. There are five RF states that a device can be in: standby, advertiser, slave, scanner, initiator, and/or master (Fig. 3.3). The standby state is an idle, unconnected device. When a device wants to connect to another device, it sends out an advertisement on one of the designated Bluetooth channels to say that it is a connectable device (advertiser). The other device wanting to connect looks for advertisements by scan requests (scanner). Device discovery occurs when the advertiser sends a scan response back to the scanner. Then, the scanner becomes an initiator and a request to initiate a link with the advertising device is sent. After they are connected, the scanner and advertiser become the master and slave, respectively. GAP roles include broadcaster, observer, peripheral, and central. The broadcaster and peripheral roles are advertisers, while the observer and the central are the scanners. The broadcaster and observer are not connectable, while the peripheral and central can connect to each other. Devices can operate in one or more roles and utilize one or more states.

A connection event occurs two devices send and receive data to each other. The time between connection events is the connection interval. The slave device can deny a particular number of connection events. This is known as called slave latency. The maximum time span between two successful connection events is known as the supervision time-out. The connection terminates after this time-out.

Generic Attribute Profile (GATT) and Attribute Protocol (ATT)

The GATT layer is an abstraction of the ATT layer and is used to transfer data between connected devices. An attribute is the smallest data unit. Characteristics are made up of attributes and data is communicated in the form of characteristics. Every attribute has a handle, a type or Universal Unique Identifier (UUID), and permissions. A handle is the index

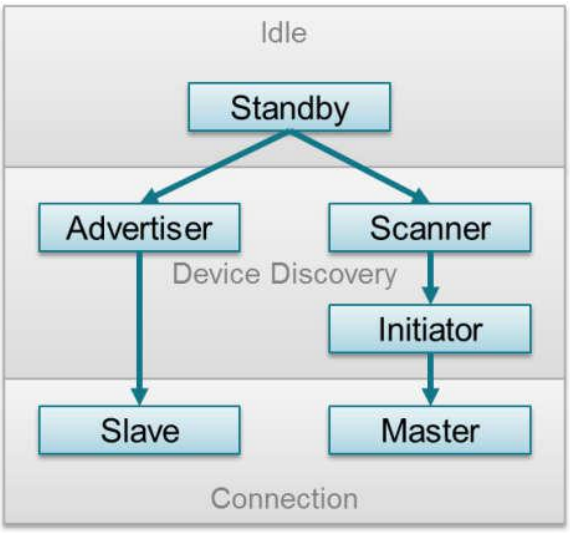


Figure 3.3: GAP State Diagram. [39]

of the attribute in the table, a type indicates how to interpret the data, and the permissions enforce how an attribute can be accessed from another device. Every characteristic contains the following attributes [39]:

- Characteristic Declaration: stores the properties, location, and type of characteristic value
- Characteristic Value Declaration: stores the data value
- Client Characteristic Configuration: permits the GATT server to write a 0, 1, or 2 within the characteristic to allow for no updates, updates (notifications), or updates with acknowledgements (indications), respectively.

There are two devices: GATT server and GATT client. The server device contains the database of attributes grouped by characteristics. The GATT client communicates (reads and writes data) with the GATT server.

In a connection between two devices, a GATT server is the device that contains the characteristic database and the GATT client is the device that is reading or writing data from/to the GATT server. A GATT server can independently define permissions for each characteristic. Two permission techniques are authentication and authorization. In authentication, characteristics cannot be read or written to until the GATT client has undergone a pairing method that is performed within the BLE stack; in authorization, the stack forwards any requests on the characteristics to the application layer, where the requirements for authorization are defined. A group of characteristics is called a service, and a group of services is called a profile, as shown in Fig. 3.4. Many profiles implement one service, so the terms profile and service can be used interchangeably in that case.

GAP Bond Manager

The GAP Bond Manager performs pairing and bonding security processes associated with the Security Manager (SM) protocol from the application. The module is configurable and is responsible for the pairing process (where keys are exchanged), encrypting the link, storing keys in the secure flash (SNV), and reconnecting if needed. The pairing methods include:

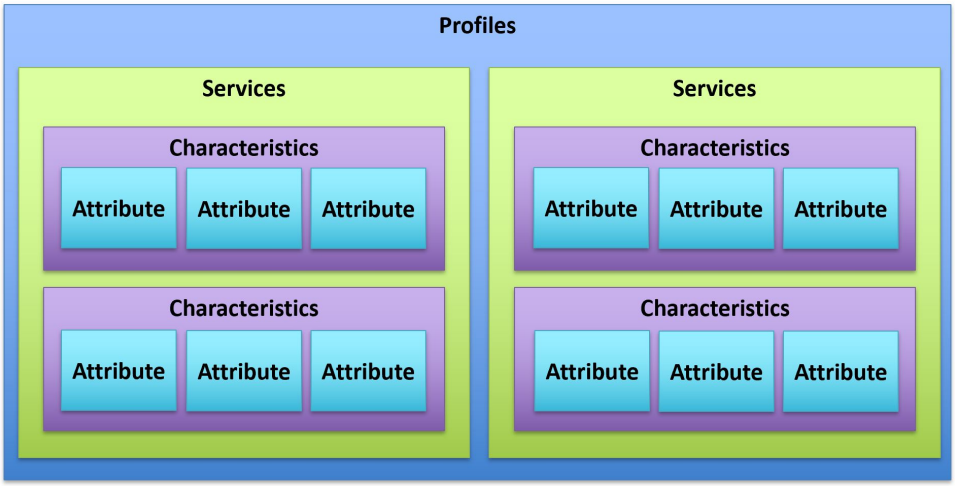


Figure 3.4: Data units in the BLE-5 Stack.

- Just Works: it just pairs
- Passkey Entry: an authenticated pairing method where one device displays a passcode and the other inputs it
- Numeric Comparison: an authenticated pairing method where both devices show a 6-digit code and indicate if codes match
- Out of Band: devices send authentication information over an out of band channel

Logical Link Control and Adaption Protocol (L2CAP)

The Logical Link Control and Adaption Protocol (L2CAP) is responsible for transferring data between the upper layers of the host and the link layer. It performs multiplexing, segmentation, and reconstruction of the communicated information. The L2CAP channel is the logical link between the protocol endpoints of the peer devices. A service data unit (SDU) is a packet of data tha contains the raw data from the application with no headers, while a protocol data unit (PDU) is the same packet of data but with L2CAP headers. Fragmentation is the process of breaking down PDUs into smaller units, and recombination is the process of reassembling the smaller units into complete PDUs. Similarly, segmentation is the process of breaking a single SDU up into smaller segments, and reassembly is the process of combining the smaller segments together into a complete SDU. These processes that the L2CAP is responsible for is demonstrated in Fig. 3.5.

Host Controller Interface (HCI)

The host controller interface (HCI) is responsible for transferring data and commands between the host and the controller elements of Bluetooth. This layer can use transport protocols such as SPI or UART, or they can use function calls and code all within one microcontroller (MCU). Executing this layer through transport protocols allows the ability for the application to run on an external MCU to interface with the Bluetooth stack.

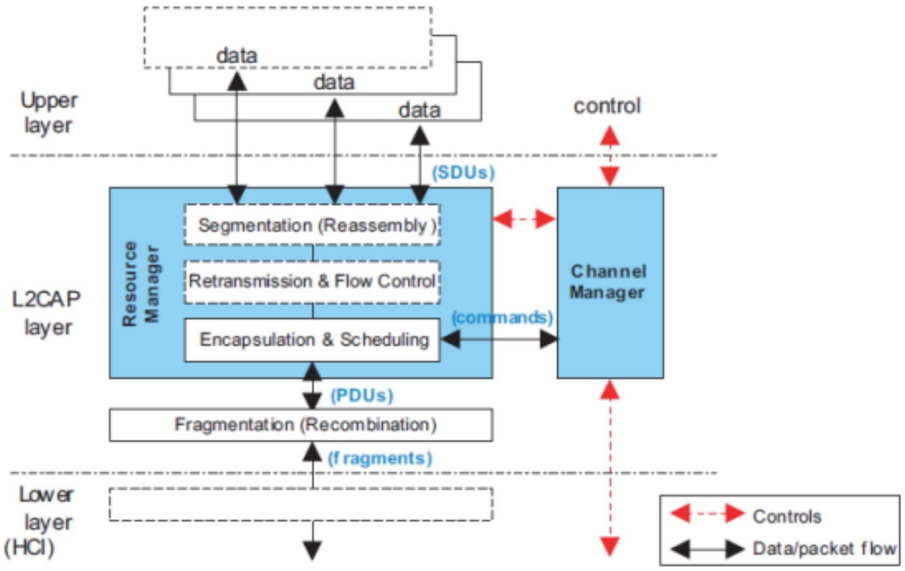


Figure 3.5: L2CAP Data Flow. [39]

Link Layer (LL) and Physical Layer (PHY)

The link layer (LL) and physical layer (PHY) have the same responsibilities as described in the standard, general OSI model. In the Bluetooth stack, the link layer controls which of the five RF states the device is in: standby, advertising, scanning, initiating, or connected. When in the connected state, a device can either be a central or peripheral device. The link layer is also responsible for scheduling, which physical channel to be on, and the length of the data packets. The Bluetooth stack supports three different PHYs: LE 1M, 2M, and coded PHY. LE 1M transfers data at a symbol rate and data rate of 1 Mbps, LE 2M at 2 Mbps. One symbol is equal to one bit. Using the same transmit power, the difference between 1M and 2M is the modulation type. In LE Coded PHY, each bit is represented by either two or eight symbols (S2 or S8, respectively). This allows the signal range to increase, but data throughput decreases, at a data rate of 500 kbps and 125 kbps for S2 and S8, respectively. Table 3.1 summarizes the differences in the PHYs.

3.2 System Implementation

Portions of this section have been submitted as *"A Wireless Time-Scaling Chaotic Shift Keying Encryption System For Biosensing Systems,"* to the IEEE Engineering in Medicine and Biology Conference (EMBC), 2021.

3.2.1 Hardware

Transceiver Boards

The LAUNCHXL-CC26x2R1 evaluation board was chosen to develop the software running on the CC2652R wireless microcontroller (MCU). This particular wireless microcontroller was chosen because it supports Bluetooth 5.1 Low Energy and contains the same microcontroller model that our lab has a tapeout for, the Arm Cortex-M0. In fact, the CC2642R has two microcontrollers: the ARM Cortex-M4F that is used as the core MCU for the BLE stack and the Arm Cortex-M0 that is used to interface with sensor data, called the sensor controller. This allows the main MCU to consume less power by offloading some of the signal processing.

Table 3.1: A comparison of the different PHY options in BLE-5. [39]

Parameter	LE 1M	LE 2M	LE Coded S=2	LE Coded S=8
Symbol Rate	1Msps	2Msps	1Msps	1Msps
Data Rate	1Mbps	2Mbps	500kbps	125kbps
Error Correction	None	None	FEC	FEC
Range Multiplier	1	~0.8	~2	~4

The MCU has 352 kB of programmable flash, 256 kB of Read Only Memory (ROM), 8 kB of cache SRAM, and 80 kB of ultra-low leakage SRAM. It offers many peripherals such as GPIO pins, general-purpose timers, two UART, two SSI, I2C, and a real-time clock (RTC). It contains eight channels for a 12-bit ADC with a sampling rate of up to 200 kSamples/sec.

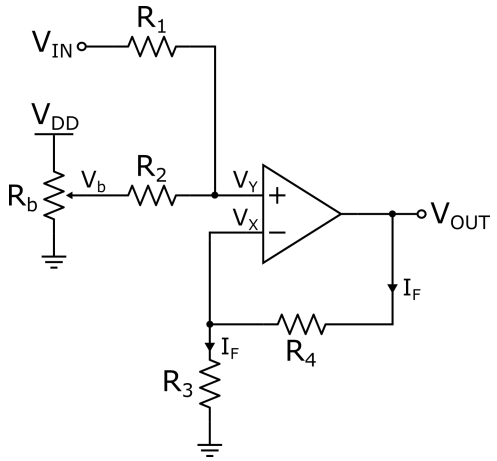
DAC

A DAC is needed to convert the signal back to an analog signal on the receiver's side in order to run it back through the cipher to recover the original signal information. The BOOST-DAC7551Q1 was chosen because it is a 12 bit, like the ADC on the transmitter side, and is compatible with the BoosterPack layout that the transceiver evaluation boards use. The DAC is compatible with SPI to communicate with the transceiver boards. To input data to the DAC, a 16-bit word is loaded into the input shift register, under the control of a clock signal SCLK. In normal mode, the first two bits are don't care bits, the next two bits should be low, and the rest of the bits are the data for the DAC with the most significant bit (MSB) first. Data is loaded when the $\overline{\text{SYNC}}$ signal is low, and when the $\overline{\text{SYNC}}$ signal is brought high again, the last 16 bits of data stored in the register and latched into the DAC register and updates the DAC. The clock signal operates in Mode 1, meaning the clock polarity is non-inverted (idle=0, active=1), and data is sampled at he falling edge of the clock.

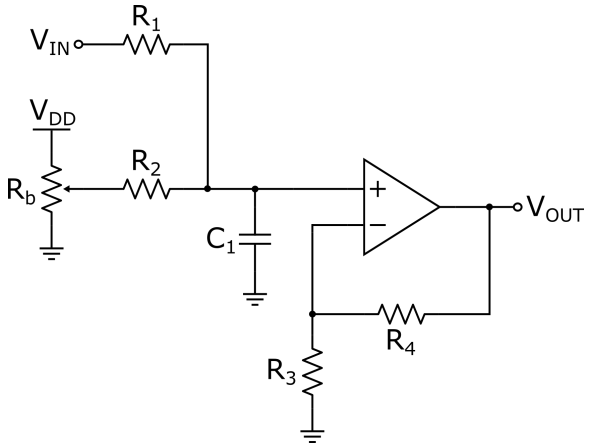
Signal Processing Circuitry

A non-inverting summing amplifier was added in between the cipher output and the ADC input on the transmitter side for two purposes: to provide a buffer before the ADC input with low output resistance and to add a DC offset to the encrypted signal. The DC offset moved the signal into the voltage range of required by the ADC of all positive values and in between the range of 0 to 4.2 V. The schematic of the operational amplifier can be seen in Fig. 3.6, and the derivation of the gain is as follows,

$$\begin{aligned} V_{out} &= R_4 I_F + R_3 I_F \\ I_F &= \tfrac{V_X}{R_3} \\ \tfrac{V_{out}}{V_X} &= \tfrac{R_3+R_4}{R_3} \end{aligned} \tag{3.1}$$



(a) TX Buffer



(b) RX Buffer

Figure 3.6: The operational amplifier circuits.

In the summing part of the amplifier, KCL at the node V_Y and assuming R_1 equals R_2 yields,

$$\begin{aligned} \tfrac{V_{IN}-V_Y}{R_1} + \tfrac{V_b-V_Y}{R_2} &= 0 \\ R_1 = R_2 &= R \\ \tfrac{V_{IN}}{R} + \tfrac{V_b}{R} &= \tfrac{2V_Y}{R} \\ V_Y &= \tfrac{1}{2}(V_{IN} + V_b) \end{aligned} \tag{3.2}$$

In an ideal operational amplifier, the voltages at both inputs are equal to each other ($V_X = V_Y$). Assuming the op amp is close to ideal, the full equation after combining Eq. 3.1 and Eq. 3.2 becomes,

$$V_{OUT} = \left(1 + \frac{R_4}{R3}\right)\left(\frac{V_{IN} + V_b}{2}\right) \tag{3.3}$$

On the receiver side, another non-inverting summing amplifier was added after the DAC to reverse the DC offset added on the transmitter side. An RC low pass filter was added right before the amplifier to remove the high frequency components added from the DAC, as shown in Fig. 3.6b. It uses the resistor from the summing amplifier and adds a shunt capacitor after the node. The gain of the RC filter can be found from using voltage division:

$$\frac{|V_Y|}{|V_b|} = \frac{|\frac{1}{j\omega C_1}|}{|R_2 + \frac{1}{j\omega C_1}|} \tag{3.4}$$

The cutoff frequency is,

$$f_c = \frac{1}{2\pi R_1 C_1} \tag{3.5}$$

Lastly, a difference operational amplifier was added added the receiver buffer to subtract the two signals needed for decryption, shown in Fig. 3.7. The gain is calculated by first doing KCL at nodes V_X and V_Y and voltage division at V_Y,

$$\begin{aligned} \tfrac{V_{IN1}-V_X}{R_1} &= \tfrac{V_X-V_{OUT}}{R_2} \\ \tfrac{V_{IN2}-V_Y}{R_3} &= \tfrac{V_Y}{R_4} \\ V_Y &= V_{IN2}\tfrac{R_4}{R_3+R_4} \end{aligned} \tag{3.6}$$

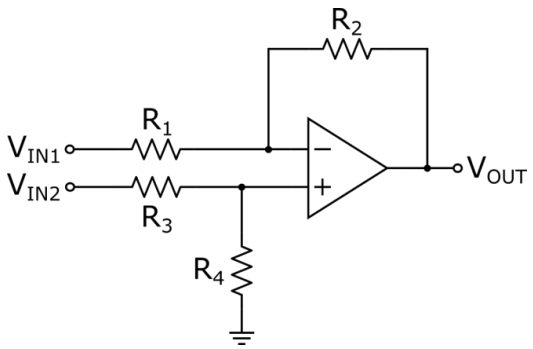


Figure 3.7: Signal extraction circuit.

then using superposition,

$$\begin{aligned} V_{IN2} = 0 : V_{OUT,A} &= -V_{IN1}\frac{R_2}{R_1} \\ V_{IN1} = 0 : V_{OUT,B} &= V_{IN2}\frac{R_4}{R_3+R_4}\left(\frac{R_1+R_2}{R_1}\right) \\ V_{OUT} &= V_{OUT,A} + V_{OUT,B} \end{aligned} \tag{3.7}$$

If $R_1 = R_3$ and $R_2 = R_4$, then the final gain equation can be simplified to,

$$V_{OUT} = \frac{R_2}{R_1}(V_{IN2} - V_{IN1}) \tag{3.8}$$

For the interface circuits, the LM741 operational amplifiers were used with 10 kΩ resistors for all resistors. The capacitor in the low-pass filter is a 0.47 μF electrolytic capacitor. Bypass capacitors were used as well.

3.2.2 Software

BLE Profile

A custom Bluetooth Low Energy profile was created that contains one custom service called CipherService and two characteristics called CipherValue and StreamEN, as shown in Fig. 3.8. The service has a UUID equal to 0xBA55. The CipherValue characteristic has a UUID of 0x2BAD and has read and notify properties. Its attributes include a characteristic declaration, a characteristic value declaration, and a client characteristic configuration, where the client characteristic configuration's value allows for notifications. When notifications are turned on, an alert will be sent to the receiver that the value has been changed, and no response from the receiver is required. The CipherValue characteristic is used to hold and update the value of the encrypted signal. The StreamEN characteristic has a UUID of 0x2BE, and its attributes include a characteristic declaration and a characteristic value declaration. The characteristic is readable and writable. It is initially set to zero. When the value "01" is written to it, it turns on the ADC through a callback function to record and stream data. To turn off stream, write another value besides "01".

BLE Custom Profile: CipherService	
GATT Primary Service Declaration	
GATT Characteristic Declaration	
CipherValue:	Contains 20-byte array of sampled data from ADC buffer
Client Characteristic Configuration:	Read and notify properties
GATT Characteristic Declaration	
StreamEN:	Write “01” to turn on streaming and “00” to turn off

Figure 3.8: The custom BLE service.

3.3 Results

3.3.1 System Overview

Fig. 3.9 shows a block diagram of the entire cipher system, and Fig. 3.10 shows the physical setup of the system. It starts off with a low frequency, digital signal that would be produced by a sensor. More specifically, sensors that could be used with this system are those with a quasi-digital output, in which information is encoded in its frequency. The encryption module masks the digital data using time-scaling chaotic shift keying and makes it look like a noisy, random signal. This analog signal is the input to the transmitter board, which continuously samples it with a built-in 12-bit ADC buffer and stores it in a 16-bit unsigned integer array storing 10 samples when the StreamEN characteristic is enabled. Once the array is full of samples, the array is converted from type 16-bit unsigned integer to type 8-bit unsigned integer storing 20 samples, since the BLE stack transfers data in bytes. Once converted, a notification is sent to the receiver that the value has changed and the transmitter sends the receiver the updated data. The maximum data length that BLE can send in one packet for notifications is 20 bytes.

On the receiver side, the Bluetooth data packet is collected and processed. After processing the notification signal and extracting the sent data, called the payload, the receiver converts the type 8-bit array back to type 16-bit to be able to communicate the value to the DAC through SPI and convert it back to an analog signal. The encrypted analog signal goes through the cipher again to get another chaotic signal for decryption, and the two signals are subtracted in circuitry to recover the original signal information.

Bluetooth uses frequency-hopping spread spectrum to avoid interference with other signals in the same frequency band, meaning that two devices communicate data with each other on specific channels at specific times, and continuously hop between channels to transfer data. This meeting is known as a connection event, and the amount of time between two connection events is called the connection interval. In Bluetooth Low Energy, the connection interval is between 7.5 ms to 4 s. In order to stream the data continuously and at the correct rate, the minimum connection interval is used, at 7.5 ms.

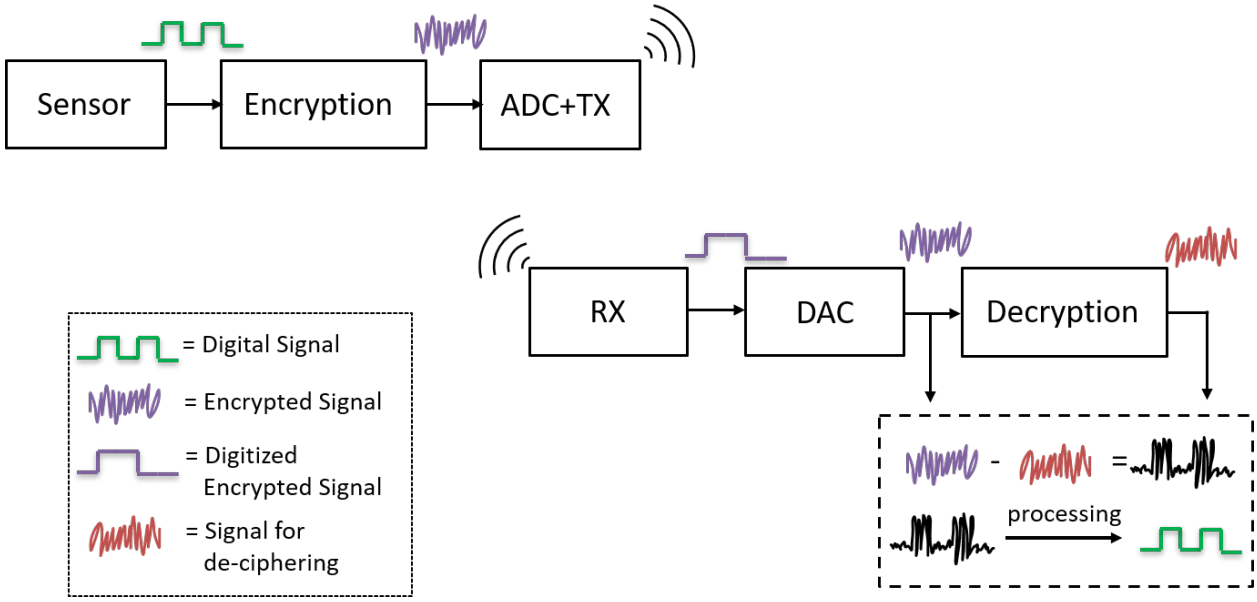


Figure 3.9: Block diagram of the whole system.

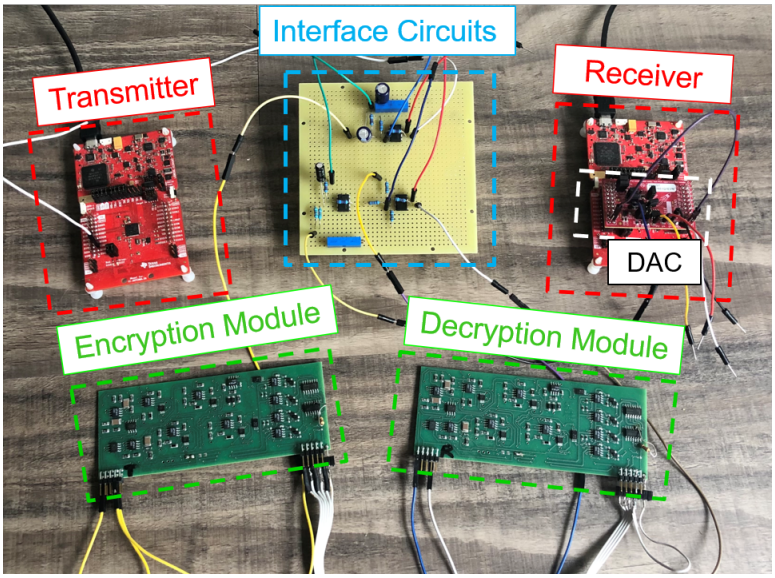


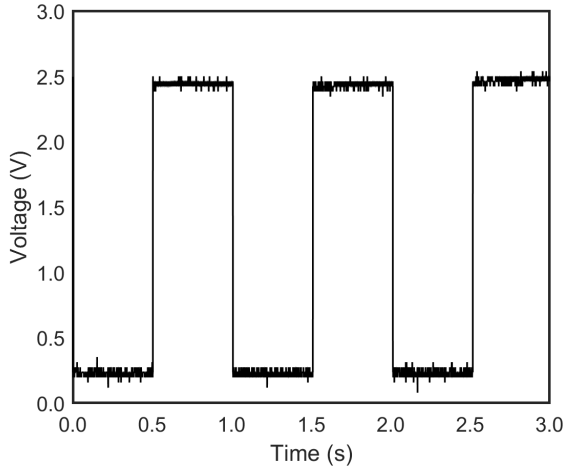
Figure 3.10: Set-up of the system.

3.3.2 Performance

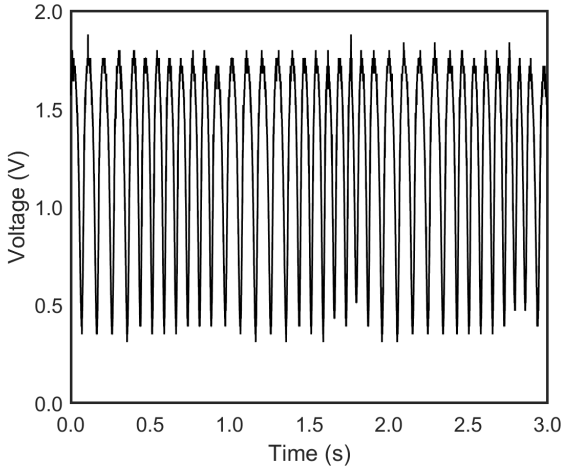
Fig. 3.11 show the final results of the system. Fig. 3.11a shows the low-frequency, digital input signal that replicates what a biological sensor would produce. For this test, a 2.5 V, 1 Hz pulse is produced. Fig. 3.11b and fig. 3.11c show the encrypted signal at the transmitter and receiver, respectively. The logical highs and lows are indistinguishable. The encrypted signal at the receiver side shows the effect of sampling at 133 Hz on the signal. Fig. 3.11d shows the recovered signal after subtracting the two chaotic signals for decryption, and the dashed line is the fully recovered, original signal after running it through a thresholding algorithm detailed in [4]. The parameters were the following: a weight of 0.4 and a cut of 0.15. It should be noted that the algorithm requires a previous knowledge of the message's frequency.

The continuous time fourier transform (CTFT) was taken for the input signal and encrypted signals. The frequency domain of the encrypted signal is shown in Fig. 3.12, and the frequency domain of the input signal is shown in Fig. 3.13. The CTFT of Fig. 3.11b is shown in Fig. 3.12a, and a zoomed-in version of the same graph is shown in Fig. 3.12b. This signal is before any sampling from the ADC, and represents the original encrypted signal information. Fig. 3.12c shows the encrypted signal at the receiver side, after it has been sampled by the ADC on the transmitter side and after it goes back to an analog signal by the DAC. The sampling process adds some higher frequency components to the signal; however, the original frequency information is still intact. The low-pass filter is added to remove those high-frequency components, which is shown in Fig. 3.12d. The frequency information is not lost from sampling. The slight drop in gain from the low-pass filter does affect the recovered signal because the chaotic system is sensitive to the trajectory, so any slight change in information will result in a change in the second chaotic signal being generated, thereby affecting the decryption process. The result is some missed bits after decryption.

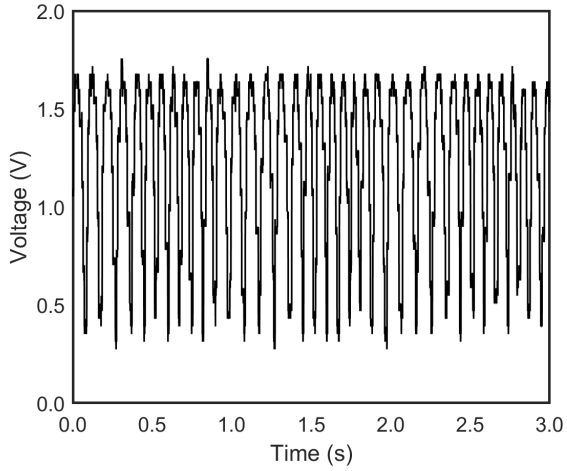
The system operates on a dual supply rail of $\pm$ 12 V and a 5 V supply rail. Total power consumption for transmitting and receiving is 2.8 W, and a breakdown of the power can be shown in Fig. 3.14. The largest contributor to power consumption is the encryption and decryption modules, consuming about 80% of the power, or 1.104 W for each board. The



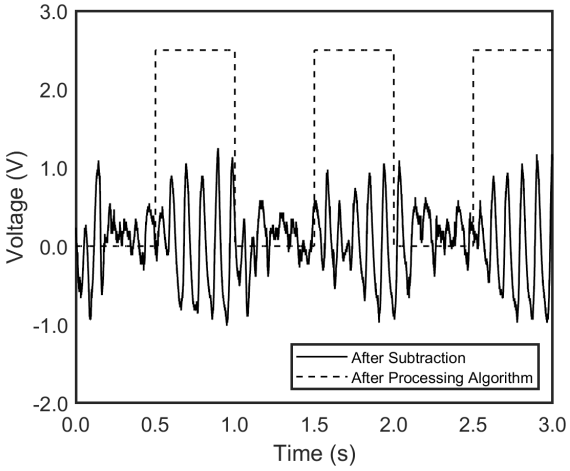
(a) Input signal.



(b) Encrypted signal at the transmitter side.

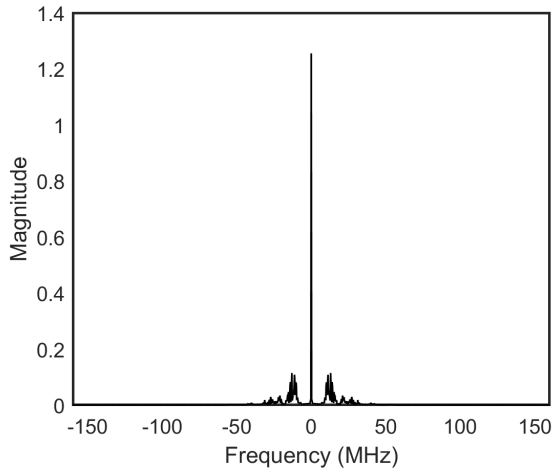


(c) Encrypted signal at the receiver side.

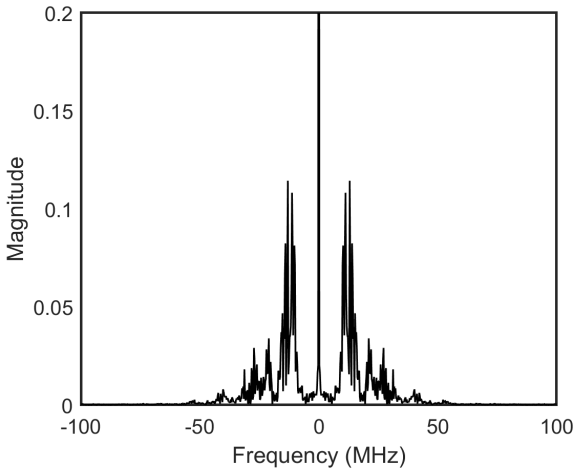


(d) Recovered signal.

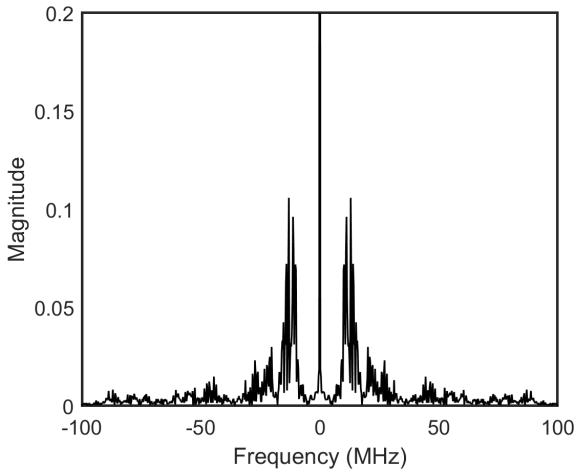
Figure 3.11: Experimental Results.



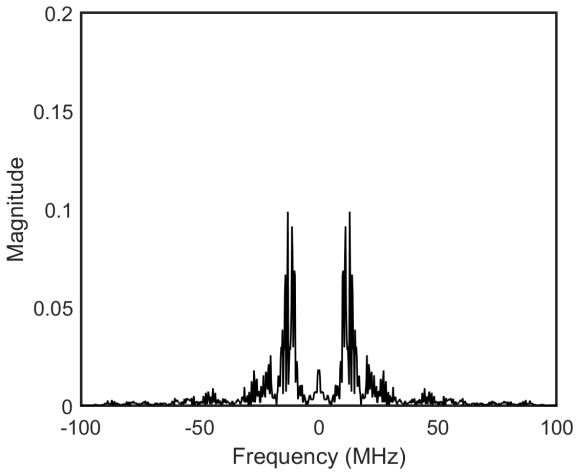
(a) The frequency domain of the encrypted signal at the transmitter side before sampling.



(b) A zoomed-in view of the encrypted signal's frequency domain (TX side).

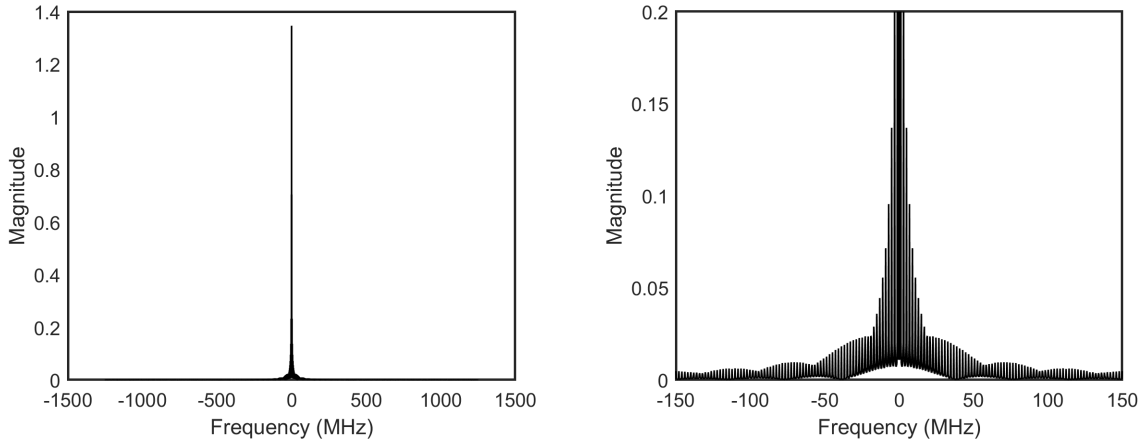


(c) The frequency domain of the encrypted signal at the receiver side after sampling.



(d) The frequency domain of the encrypted signal at the receiver side after sampling and after the low-pass filter.

Figure 3.12: Frequency Domain of the Results.



(a) A zoomed-out graph of the frequency domain. **(b)** A zoomed-in graph of the frequency domain.

Figure 3.13: Frequency domain of the input signal.

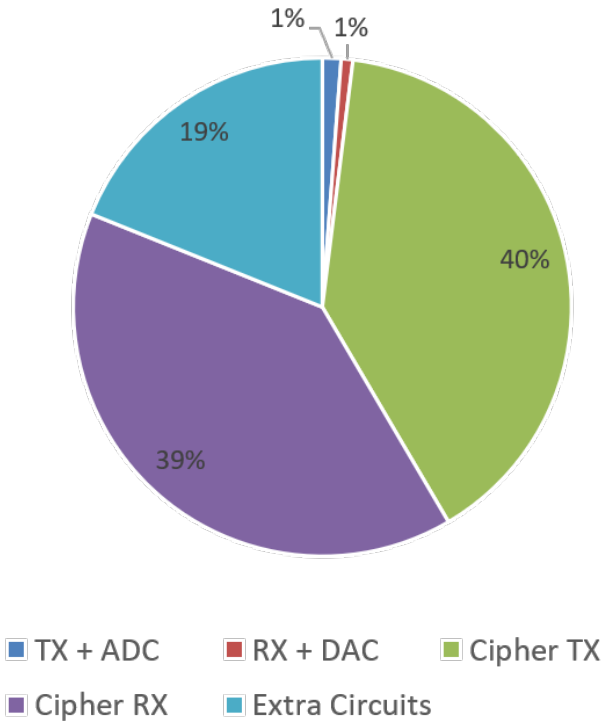


Figure 3.14: A breakdown of power consumption.

second largest contributor is the interfacing circuits, which consume 528 mW or 19% of the overall power. The transmitter and ADC consume 0.034 W, and the receiver and the DAC consume 0.021 W. Each contribute about 1% of the overall power. Power is calculated by first measuring the current through each supply rail. For the 5 V supply rail, the current was multiplied by the total voltage (5 V). For the dual supply rails, the higher current from one of the rails was multiplied by the total voltage change (24 V).

Each module in the system occupies the following area: the encryption/decryption boards occupy 131 $\times$ 56 mm^2, the transceiver evaluation boards occupy 96 $\times$ 59 mm^2, the DAC occupies 51 $\times$ 31 mm^2, and the interface circuits occupy 50 $\times$ 50 mm^2. Since the DAC sits directly on top of the receiver, it does not add any additional area to the system. The effective area for the system (in a 2D configuration) is approximately 285 cm^2. However, a stacked configuration with each module would allow for a smaller area, making the effective area of the system equal to 73.4 cm^2. Further reduction in area could be obtained through implementing the interface circuitry in SMD components on a PCB and making a custom PCB for the transceiver module instead of using an evaluation board.

Chapter 4

CMOS Low Noise Amplifier

4.1 Background

4.1.1 Radio Frequency Concepts

Scattering Parameters

In microwave theory, power quantities are usually used over voltage or current qualities for two main reasons: traditional microwave characterization relies on the amount of power that is transferred from the preceding stage to the next, and the measurements of high-frequency power quantities are more straightforward and easier to obtain than high-frequency voltages or currents [32]. For these reasons, microwave circuits are characteristics by high-frequency quantities called "scattering parameters" (S-parameters). S-parameters allow a complicated circuit or network to be modeled as a "black box" or an N port network, and they quantify how RF energy propagates through the network. When a wave encounters an impedance discontinuity within a circuit, a fraction of the wave will be reflected and the wave continuing through (incident wave) will lower in magnitude. The reflected wave can scatter to the other ports of the network. S-parameters describe this response of the network to the incident signals, and they are defined for a given frequency and system impedance. The subscripts in the s-parameter S_{ij} refer to the input and output path being measured; i refers to the responding (output) port and j refers to the incident (input) port.

In the 2-port network shown in Fig. 4.1, the coefficients a1 and a2 represent the incident wave and coefficients b1 and b2 represent the reflected waves. The S parameters are related to the waves as follows:

$$\begin{aligned} b_1 &= S_{11}a_1 + S_{12}a_2 \\ b_2 &= S_{21}a_1 + S_{22}a_2 \end{aligned} \tag{4.1}$$

Solving for each S-parameter:

$$\begin{aligned} S_{11} &= \frac{b_1}{a_1}\Big|_{a_2=0} \\ S_{12} &= \frac{b_1}{a_2}\Big|_{a_1=0} \\ S_{21} &= \frac{b_2}{a_1}\Big|_{a_2=0} \\ S_{22} &= \frac{b_2}{a_2}\Big|_{a_1=0} \end{aligned} \tag{4.2}$$

where S_{12} is reverse gain coefficient, S_{21} is forward gain coefficient, S_{11} is input reflection coefficient, and S_{22} is output reflection coefficient.

Impedance Transformation

Impedance matching is used to minimize power loss and thereby maximize power transfer. Maximum power transfer occurs when the resistance of the load equals the resistance of the source, and the load and source reactance cancel each other. Impedance matching is used to transform the impedance to meet this condition as close as possible through the use of matching networks.

Smith Chart

The Smith chart is the most widely used tool to solve transmission line problems. It also aids in designing matching networks. Shown in Fig. 4.2, the Smith chart is a polar plot of the voltage reflection coefficient (Γ), defined as:

$$\Gamma = \frac{ReflectedWave}{IncidentWave} = \frac{Z_L - Z_S}{Z_L + Z_S} \tag{4.3}$$

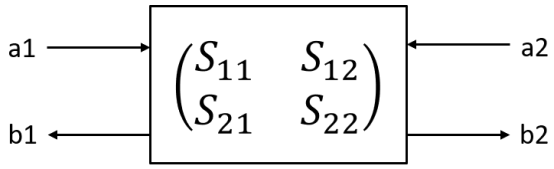


Figure 4.1: Scattering Parameters for a 2-Port Network.

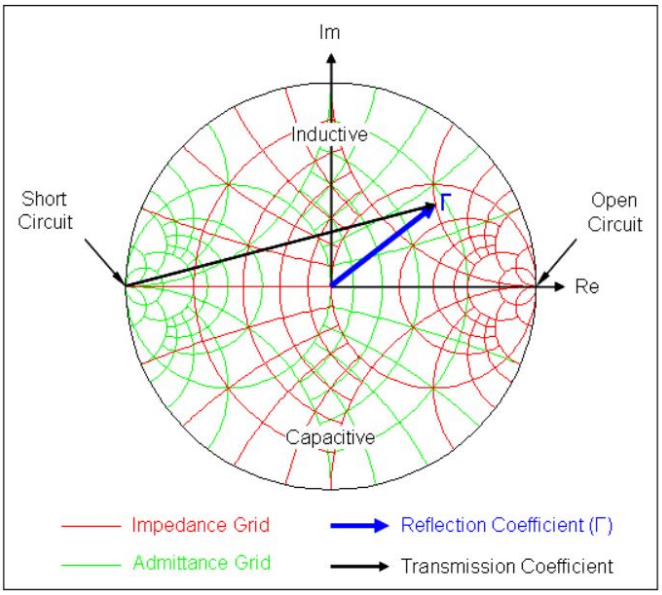


Figure 4.2: Smith Chart. [41]

where Z_S is the source impedance and Z_L is the load impedance from the impedance discontinuity. It is used to convert reflection coefficients to and from normalized impedances (or admittances) with respect to the source impedance (or admittance).

The chart contains constant resistance circles and constant reactance circles. The open circuit is on the right side of the circle (infinite resistance) and the short circuit is on the left side (zero resistance). The upper half of the circle is inductive due to the positive reactance, and lower half is capacitive due to the negative reactance.

Noise

Because noise is a random process, noise is measured by its distribution of power over a range of frequency. Thermal noise and flicker noise are the two main sources of noise for the LNA. Flicker noise is due to fluctuations in the current when charge carriers are randomly captured and released from traps at the silicon-silicon dioxide interface. Thermal noise comes from heat and is dependent on temperature. The effect of noise can be modeled as either a voltage source or current source [32]. For a resistor, the effect of noise can be modeled as a voltage source in series with the resistor R1 or a current source in parallel with the resistor R1, with a value of

$$\begin{aligned} V &= 4kTR1 \\ I &= \tfrac{4kT}{R1} \end{aligned} \tag{4.4}$$

where V is the value voltage source and I is the value of the current source, k is Boltzmann's constant, and T is temperature. For MOSFETs, thermal noise can be modeled as a voltage source in series with the gate, or a current source between the drain and source connections, with the values of

$$\begin{aligned} V &= 4kT\gamma g_m \\ I &= \tfrac{4kT\gamma}{g_m} \end{aligned} \tag{4.5}$$

for the voltage and current source, respectively. The excess noise coefficient γ is dependent on the technology process, and g_m is the transconductance of the transistor.

The contribution of noise is measured by signal-to-noise ratio (SNR), which is defined as the signal power divided by the noise power. Another main measure of noise is the noise figure (NF),

$$NF = 10log(\frac{SNR_{in}}{SNR_{out}}) \tag{4.6}$$

where SNR_{in}in is the SNR at the input of the circuit blackbox, and SNR_{out} is the SNR at the output. Another definition of NF is the total noise at the output divided by the noise at the output due to the source impedance [32], which is usually 50 Ω, making the equation,

$$NF = \frac{1}{4kTR_s}\frac{\overline{V_{n,out}^2}}{A_0^2} \tag{4.7}$$

where k is Boltzmann's constant, T is temperature (K), R_s is the source impedance (usually 50 Ω), $\overline{V_{n,out}^2}$ is output power, and A_0 is gain.

In the receiver chain, the LNA is the largest contributor to the noise figure, as its performance directly impacts the whole receiver's performance. The noise figure of the LNA directly adds to that of the receiver. The gain should be large enough to minimize the noise contribution of the following stages, but not too large as to compromise the noise figure and linearity [32].

4.1.2 IEEE 802.15.6 Standard

The first standard for BSNs is the IEEE 802.15.6 standard [3], which supports low power, short-range, and highly reliable communication. This standard defines the different frequency bands that can be used for body area networks. There are three categories for the frequency bands: human body communication (HBC), narrowband communication (NB), and ultra-wideband communication (UWB). One NB band is the Medical Implantable Communication Services (MICS) band, operating between 402-405 MHz. This band is dedicated for implantable medical devices. Other NB bands include the ISM bands 2.4 GHz and and 915 MHz. Wearable sensors most commonly use the 2.4 GHz band because many communication protocols already operate on it, such as Zigbee, Bluetooth, and Wifi. This band also allows for a small antenna and a small design. However, the body shadowing

effect can lead to significant path loss on frequencies over 1 GHz, which makes the 915 MHz an attractive alternative [42].

The design presented is a dual-band low noise amplifier that operates between 402-405 MHz (MICS band) and between 902-928 MHz (ISM band). The 915 MHz ISM band was chosen instead of the 2.4 GHz band in order to have less pass loss and less congestion due to other communication protocols. The concurrent dual-band topology was chosen to be able to operate at the desired frequencies at the same time and reject other frequency bands. The unique combination of input and output matching networks creates an LNA competitive with similar reported works in literature. The LNA could potentially be used in a receiver for both implantable and wearable sensors.

4.2 Design and Simulation

Portions of this section were published in *"A Concurrent MICS/ISM Dual-Band CMOS Low Noise Amplifier for an Integrated Body Sensor Network,"* IEEE Asia-Pacific Microwave Conference (APMC), 2020.

4.2.1 Design

The low noise amplifier, Fig. 4.3, is in a current-reuse topology. Adding the second transistor improves stability and isolation between the input and output, while "reusing" the same current from the first transistor. There is a trade-off that exists for the sizing of the width of the second transistor W2. The noise power increases as the width increases, and as the width of M2 increases, the capacitance at the node between M1 and M2 reduces. This reduced capacitance causes noise mismatch and increased equivalent noise resistance (R_n) value [43]. For this reason, the width of M2 is the same as M1. Both transistors are biased in weak inversion to yield a high transconductance. This can be estimated by the weak inversion saturation equation for MOSFETS,

$$I_D = I_{d0}\frac{W}{L}exp(\frac{V_{gs} - V_{th}}{nU_t}) \tag{4.8}$$

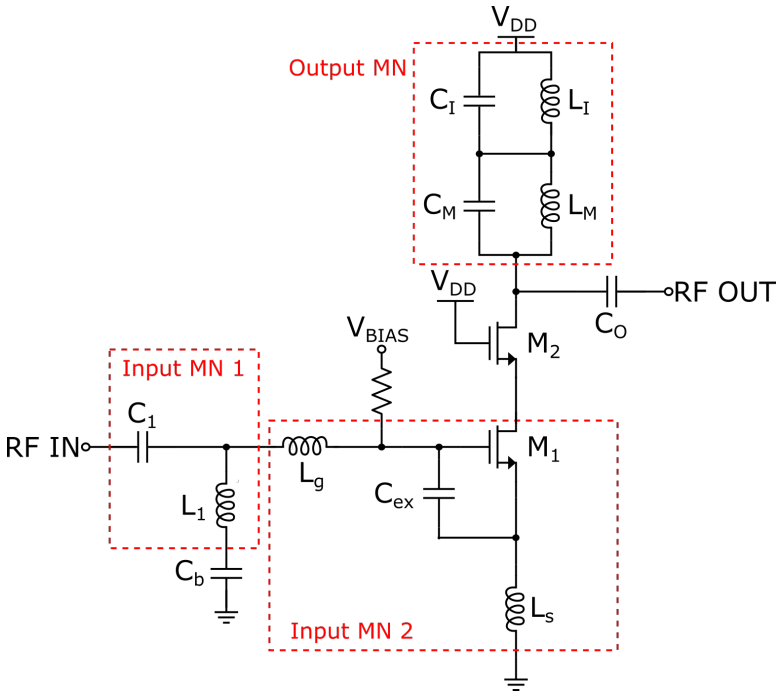


Figure 4.3: LNA Schematic.

where I_D is drain current, I_{d0} is a function of oxide capacitance and mobility, W/L is the aspect ratio, n is the subthreshold slope, and U_t is thermal voltage. The bias voltage applied to bias the transistors is 0.65 V. Capacitors C_O, C_1, and C_b are DC blocking capacitors. The gain and noise figure circles for each frequency band before any matching networks are added are shown in Fig. 4.4, which were used to help design the input and output matching networks for each band.

Input Matching Network

The input matching network combines two matching network in series. The first uses the gate inductor L_g for the 915 MHz band and the second uses an L-shaped matching network for the MICS band. The input impedance of a common source LNA is [32],

$$Z_{in1} = \frac{g_m L_s}{C_t} + j(\omega L_t - \frac{1}{\omega C_t}) \tag{4.9}$$

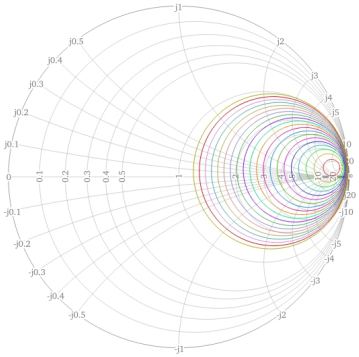
where

$$\begin{aligned} L_t &= L_g + L_s \\ C_t &= C_{gs1} + C_{ex}. \end{aligned} \tag{4.10}$$

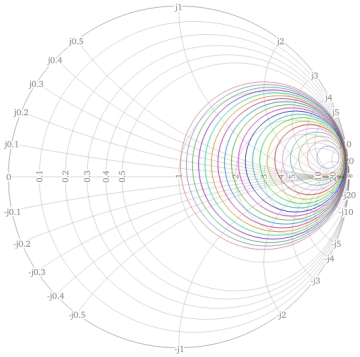
The impedance at the MICS center frequency was calculated using Eq. 4.9 and modeled as a capacitor and resistor in series (Fig. 4.5). A L-matching network was used to create a second resonance and minimize the number of components.

$$\begin{aligned} Z_{in2} &= Z_{in1}^* \\ Zin &= 50\Omega. \end{aligned} \tag{4.11}$$

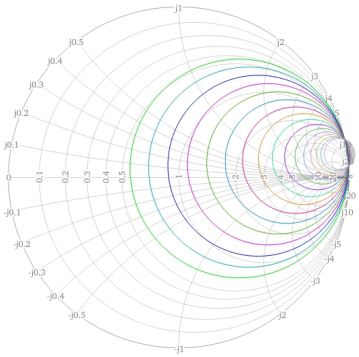
To lower load impedance, the network is in shunt with the load. A DC blocking capacitor C_1 is added to prevent current flow in the input matching network. The network values are,



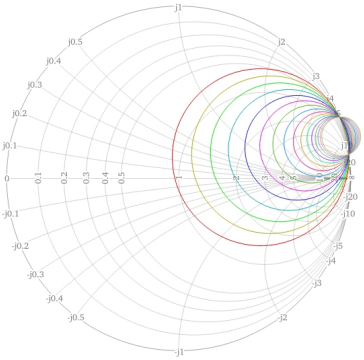
(a) Noise circles at 403.5 MHz.



(b) Noise circles at 915 MHz.



(c) Gain circles at 403.5 MHz



(d) Gain circles at 915 MHz

Figure 4.4: Gain and noise figure circles of amplifier without matching networks.

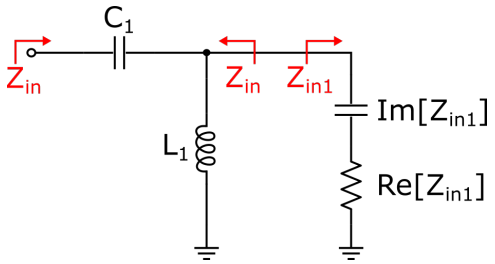


Figure 4.5: The model for the input matching network.

$$
\begin{aligned}
R_s &= Re[Zin1] \\
Q &= \tfrac{2\pi L_t}{R_s} \\
R_p &= R_s(Q^2+1) \\
C_1 &= \tfrac{1}{\omega Q R_s} \\
L_1 &= \tfrac{R_p}{Q\omega}.
\end{aligned}
\tag{4.12}
$$

Output Matching Network

Two LC tanks are added in series with one another, or two bandpass filters connected in series. Each tank produces a resonance that in-turn creates the gain. The output impedance is,

$$
Z_{out} = \left(j\omega L_I || \frac{1}{j\omega C_I}\right) + \left(j\omega L_M || \frac{1}{j\omega C_M}\right) \tag{4.13}
$$

which, after simplifying, yields

$$
Z_{out} = \frac{j\omega(L_I + L_M - \omega^2 L_I L_M C_M - \omega^2 L_I C_I L_M)}{\omega^4 L_I C_I L_M C_M - \omega^2(L_I C_I + L_M C_M) + 1} \tag{4.14}
$$

The characteristic equation can be found from Eq. 4.14, and the solutions are [43]

$$
\begin{aligned}
\omega_I &= \tfrac{1}{\sqrt{L_I C_I}} \\
\omega_M &= \tfrac{1}{\sqrt{L_M C_M}}
\end{aligned}
\tag{4.15}
$$

which is used to size the component values for the correct frequencies. The output capacitor C_O is sized to minimize S22.

4.2.2 Simulations

The equations in section II were used to derive the component values in Fig. 4.3. The design was then simulated using Cadence's Virtuoso Spectre in a commercial 1 poly 6 metal 180 nm process, and simulation results are shown in Fig. 4.6. Published simulation results do not factor in parasitic capacitances and inductances from ESD circuits, pad frame, packaging, or wire bonds.

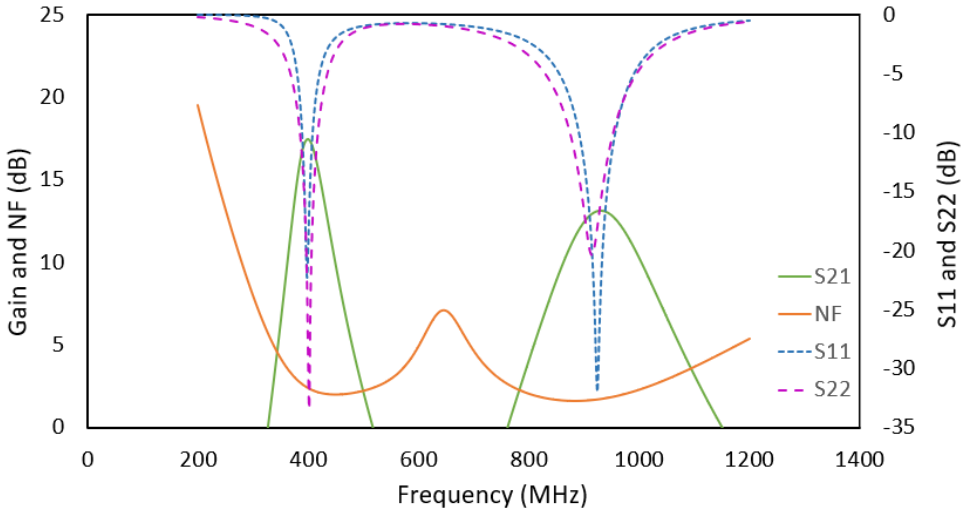


Figure 4.6: Simulated results of return loss, gain, and NF.

Using a 1.8 V supply rail, the LNA draws 2 mA of current. At 403.5 MHz, the input and output return loss are -14.7 dB and -26.3 dB, respectively. The gain is 17 dB and the NF is 2.3 dB. At 915 MHz, the input and output return loss are -18.3 and -20.9 dB, respectively. The gain is 12.7 dB and the NF is 1.7 dB.

The effects of mismatch and process variation on the RF transistors and MIM capacitors were evaluated using Monte Carlo simulations for input and output return loss, gain, and noise figure. Out of 250 runs, Monte Carlo analysis demonstrates that the design maintains a competitive range for a majority of the runs. For all runs, the gain and noise figure values stay within a competitive range. For the majority of runs, return loss stays within a competitive range. More specifically, for 86.0% of runs, input return loss stays better than 10 dB. For (94.4%) of runs, output return loss stays better than 10 dB, and the results are shown in Fig. 4.7 and tabulated in Table 4.1. If a fabricated design did have poor return loss, the performance could be tweaked by adjusting the DC blocking capacitor values, which would positively affect gain and noise figure as well.

4.3 Results

The integrated circuit (IC) was taped-out in a commercial 1 poly 6 metal 180 nm process, and the photomicrograph can be seen in Fig. 4.8. The area of the circuit is approximately 830×915 μm^2 within a 2×2 mm^2 padframe. Originally, the IC was going to be directly wire-bonded to a PCB for testing. The PCB designed in Fig. 4.9 contains matched 50 Ω transmission lines in the RF input and output paths, the off-chip input matching network, and the DC biasing with bypass capacitors to short the AC noise from the DC voltage source.The input matching network for the MICS band consisted of a 56 nH inductor at gate. The input matching network for the MICS band consisted of a 1 ρF capacitor, a 1 μF DC blocking capacitor, and a 48 nH inductor. The changes in values were to account for the parasitics from the pad frame and PCB board. The area for the IC was a 2.2×2.2 mm^2 copper trace with vias to ground to connect the substrate ground to the board's ground. The traces for wire bonding were 200 μm away from the IC and the minimum distance of 6 mil, or 152.4 μm, wide. The distance between traces was also 152.4 μm apart. Since the

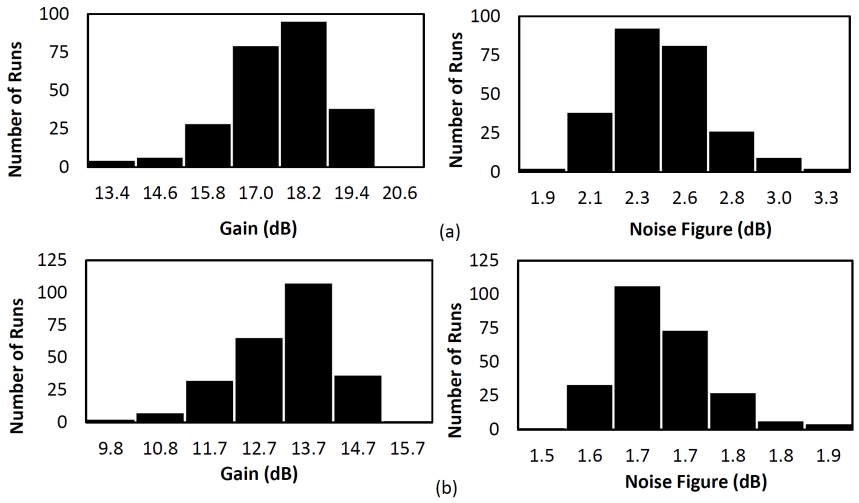


Figure 4.7: Monte Carlo results for S21 and NF at (a) 403.5 MHz and (b) 915 MHz.

Table 4.1: A summary of the Monte Carlo analysis.

	Center Freq (MHz)	Min	Max	Mean	Standard Deviation
Gain	403.5	12.78	19.17	17.01	1.2
	915	9.07	14.72	12.73	0.99
NF	403.5	1.815	3.130	2.323	0.235
	915	1.538	1.877	1.661	0.061
S11	403.5	-24.44	-4.18	-13.87	5.38
	915	-39.52	-5.55	-16.94	6.68
S22	403.5	-33.81	-6.29	-19.03	7.17
	915	-20.41	-7.13	-16.09	3.49

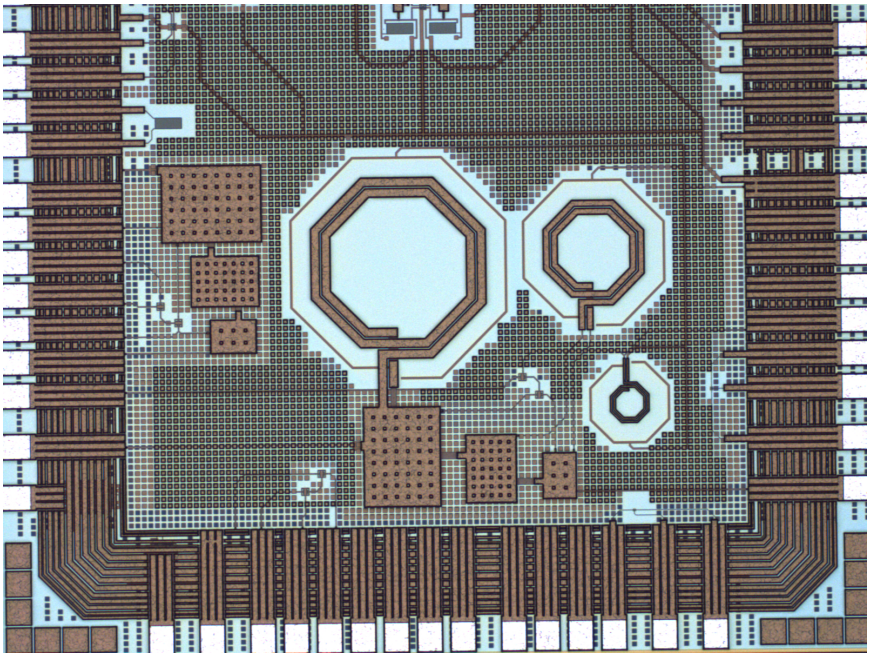


Figure 4.8: Chip Photomicrograph.

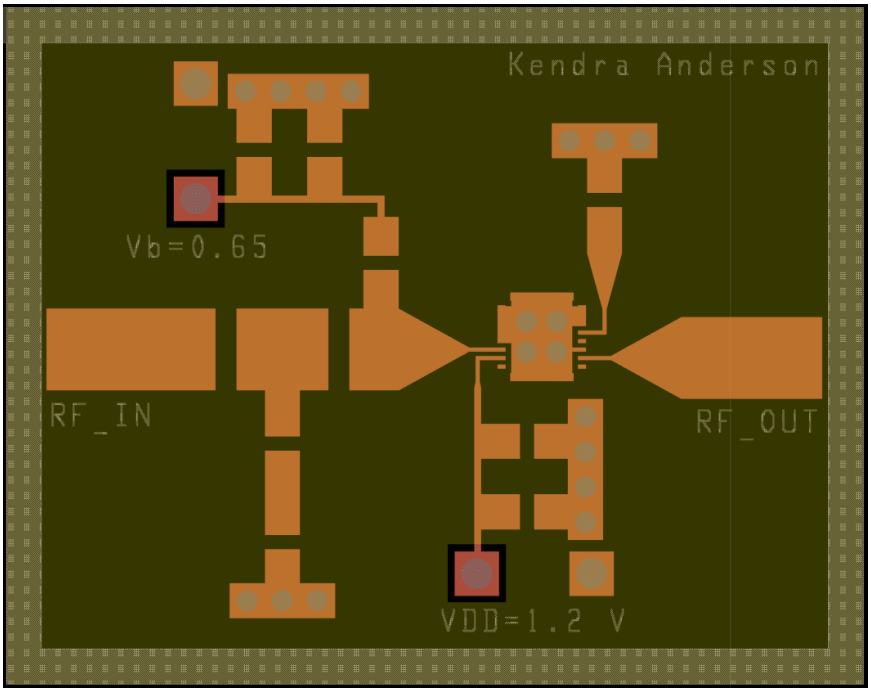


Figure 4.9: Fabricated PCB for the LNA.

IC was shared with other circuits, the other inputs were grounded. However, the wire bond machine was not used in the end because the machine was not working consistently, and no successful wire bonds could be made on the PCB traces due to the lead-free HAL finish. Future PCB boards are suggested to have an ENIG or ENIPIG finish.

The probe station was used instead of the wire bonding machine. The PCB was cut into two so that the off-chip input matching network could still be connected to the circuit, and the output port ground was connected to the PCB board ground, as shown in Fig. 4.10. A total of 4 probes were used: two DC probes for the DC biasing (VDD and Vbias) and 2 RF probes for the RF input and output pins. The RF probe has 3 connections: ground, signal, ground. Because the circuit and padframe were not designed for this, the two ground connections were left open on the probe end, as shown in Fig. 4.11. For future designs, the RF probe connections are spaces 100 μm apart, so the pads in the pad frame would need to be spaced out wider to account for this, and two ground connections would need to be placed to surround the signal connection.

The experimental results for this setup can be seen in Fig. 4.12. Fig. 4.12a and 4.12b show the input and output return loss, respectively. The output return loss shows a dip near the desired ISM band, but a negligible dip near the MICS band. This loss could be explained by parasitic capacitance from the probes, which kills the MICS band return loss and impacts the ISM band return loss. Two resonances occur at the input matching network; however, they are not at the correct frequencies. Possible explanations could be again parasitic capacitances affecting results along with parasitic inductances in the probes. Also, parasitic inductance from the solder and SMAs most likely shifted results. Because the input and output return loss is severely impacted by parasitics, the reverse and forward gain coefficients are severely impacted as well. This explains why S12 and S21 do not resemble simulation results, shown in Fig. 4.12c and Fig. 4.12d.

Another major factor that impacted performance was the open ground connections on the RF probes. The length of the RF probes and cables were roughly 140 cm combined. The wavelength of 403.5 MHz is 74.3 cm, and the wavelength of 915 MHz is 32.8 cm. This means that the electromagnetic signals have to travel over one wavelength to reach the RF ground on the other ends of the probes, resulting in degradation of the RF electromagnetic

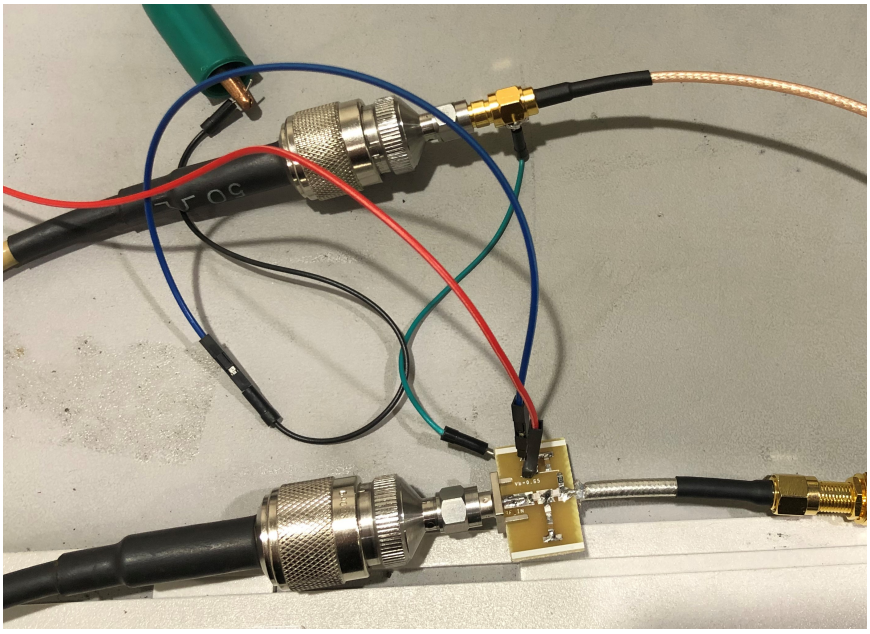


Figure 4.10: Setup to Probe Station.

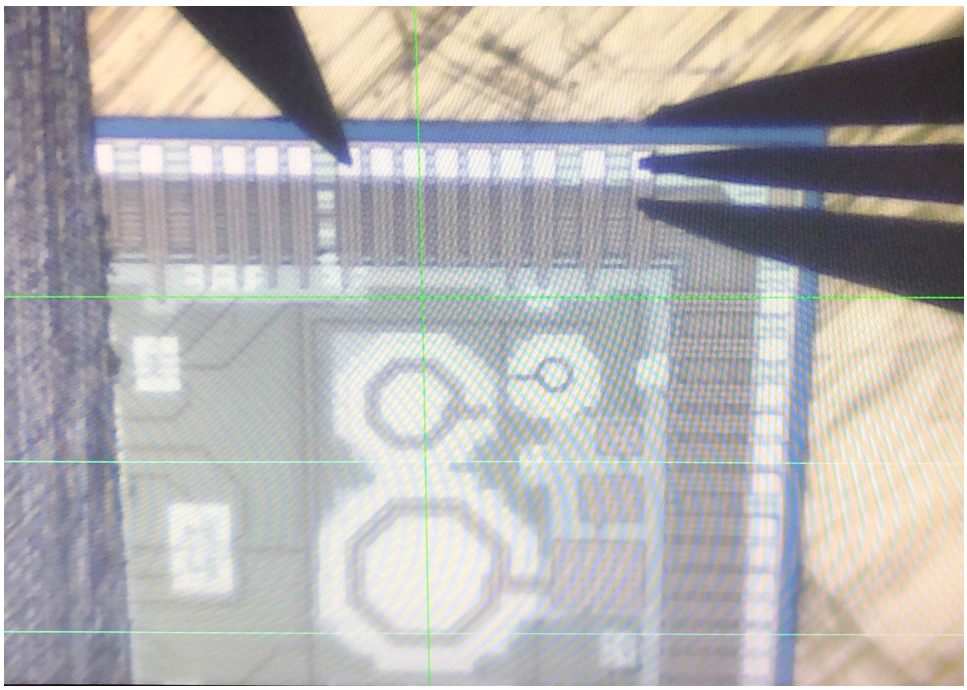
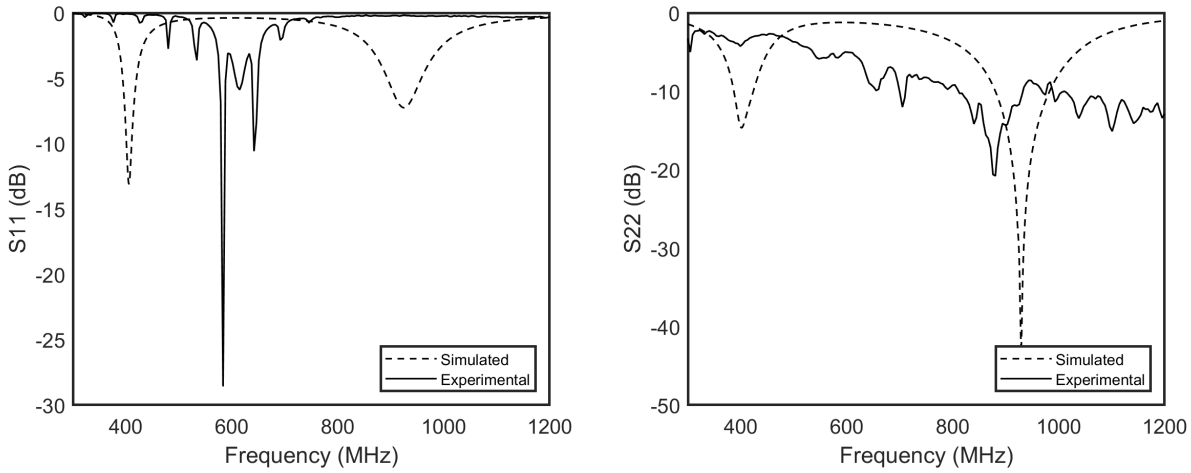
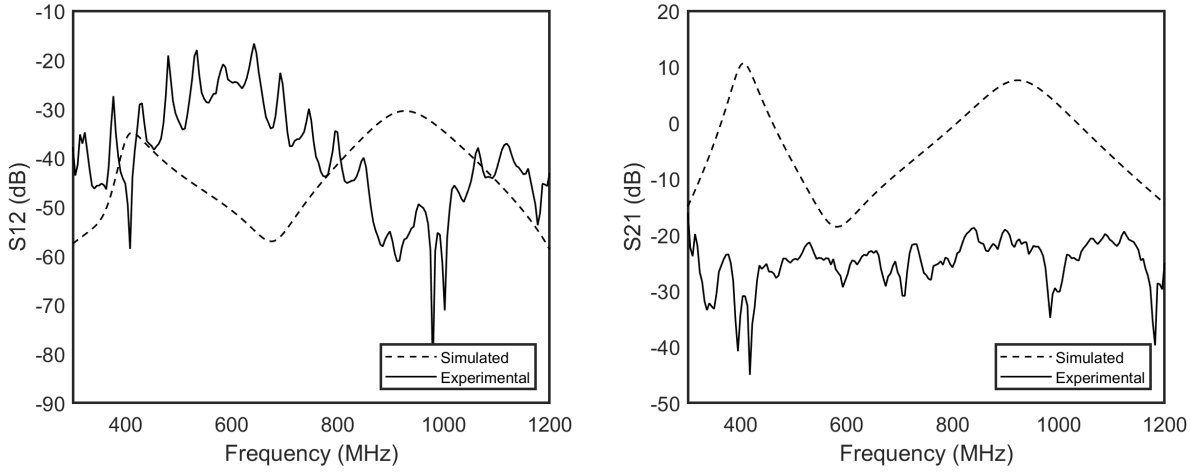


Figure 4.11: The RF and DC probe connections on the chip.



(a) S11 **(b)** S22



(c) S12 **(d)** S21

Figure 4.12: Experimental Results.

waves and negatively impacting the signal. Current draw for the chip was around 1.5 mA, which is close to the simulated 2 mA, with the difference most likely due to variations in the chip.

Chapter 5

Conclusions

This thesis detailed the first generation system implementation of a wireless chaotic communication system that uses time-scaling chaotic shift keying, which can be used for body sensor network applications. A streaming algorithm was developed and successfully integrated with the Bluetooth Low Energy communication stack. Interface circuits were designed and experimentally verified to enable integration of the encrytpion and transceiver modules. The system adds a layer of security to WSNs on top of the security methods that the communication protocol provides. Chaotic shift keying is an attractive real-time encryption method because it can be implemented in hardware using analog multipliers, operational amplifiers, resistors and capacitors using competitive power consumption to software implementations. Typical microcontroller implementations of security systems range from 2 W to 5 W; this proposed system consumes approximately 2.8 W. Chaotic shift keying can also be implemented on-chip, as proposed in [13]. This means that encryption can be taped out on the same chip as a sensor, protecting the data as soon as its produced. Power consumption is significantly reduced in IC implementations of chaotic shift keying. Another appealing property of this encryption method is that if an attacker wanted to decrypt the signal, they would need to replicate the exact chaotic system. Any variation in component values would change the system parameters and would therefore result in an inability to recover the original message signal. A replica for IC implementation chaotic systems would prove even more challenging, as fabrication of the IC costs thousands of dollars. If an attack wanted to use a return map attack to recover the message signal, the time-scaling factor in

this implementation protects against this by obscuring the changing system characteristics of the system.

The wireless system demonstrates that this particular encryption method can be used for wireless body sensor networks. Many wireless body sensor networks use Bluetooth Low Energy for their communication protocol. The main limiting factor in Bluetooth Low Energy is a maximum streaming rate of 133 Hz, which stems from the minimum connection event interval being 7.5 ms. Frequency domain analysis shows that the information in the frequency spectrum is maintained after sampling and filtering. A small change in the magnitude of the frequency information does result in occasionally missed bits due to the nature of the chaotic system being sensitive to different trajectories. To be truly competitive with wearable sensor nodes in literature, power and area need to be significantly reduced, which can be achieved by integrating the entire system. The encryption method has already been integrated and fabricated [13]. Future work will include the design and fabrication of a custom transceiver to put on the same chip as the encryption method.

This thesis also detailed the design, fabrication, and testing of a CMOS dual-band low noise amplifier. This design can be used in WBSN receivers that collect data from both implantable and wearable sensors. The design is compatible with the IEEE 802.15.6 standard for wireless body area networks. To the best of the author's knowledge, this published design is the only work in literature that operates concurrently at the Medical Implantable Communication Services (MICS) band and the 915 MHz Industrial, Scientific, and Medical (ISM) band. The design uses two LC tanks for the output matching network. For the input matching network, an inductor was added for the first band, and an L-shaped matching network was added for the second band. The matching networks were designed to transform the impedance of the input and output to yield high gain and low noise figure results. The transistors were biased in weak inversion to yield a high transconductance.

While the LNA simulation results were promising, the results did not match the simulation. Many unaccounted for factors contributed to this. First, since the IC was originally going to be directly wire bonded to a PCB, there was no parasitic model for packaging or for the probes used in the probe station. The parasitic model used was a 1 nH inductor at all of the ports (RF input, RF output, VDD, GND) for the wire

bonds. The ESD protection circuits from the padframe were simulated with the design as well. Therefore, parasitic capacitance from the probes impacted performance. Parasitic inductance from the probes probably impacted performace as well; however, the effect was not as big as the parasitic capacitance since there was some parasitic inductance model in the simulated design. The PCB for the input matching network also introduced significant parasitic inductance and thereby shifting the resonances. Even though the transmission lines were simulated in ADS after the S-parameters of the circuit was extracted from Cadence, the effect of solder and SMAs were not included in simulation. Lastly, the RF probes used in the probe station had a ground, signal, ground connection, which was not accounted for in the original design. The ground signals had to be left open, which means the RF signal did not have an immediate ground, resulting in dissipation in the electromagnetic waves. All of these factors negatively impacted performance. However, the current draw was similar to simulated current, showing that the transistors are working and turned on. Small resonances in input and output return loss show potential in LNA performance if some of the above issues were fixed. For example, packaging the IC could mitigate the effect of the ground problem, since the RF ground would be about 2 mm apart (the length of the padframe) instead of the length of the RF probe and cables apart. The input matching network could be tweaked to account for shifts from solder and SMAs to produce resonances in the correct places.

5.1 Future Work

More work can be taken to further develop this system. The following points are some possible ideas:

1. A custom PCB can be developed to include the wireless MCU, the DAC, and the buffer stages and signal processing circuitry. The developed code presented in this thesis can be uploaded onto the wireless MCU. This would make the system more compact and may improve performance due to the reduced parasitics.

2. The use of nano-fabricated electrodes can be considered to implement this system as a wearable sensor node. Also, adding more sensors to create a fuller monitoring node.

3. Further research can be taken into chaotic circuit implementations in order to increase the input frequency range of the system. In this thesis, the digital data is running at 1 Hz, which is impractical for many applications. Newer implementations of chaos need to be explored. One possibility might be finite time chaotic systems.

Bibliography

[1] A. Forster, *Introduction to Wireless Sensor Networks*. Wiley-IEEE Press, 2016. 1

[2] C. C. Y. Poon, B. P. L. Lo, M. R. Yuce, A. Alomainy, and Y. Hao, "Body sensor networks: In the era of big data and beyond," *IEEE Reviews in Biomedical Engineering*, vol. 8, pp. 4–16, 2015. 1

[3] "Ieee standard for local and metropolitan area networks - part 15.6: Wireless body area networks," *IEEE Std 802.15.6-2012*, pp. 1–271, 2012. 3, 45

[4] D. Brown, "A practical realization of a return map immune lorenz based chaotic stream cipher in circuitry," 2017. 3, 6, 36

[5] D. Brown, A. Hedayatipour, M. Majumder, G. Rose, N. McFarlane, and D. Materassi, "A practical realization of a return map immune lorenz based chaotic stream cipher in circuitry," *IET Computers & Digital Techniques*, vol. 12, no. 6, pp. 297–305, 2018. 3

[6] T. Tran and W. Chung, "High-efficient energy harvester with flexible solar panel for a wearable sensor device," *IEEE Sensors Journal*, vol. 16, no. 24, pp. 9021–90282, 2016. 8

[7] T. Wu, F. Wu, J. Redouté, and M. Yuce, "An autonomous wireless body area network implementation towards iot connected healthcare applications," *IEEE Access*, vol. 5, pp. 11413–11422, 2017. 8

[8] W. Zhang, Y. Wang, J. Zou, A. Cui, and G. Zou, "The design of wireless sensor network calling system based on zigbee," *Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 625–628, 2017. 8

[9] M. F. Shaik and M. M. Subashini, "Implementation of wearable glucose sensor node with energy harvesting for wireless body area network," in *2019 5th International Conference on Advanced Computing Communication Systems (ICACCS)*, pp. 624–627, 2019. 8

[10] B. Vaseghi, S. Mobayen, S. S. Hashemi, and A. Fekih, "Fast reaching finite time synchronization approach for chaotic systems with application in medical image encryption," *IEEE Access*, vol. 9, pp. 25911–25925, 2021. 9

[11] L. Yi, X. Tong, Z. Wang, M. Zhang, H. Zhu, and J. Liu, "A novel block encryption algorithm based on chaotic s-box for wireless sensor network," *IEEE Access*, vol. 7, pp. 53079–53090, 2019. 9

[12] N. Lord, "What is data encryption? definition, best practices & more." https://digitalguardian.com/blog/what-data-encryption, 2020. 9

[13] A. Hedayatipour, "Design and implementation of a multi-modal sensor with on-chip security," 2020. 9, 11, 59, 60

[14] N. Savage, "Quantum computers compete for supremacy." https://www.scientificamerican.com/article/quantum-computers-compete-for-supremacy/, 2017. 10

[15] K. Cuomo, A. Oppenheim, and S. Strogatz, "Synchronization of lorenz-based chaotic circuits with applications to communications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 10, pp. 626–633, 1993. 10

[16] M. Delgado-Restituto and A. Rodriguez-Vazquez, "A CMOS analog chaotic oscillator for signal encryption," *ESSCIRC '93: Nineteenth European Solid-State Circuits Conference*, pp. 110–113, 1993. 10

[17] R. Trejo-Guerra, E. Tlelo-Cuautle, M. Jimenez-Fuentes, and C. Sánchez-López, "Multiscroll oscillator based on floating gate CMOS inverter," *International Conference on Electrical Engineering Computing Science and Automatic Control*, pp. 541–545, 2010. 10

[18] R. Trejo-Guerra and E. Tlelo-Cuautle, "On the frequency limitations of fgmos transitor-based integrated chaotic oscillators," *International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 458–462, 2013. 10

[19] O. Gonzales, G. Han, J. de Gyvez, and E. Sanchez-Sinencio, "Lorenz-based chaotic cryptosystem: a monolithic implementation," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 8, pp. 1243–1247, 2000. 10

[20] Y. Wu, Y. Yang, Y. Li, and C. Wu, "Nonlinear dynamic analysis and chip implementation of a new chaotic oscillator," *IEEE International Conference on Networking, Sensing and Control*, pp. 554–559, 2015. 11

[21] M. Dar, N. Kant, and F. Khanday, "Realization of fractional-order doublescroll chaotic system using operational transconductance amplifier," *Journal of Circuits, Systems and Computers,*, vol. 27, no. 1, p. 1850006, 2018. 11

[22] V. Carbajal-Gomez, E. Tlelo-Cuautle, J. Mu noz Pacheco, L. de la Fraga, C. Sanchez-Lopez, and F. ernandez Fernandez, "Optimization and CMOS design of chaotic oscillators robust to pvt variations," *Integration*, vol. 65, pp. 32–42, 2019. 11

[23] T. Belkhouja, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani, "Light-weight encryption of wireless communication for implantable medical devices using henon chaotic system," *International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pp. 1–6, 2017. 11

[24] M. Azzaz, C. Tanougast, S. Sadoudi, and A. Dandache, "Real-time FPGA implementation of lorenz's chaotic generator for ciphering telecommunications," *Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*, pp. 1–4, 2009. 11

[25] A. El-Maksoud, A. El-Kader, B. Hassan, M. Abdelhamed, N. Rihan, M. Tolba, L. Said, A. Radwan, and M. Abu-Elyazeed, "FPGA implementation of fractional-order chua's chaotic system," *7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, 2018. 11

[26] J. Pan, N. Qi, B. Xue, and Q. Ding, "Design and hardware implementation of FPGA & chaotic encryption-based wireless transmission system," *First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pp. 691–695, 2011. 11

[27] H. Chen, B. Liau, and Y. Hou, "Hardware implementation of lorenz circuit systems for secure chaotic communication applications," *Sensors*, vol. 13, no. 2, pp. 2494–2505, 2013. 11

[28] L. Xiong, Y. Lu, Y. Zhang, X. Zhang, and P. Gupta, "Design and hardware implementation of a new chaotic secure communication technique," *PloS One*, vol. 11, no. 8, p. e0158348, 2016. 11

[29] O. Eslamifar and R. Shirazi, "Malek, m. and saini, s.," *International Conference on Signal Processing and Communication Engineering Systems*, pp. 157–161, 2015. 12

[30] X. Luo, W. Feng, H. Zhu, L. Wu, W. Che, and Q. Xue, "A 21-41 ghz compact wideband low-noise amplifier based on transformer-feedback technique in 65-nm CMOS," *IEEE Asia-Pacific Microwave Conference (APMC)*, pp. 92–94, 2020. 12

[31] M. Kusuma, S. Shanthala, and R. Prasanna, "A 3-6 ghz CMOS low noise amplifier with 14.8 db gain and 3 dB noise figure for UWB applications," *International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 1430–1433, 2019. 12

[32] B. Razavi, *RF Microelectronics*. Prentice-Hall, 2011. 12, 13, 41, 44, 45, 48

[33] G. Cheng, Z. Li, L. Luo, and Z. Wang, "A 2–3 ghz high gain and high linearity current-reused LNA with wideband input matching," *International Conference On Communication Problem-Solving (ICCP)*, pp. 1–2, 2016. 13

[34] B. Guo, J. Chen, Y. Li, H. Jin, Y. Yang, and W. Chen, "A wideband common-gate LNA with enhanced linearity by using complementary mgtr technique," *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1540–1542, 2016. 13

[35] L. Lian, N. Noh, M. Mustaffa, A. Manaf, and O. Sidek, "A dual-band LNA with 0.18-μm CMOS switches," *IEEE Regional Symposium on Micro and Nano Electronics*, pp. 172–176, 2011. 13

[36] J. Borremans, P. Wambacq, G. Van der Plas, Y. Rolain, and M. Kuijk, "A switchable low-area 2.4-and-5 ghz dual-band LNA in digital CMOS," *European Solid-State Circuits Conference*, pp. 376–379, 2007. 13

[37] X. Xing, P. Cao, H. Feng, and Z. Wang, "A 0.9/1.8/2.4ghz-reconfigurable LNA with inductor and capacitor tuning for iot application in 65nm CMOS," *IEEE International Conference on ASIC (ASICON)*, pp. 1–4, 2019. 13

[38] S. Sattar and T. Zulkifli, "A 2.4/5.2-GHz concurrent dual-band CMOS low noise amplifier," *IEEE Access*, vol. 5, pp. 21148–21156, 2017. ix, 15

[39] T. Instruments, "Ble5-stack user's guide." https://dev.ti.com/tirex/explore/node?node=AOimuSWjap.4RuDbcp7OqA__pTTHBmu__LATEST, 2020. viii, ix, 16, 19, 21, 22, 25, 27

[40] K. Thiyari and H. Pandey, "Layers of OSI model." https://www.geeksforgeeks.org/layers-of-osi-model/, 2020. ix, 17, 18

[41] U. Editor, "Smith chart basics." https://www.microwaves101.com/encyclopedias/smith-chart-basics, 2. ix, 43

[42] N. Cho, J. Bae, and H. Yoo, "A 10.8 mw body channel communication/mics dual-band transceiver for a unified body sensor network controller," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 12, pp. 3459–3468, 2009. 46

[43] S. Sattar and T. Z. A. Zulkifli, "A 2.4/5.2-GHz concurrent dual-band CMOS low noise amplifier," *IEEE Access*, vol. 5, pp. 21148–21156, 2017. 46, 50

Appendices

A Application Code

The base code is from Texas Instruments and developed using Code Composer Studio version 10.2.1.00009 and Sensor Controller Studio version 2.6.0.132. The base project files are Project Zero for the transmitter and Host Test App for the receiver, both from SDK 5.10.00.48. The additional system configuration settings that were added are summarized for the transmitter and the receiver in Table 1 and Table 2, respectively. The subsections show the files within the project, and the numbers are the line numbers where the code is added.

Table 1: Added System Configuration Tool Settings for the Transmitter's Code (Project Zero).

Added Driver	Instance Name	Pin Number	Notes
ADCBuff	CONFIG_ADCBUF_0	DIO 30	–

Table 2: Added System Configuration Tool Settings for the Receiver's Code (Host Test App).

Added Driver	Instance Name	Pin Number	Notes
GPIO	CONFIG_GPIO_SS*	DIO 0	Output, Low strength,
	CONFIG_GPIO_CLR	DIO 15	High initial state
SPI	CONFIG_SPI_MASTER	SCLK: DIO 10 MISO: DIO 8 MOSI: DIO 9	Uses LaunchPad SPI Bus hardware, three pin mode

***Note: DIO 0 should be externally connected to DIO 14 to connect correctly to the DAC.** The external connection avoids hardware conflict.

A.1 cipherService.h

```
1  #ifndef _CIPHERSERVICE_H_
2  #define _CIPHERSERVICE_H_
3
4  #ifdef __cplusplus
5  extern "C"
6  {
7  #endif
8
9  /****************************
10  * INCLUDES
11  */
12
13 /****************************
14 * CONSTANTS
15 */
16
17 // Service UUID
18 #define CIPHERSERVICE_SERV_UUID 0xBA55
19
20 //  CipherValue Characteristic defines
21 #define CIPHERSERVICE_CIPHERVALUE_ID   0
22 #define CIPHERSERVICE_CIPHERVALUE_UUID 0x2BAD
23 #define CIPHERSERVICE_CIPHERVALUE_LEN  20
24
25 //  Characteristic defines
26 #define CIPHERSERVICE_STREAMEN_ID   1
27 #define CIPHERSERVICE_STREAMEN_UUID 0x2BAE
28 #define CIPHERSERVICE_STREAMEN_LEN  1
29
30 /****************************
31  * TYPEDEFS
32  */
33
34 /****************************
```

```
35  * MACROS
36  */
37
38 /****************************
39  * Profile Callbacks
40  */
41
42 // Callback when a characteristic value has changed
43 typedef void (*cipherServiceChange_t)(uint16_t connHandle, uint8_t
       ↪ paramID, uint16_t len, uint8_t *pValue);
44
45 typedef struct
46 {
47   cipherServiceChange_t        pfnChangeCb;  // Called when
       ↪ characteristic value changes
48   cipherServiceChange_t        pfnCfgChangeCb;
49 } cipherServiceCBs_t;
50
51
52
53 /****************************
54  * API FUNCTIONS
55  */
56
57
58 /*
59  * CipherService_AddService- Initializes the CipherService service
       ↪ by registering
60  *          GATT attributes with the GATT server.
61  *
62  */
63 extern bStatus_t CipherService_AddService( uint8_t rspTaskId);
64
65 /*
66  * CipherService_RegisterAppCBs - Registers the application
       ↪ callback function.
```

```
67  *                    Only call this function once.
68  *
69  *    appCallbacks - pointer to application callbacks.
70  */
71 extern bStatus_t CipherService_RegisterAppCBs( cipherServiceCBs_t *
       ↪ appCallbacks );
72
73 /*
74  * CipherService_SetParameter - Set a CipherService parameter.
75  *
76  *    param - Profile parameter ID
77  *    len - length of data to right
78  *    value - pointer to data to write.  This is dependent on
79  *          the parameter ID and WILL be cast to the appropriate
80  *          data type (example: data type of uint16 will be cast to
81  *          uint16 pointer).
82  */
83 extern bStatus_t CipherService_SetParameter(uint8_t param, uint16_t
       ↪  len, void *value);
84
85 /*
86  * CipherService_GetParameter - Get a CipherService parameter.
87  *
88  *    param - Profile parameter ID
89  *    value - pointer to data to write.  This is dependent on
90  *          the parameter ID and WILL be cast to the appropriate
91  *          data type (example: data type of uint16 will be cast to
92  *          uint16 pointer).
93  */
94 extern bStatus_t CipherService_GetParameter(uint8_t param, uint16_t
       ↪  *len, void *value);
95
96 /*************************************
97 *************************************/
98
99 #ifdef __cplusplus
```

```
100 }
101 #endif
102
103 #endif /* _CIPHERSERVICE_H_ */
```

A.2 cipherService.c

```
1  /****************************
2   * INCLUDES
3   */
4  #include <string.h>
5
6  #include <icall.h>
7
8  /* This Header file contains all BLE API and icall structure
       ↪ definition */
9  #include "icall_ble_api.h"
10
11 #include "cipherService.h"
12
13 /****************************
14  * MACROS
15  */
16
17 /****************************
18  * CONSTANTS
19  */
20
21 /****************************
22  * TYPEDEFS
23  */
24
25 /****************************
26 * GLOBAL VARIABLES
27 */
```

```
28
29 // cipherService Service UUID
30 CONST uint8_t cipherServiceUUID[ATT_BT_UUID_SIZE] =
31 {
32   LO_UINT16(CIPHERSERVICE_SERV_UUID), HI_UINT16(
       ↪ CIPHERSERVICE_SERV_UUID)
33 };
34
35 // cipherValue UUID
36 CONST uint8_t cipherService_CipherValueUUID[ATT_UUID_SIZE] =
37 {
38   TI_BASE_UUID_128(CIPHERSERVICE_CIPHERVALUE_UUID)
39 };
40 // streamEN UUID
41 CONST uint8_t cipherService_StreamENUUID[ATT_UUID_SIZE] =
42 {
43   TI_BASE_UUID_128(CIPHERSERVICE_STREAMEN_UUID)
44 };
45
46 /*****************************
47  * LOCAL VARIABLES
48  */
49
50 static cipherServiceCBs_t *pAppCBs = NULL;
51
52 /*****************************
53 * Profile Attributes - variables
54 */
55
56 // Service declaration
57 static CONST gattAttrType_t cipherServiceDecl = { ATT_BT_UUID_SIZE,
       ↪  cipherServiceUUID };
58
59 // Characteristic "CipherValue" Properties (for declaration)
60 static uint8_t cipherService_CipherValueProps = GATT_PROP_READ |
       ↪ GATT_PROP_NOTIFY;
```

```
61
62 // Characteristic "CipherValue" Value variable
63 static uint8_t cipherService_CipherValueVal[
      ↪ CIPHERSERVICE_CIPHERVALUE_LEN] = {0};
64
65 // Characteristic "CipherValue" CCCD
66 static gattCharCfg_t *cipherService_CipherValueConfig;
67 // Characteristic "StreamEN" Properties (for declaration)
68 static uint8_t cipherService_StreamENProps = GATT_PROP_READ |
      ↪ GATT_PROP_WRITE_NO_RSP;
69 //| GATT_PROP_WRITE
70
71 // Characteristic "StreamEN" Value variable
72 static uint8_t cipherService_StreamENVal[CIPHERSERVICE_STREAMEN_LEN
      ↪ ] = {0};
73
74 /*****************************
75 * Profile Attributes - Table
76 */
77
78 static gattAttribute_t cipherServiceAttrTbl[] =
79 {
80   // cipherService Service Declaration
81   {
82     { ATT_BT_UUID_SIZE, primaryServiceUUID },
83     GATT_PERMIT_READ,
84     0,
85     (uint8_t *)&cipherServiceDecl
86   },
87     // CipherValue Characteristic Declaration
88     {
89       { ATT_BT_UUID_SIZE, characterUUID },
90       GATT_PERMIT_READ,
91       0,
92       &cipherService_CipherValueProps
93     },
```

```
94        // CipherValue Characteristic Value
95        {
96          { ATT_UUID_SIZE, cipherService_CipherValueUUID },
97          GATT_PERMIT_READ,
98          0,
99          cipherService_CipherValueVal
100       },
101       // CipherValue CCCD
102       {
103         { ATT_BT_UUID_SIZE, clientCharCfgUUID },
104         GATT_PERMIT_READ | GATT_PERMIT_WRITE,
105         0,
106         (uint8 *)&cipherService_CipherValueConfig
107       },
108     // StreamEN Characteristic Declaration
109     {
110       { ATT_BT_UUID_SIZE, characterUUID },
111       GATT_PERMIT_READ,
112       0,
113       &cipherService_StreamENProps
114     },
115       // StreamEN Characteristic Value
116       {
117         { ATT_UUID_SIZE, cipherService_StreamENUUID },
118         GATT_PERMIT_READ | GATT_PERMIT_WRITE,
119         0,
120         cipherService_StreamENVal
121       },
122 };
123
124 /*****************************
125  * LOCAL FUNCTIONS
126  */
127 static bStatus_t cipherService_ReadAttrCB( uint16_t connHandle,
    ↪ gattAttribute_t *pAttr,
```

```
128                                                 uint8_t *pValue,
                                                       ↪ uint16_t *pLen,
                                                       ↪ uint16_t offset,
129                                                 uint16_t maxLen, uint8_t
                                                       ↪  method );
130 static bStatus_t cipherService_WriteAttrCB( uint16_t connHandle,
       ↪ gattAttribute_t *pAttr,
131                                                  uint8_t *pValue,
                                                       ↪ uint16_t len,
                                                       ↪ uint16_t offset,
132                                                  uint8_t method );
133
134 /*****************************
135  * PROFILE CALLBACKS
136  */
137 // Simple Profile Service Callbacks
138 CONST gattServiceCBs_t cipherServiceCBs =
139 {
140   cipherService_ReadAttrCB,  // Read callback function pointer
141   cipherService_WriteAttrCB, // Write callback function pointer
142   NULL                       // Authorization callback function
        ↪ pointer
143 };
144
145 /*****************************
146 * PUBLIC FUNCTIONS
147 */
148
149 /*
150  * CipherService_AddService- Initializes the CipherService service
        ↪ by registering
151  *          GATT attributes with the GATT server.
152  *
153  */
154 extern bStatus_t CipherService_AddService( uint8_t rspTaskId )
155 {
```

```
156   uint8_t status;
157
158   // Allocate Client Characteristic Configuration table
159   cipherService_CipherValueConfig = (gattCharCfg_t *)ICall_malloc(
          ↪ sizeof(gattCharCfg_t) * linkDBNumConns );
160   if ( cipherService_CipherValueConfig == NULL )
161   {
162     return ( bleMemAllocError );
163   }
164
165   // Initialize Client Characteristic Configuration attributes
166   GATTServApp_InitCharCfg( LINKDB_CONNHANDLE_INVALID,
          ↪ cipherService_CipherValueConfig );
167   // Register GATT attribute list and CBs with GATT Server App
168   status = GATTServApp_RegisterService( cipherServiceAttrTbl,
169                                         GATT_NUM_ATTRS(
                                                ↪ cipherServiceAttrTbl
                                                ↪ ),
170                                         GATT_MAX_ENCRYPT_KEY_SIZE,
171                                         &cipherServiceCBs );
172
173   return ( status );
174 }
175
176 /*
177  * CipherService_RegisterAppCBs - Registers the application
          ↪ callback function.
178  *                    Only call this function once.
179  *
180  *    appCallbacks - pointer to application callbacks.
181  */
182 bStatus_t CipherService_RegisterAppCBs( cipherServiceCBs_t *
       ↪ appCallbacks )
183 {
184   if ( appCallbacks )
185   {
```

```
186     pAppCBs = appCallbacks;
187
188     return ( SUCCESS );
189   }
190   else
191   {
192     return ( bleAlreadyInRequestedMode );
193   }
194 }
195
196 /*
197  * CipherService_SetParameter - Set a CipherService parameter.
198  *
199  *    param - Profile parameter ID
200  *    len - length of data to right
201  *    value - pointer to data to write.  This is dependent on
202  *          the parameter ID and WILL be cast to the appropriate
203  *          data type (example: data type of uint16 will be cast to
204  *          uint16 pointer).
205  */
206 bStatus_t CipherService_SetParameter( uint8_t param, uint16_t len,
      ↪ void *value )
207 {
208   bStatus_t ret = SUCCESS;
209   switch ( param )
210   {
211     case CIPHERSERVICE_CIPHERVALUE_ID:
212       if ( len == CIPHERSERVICE_CIPHERVALUE_LEN )
213       {
214         memcpy(cipherService_CipherValueVal, value, len);
215
216 //        Log_info0("In SetParameter beofre notifications");
217
218         // Try to send notification.
219         GATTServApp_ProcessCharCfg( cipherService_CipherValueConfig
              ↪ , (uint8_t *)&cipherService_CipherValueVal, FALSE,
```

```
220                                     cipherServiceAttrTbl,
                                           ↪ GATT_NUM_ATTRS(
                                           ↪ cipherServiceAttrTbl ),
221                                     INVALID_TASK_ID,
                                           ↪ cipherService_ReadAttrCB)
                                           ↪ ;
222
223 //        Log_info0("In SetParameter after notifications");
224       }
225       else
226       {
227         ret = bleInvalidRange;
228       }
229       break;
230
231     case CIPHERSERVICE_STREAMEN_ID:
232       if ( len == CIPHERSERVICE_STREAMEN_LEN )
233       {
234         memcpy(cipherService_StreamENVal, value, len);
235       }
236       else
237       {
238         ret = bleInvalidRange;
239       }
240       break;
241
242     default:
243       ret = INVALIDPARAMETER;
244       break;
245   }
246   return ret;
247 }
248
249
250 /*
251  * CipherService_GetParameter - Get a CipherService parameter.
```

```
252  *
253  *    param - Profile parameter ID
254  *    value - pointer to data to write.  This is dependent on
255  *          the parameter ID and WILL be cast to the appropriate
256  *          data type (example: data type of uint16 will be cast to
257  *          uint16 pointer).
258  */
259 bStatus_t CipherService_GetParameter( uint8_t param, uint16_t *len,
       ↪  void *value )
260 {
261   bStatus_t ret = SUCCESS;
262   switch ( param )
263   {
264     case CIPHERSERVICE_STREAMEN_ID:
265       memcpy(value, cipherService_StreamENVal,
            ↪ CIPHERSERVICE_STREAMEN_LEN);
266       break;
267
268     default:
269       ret = INVALIDPARAMETER;
270       break;
271   }
272   return ret;
273 }
274
275
276 /*****************************
277  * @fn          cipherService_ReadAttrCB
278  *
279  * @brief       Read an attribute.
280  *
281  * @param       connHandle - connection message was received on
282  * @param       pAttr - pointer to attribute
283  * @param       pValue - pointer to data to be read
284  * @param       pLen - length of data to be read
285  * @param       offset - offset of the first octet to be read
```

```
286   * @param       maxLen - maximum length of data to be read
287   * @param       method - type of read message
288   *
289   * @return      SUCCESS, blePending or Failure
290   */
291 static bStatus_t cipherService_ReadAttrCB( uint16_t connHandle,
      ↪ gattAttribute_t *pAttr,
292                                            uint8_t *pValue, uint16_t *
                                                 ↪ pLen, uint16_t offset,
293                                            uint16_t maxLen, uint8_t
                                                 ↪ method )
294 {
295   bStatus_t status = SUCCESS;
296
297   // See if request is regarding the CipherValue Characteristic
        ↪ Value
298 if ( ! memcmp(pAttr->type.uuid, cipherService_CipherValueUUID,
      ↪ pAttr->type.len) )
299   {
300     if ( offset > CIPHERSERVICE_CIPHERVALUE_LEN )  // Prevent
          ↪ malicious ATT ReadBlob offsets.
301     {
302       status = ATT_ERR_INVALID_OFFSET;
303     }
304     else
305     {
306       *pLen = MIN(maxLen, CIPHERSERVICE_CIPHERVALUE_LEN - offset);
            ↪  // Transmit as much as possible
307       memcpy(pValue, pAttr->pValue + offset, *pLen);
308     }
309   }
310   // See if request is regarding the StreamEN Characteristic Value
311 else if ( ! memcmp(pAttr->type.uuid, cipherService_StreamENUUID,
      ↪ pAttr->type.len) )
312   {
```

```
313     if ( offset > CIPHERSERVICE_STREAMEN_LEN )  // Prevent
            ↪ malicious ATT ReadBlob offsets.
314     {
315       status = ATT_ERR_INVALID_OFFSET;
316     }
317     else
318     {
319       *pLen = MIN(maxLen, CIPHERSERVICE_STREAMEN_LEN - offset);  //
              ↪  Transmit as much as possible
320       memcpy(pValue, pAttr->pValue + offset, *pLen);
321     }
322   }
323   else
324   {
325     // If we get here, that means you've forgotten to add an if
            ↪ clause for a
326     // characteristic value attribute in the attribute table that
            ↪ has READ permissions.
327     *pLen = 0;
328     status = ATT_ERR_ATTR_NOT_FOUND;
329   }
330
331   return status;
332 }
333
334
335 /*****************************
336  * @fn      cipherService_WriteAttrCB
337  *
338  * @brief   Validate attribute data prior to a write operation
339  *
340  * @param   connHandle - connection message was received on
341  * @param   pAttr - pointer to attribute
342  * @param   pValue - pointer to data to be written
343  * @param   len - length of data
344  * @param   offset - offset of the first octet to be written
```

```
345   * @param   method - type of write message
346   *
347   * @return  SUCCESS, blePending or Failure
348   */
349 static bStatus_t cipherService_WriteAttrCB( uint16_t connHandle,
        ↪ gattAttribute_t *pAttr,
350                                             uint8_t *pValue, uint16_t
                                                   ↪ len, uint16_t offset,
351                                             uint8_t method )
352 {
353   bStatus_t status  = SUCCESS;
354   uint8_t   paramID = 0xFF;
355
356   // See if request is regarding a Client Characterisic
        ↪ Configuration
357   if ( ! memcmp(pAttr->type.uuid, clientCharCfgUUID, pAttr->type.
        ↪ len) )
358   {
359     // Allow only notifications.
360     status = GATTServApp_ProcessCCCWriteReq( connHandle, pAttr,
          ↪ pValue, len, offset, GATT_CLIENT_CFG_NOTIFY);
361   }
362   // See if request is regarding the StreamEN Characteristic Value
363   else if ( ! memcmp(pAttr->type.uuid, cipherService_StreamENUUID,
        ↪ pAttr->type.len) )
364   {
365     if ( offset + len > CIPHERSERVICE_STREAMEN_LEN )
366     {
367       status = ATT_ERR_INVALID_OFFSET;
368     }
369     else
370     {
371       // Copy pValue into the variable we point to from the
            ↪ attribute table.
372       memcpy(pAttr->pValue + offset, pValue, len);
373
```

```
374         // Only notify application if entire expected value is
                ↪ written
375         if ( offset + len == CIPHERSERVICE_STREAMEN_LEN)
376           paramID = CIPHERSERVICE_STREAMEN_ID;
377       }
378     }
379     else
380     {
381       // If we get here, that means you've forgotten to add an if
              ↪ clause for a
382       // characteristic value attribute in the attribute table that
              ↪ has WRITE permissions.
383       status = ATT_ERR_ATTR_NOT_FOUND;
384     }
385
386     // Let the application know something changed (if it did) by
          ↪ using the
387     // callback it registered earlier (if it did).
388     if (paramID != 0xFF)
389       if ( pAppCBs && pAppCBs->pfnChangeCb )
390         pAppCBs->pfnChangeCb(connHandle, paramID, len, pValue); //
                ↪ Call app function from stack task context.
391
392     return status;
393 }
```

A.3 project_zero.c

```
52 #include <ti/drivers/SPI.h>
53 #include <ti/drivers/GPIO.h>
54 #include <ti/drivers/ADCBuf.h>
```

...

```
101 #include "services/cipherService.h"
102 #include "scif.h"
```

```
103 #define BV(x)    (1 << (x))
```

...

```
115 /*****************************
116  * CONSTANTS
117  */
118
119 #define ADCSAMPLESIZE    (CIPHERSERVICE_CIPHERVALUE_LEN/2)
120
121 uint16_t sampleBufferOne[ADCSAMPLESIZE];
122 uint16_t sampleBufferTwo[ADCSAMPLESIZE];
123 uint8_t  updateArray[2*ADCSAMPLESIZE];
124
125 ADCBuf_Handle adcBuf;
126 ADCBuf_Conversion continuousConversion;
```

...

```
163 #define PZ_SC_CTRL_READY        11  /* Sensor controller control
    ↪ ready             */
164 #define PZ_SC_TASK_ALERT        12  /* Sensor controller task alert
    ↪                 */
165 #define PZ_ADC_UPDATE           13  /* My defined message to test
    ↪                  */
```

...

```
481 // Sensor Controller functions
482 static void scCtrlReadyCallback(void);
483 static void scTaskAlertCallback(void);
484 static void processTaskAlert(void);
485
486 static void myProcessTask(void);
487
488 void adcBufCallback(ADCBuf_Handle handle, ADCBuf_Conversion *
    ↪ conversion,
489     void *completedADCBuffer, uint32_t completedChannel,
        ↪ int_fast16_t status);
```

...

```
541 // Service callback function implementation
542 // CipherService callback handler. The type cipherServiceCBs_t is
    ↪ defined in cipherService.h
543 static cipherServiceCBs_t user_cipherServiceCBs =
544 {
545   .pfnChangeCb = user_cipherService_ValueChangeCB, //
      ↪ Characteristic value change callback handler
546   .pfnCfgChangeCb = NULL, // No CCCD change handler implemented
547 };
548
549 /*****************************
550  * SENSOR CONTROLLER FUNCTIONS
551  */
552
553 /*****************************
554 */
555
556 static void scCtrlReadyCallback(void)
557 {
558   // Notify application 'Control READY' is active
559     ProjectZero_enqueueMsg(PZ_SC_CTRL_READY, 0);
560 } // scCtrlReadyCallback
561
562 static void scTaskAlertCallback(void)
563 {
564   // Notify application 'Task ALERT' is active
565     ProjectZero_enqueueMsg(PZ_SC_TASK_ALERT, 0);
566 } // scTaskAlertCallback
567
568 static void processTaskAlert(void)
569 {
570   // Clear the ALERT interrupt source
571   scifClearAlertIntSource();
572
573   // Data Processing Here
```

```
574
575
576   // Acknowledge the ALERT event
577   scifAckAlertEvents();
578 } // processTaskAlert
579
580
581 void adcBufCallback(ADCBuf_Handle handle, ADCBuf_Conversion *
      ↪ conversion,
582     void *completedADCBuffer, uint32_t completedChannel,
          ↪ int_fast16_t status)
583 {
584     uint16_t        i, n;
585
586     if (completedADCBuffer == conversion->sampleBuffer){
587         for (i = 0; i < ADCSAMPLESIZE; i++) {
588             updateArray[n]  = sampleBufferOne[i] >> 8;
589             n++;
590             updateArray[n] = sampleBufferOne[i] & 0xff;
591             n++;
592         }
593     } else if (completedADCBuffer == conversion->sampleBufferTwo){
594         for (i = 0; i < ADCSAMPLESIZE; i++) {
595             updateArray[n]  = sampleBufferTwo[i] >> 8;
596             n++;
597             updateArray[n] = sampleBufferTwo[i] & 0xff;
598             n++;
599         }
600     }
601
602
603     ProjectZero_enqueueMsg(PZ_ADC_UPDATE,0);
604 }
605
606 static void myProcessTask(void){
```

```
607     CipherService_SetParameter(CIPHERSERVICE_CIPHERVALUE_ID, 2*
            ↪ ADCSAMPLESIZE, (void *)updateArray);
608 }
```

...

```
736 CipherService_AddService(selfEntity);
```

...

```
772 CipherService_RegisterAppCBs(&user_cipherServiceCBs);
```

...

```
790     // Initalization of characteristics in cipherService that are
            ↪ readable.
791     uint8_t cipherService_cipherValue_initVal[
            ↪ CIPHERSERVICE_CIPHERVALUE_LEN] = {0};
792     CipherService_SetParameter(CIPHERSERVICE_CIPHERVALUE_ID,
            ↪ CIPHERSERVICE_CIPHERVALUE_LEN,
            ↪ cipherService_cipherValue_initVal);
793     uint8_t cipherService_streamEN_initVal[
            ↪ CIPHERSERVICE_STREAMEN_LEN] = {0};
794     CipherService_SetParameter(CIPHERSERVICE_STREAMEN_ID,
            ↪ CIPHERSERVICE_STREAMEN_LEN,
            ↪ cipherService_streamEN_initVal);
```

...

```
841 static void ProjectZero_taskFxn(UArg a0, UArg a1)
842 {
843     ADCBuf_init();
844     GPIO_init();
845
846     // Initialize application
847     ProjectZero_init();
848
849     GPIO_setConfig(CONFIG_GPIO_RLED, GPIO_CFG_OUT_STD |
            ↪ GPIO_CFG_OUT_LOW);
850
```

```
851     ADCBuf_Params adcBufParams;
852
853     /* Set up an ADCBuf peripheral in
            ↪ ADCBuf_RECURRENCE_MODE_CONTINUOUS */
854     ADCBuf_Params_init(&adcBufParams);
855
856     adcBufParams.callbackFxn = adcBufCallback;
857     adcBufParams.recurrenceMode = ADCBuf_RECURRENCE_MODE_CONTINUOUS
            ↪ ;
858     adcBufParams.returnMode = ADCBuf_RETURN_MODE_CALLBACK;
859     adcBufParams.samplingFrequency = 1000;
860     adcBuf = ADCBuf_open(CONFIG_ADCBUF_0, &adcBufParams);
861
862     /* Configure the conversion struct */
863     continuousConversion.arg = NULL;
864     continuousConversion.adcChannel = CONFIG_ADCBUF_0_CHANNEL_0;
865     continuousConversion.sampleBuffer = sampleBufferOne;
866     continuousConversion.sampleBufferTwo = sampleBufferTwo;
867     continuousConversion.samplesRequestedCount = ADCSAMPLESIZE;
868
869     if (adcBuf == NULL){
870         /* ADCBuf failed to open. */
871         GPIO_write(CONFIG_GPIO_RLED, CONFIG_GPIO_LED_ON);
872         while(1);
873     }
874
875 //    // Initialize the Sensor Controller
876 //    scifOsalInit();
877 //    scifOsalRegisterCtrlReadyCallback(scCtrlReadyCallback);
878 //    scifOsalRegisterTaskAlertCallback(scTaskAlertCallback);
879 //    scifInit(&scifDriverSetup);
880 //
881 //    // Set the Sensor Controller task tick interval to 0.1 second
882 //    uint32_t rtc_Hz = 10000;
883 //    scifStartRtcTicksNow(0x00010000 / rtc_Hz);
884 //
```

```
885 //
886 //    // Start Sensor Controller task
887 //    scifStartTasksNbl(BV(SCIF_ANALOG_LIGHT_SENSOR_TASK_ID));
```

...

```
1011     ADCBuf_close(adcBuf);
```

...

```
1182         case CIPHERSERVICE_SERV_UUID:
1183             user_cipherService_ValueChangeHandler(pCharData);
1184             break;
```

...

```
1271       case PZ_SC_TASK_ALERT:
1272         processTaskAlert();
1273         break;
1274       case PZ_ADC_UPDATE:
1275           myProcessTask();
1276           break;
```

...

```
2664 static void user_cipherService_ValueChangeCB(uint16_t connHandle,
2665                                uint8_t paramID, uint16_t len,
2666                                uint8_t *pValue)
2667 {
2668     pzCharacteristicData_t *pValChange =
2669         ICall_malloc(sizeof(pzCharacteristicData_t) + len);
2670
2671     if(pValChange != NULL)
2672     {
2673         pValChange->svcUUID = CIPHERSERVICE_SERV_UUID;
2674         pValChange->paramID = paramID;
2675         memcpy(pValChange->data, pValue, len);
2676         pValChange->dataLen = len;
2677
2678         ProjectZero_enqueueMsg(PZ_SERVICE_WRITE_EVT, pValChange);
```

```
2679      }
2680  }
2681
2682  void user_cipherService_ValueChangeHandler(pzCharacteristicData_t *
          ↪ pData)
2683  {
2684    switch (pData->paramID)
2685    {
2686          case CIPHERSERVICE_STREAMEN_ID:
2687          // Do something useful with pData->data here
2688          // -------------------------
2689          if (pData->data[0] == 1){
2690              ADCBuf_convert(adcBuf, &continuousConversion, 1);
2691          } else if (pData->data[0] == 2) {
2692              GATT_ExchangeMTU(0, 210, 0);
2693          } else {
2694              ADCBuf_convertCancel(adcBuf);
2695          }
2696
2697          break;
2698        }
2699
2700  }
```

A.4 host_test_appc.c

```
57  #include <ti/display/Display.h>
58  #include <ti/drivers/SPI.h>
59  #include <ti/drivers/GPIO.h>
```

...

```
438  static void HostTestApp_taskFxn(UArg a0, UArg a1)
439  {
440
441    SPI_Handle      masterSpi;
```

```
442   SPI_Params      spiParams;
443
444   GPIO_init();
445   SPI_init();
446
447   // Initialize application
448   HostTestApp_init();
449
450   GPIO_setConfig(CONFIG_GPIO_SS, GPIO_CFG_OUTPUT | GPIO_CFG_OUT_LOW
         ↪ );
451   GPIO_setConfig(CONFIG_GPIO_CLR, GPIO_CFG_OUTPUT |
         ↪ GPIO_CFG_OUT_LOW);
452
453   GPIO_write(CONFIG_GPIO_CLR, 1);
454   GPIO_write(CONFIG_GPIO_SS, 1);
455
456   /* Open SPI as master (default) */
457   SPI_Params_init(&spiParams);
458   spiParams.frameFormat = SPI_POL0_PHA1;
459   spiParams.bitRate = 12000000;
460   spiParams.dataSize = 16;
461   masterSpi = SPI_open(CONFIG_SPI_MASTER, &spiParams);
462   if (masterSpi == NULL) {
463       GPIO_write(CONFIG_GPIO_LED_0,1);
464       GPIO_write(CONFIG_GPIO_LED_1,0);
465  //     while (1);
466   }
467   else {
468       GPIO_write(CONFIG_GPIO_LED_0,0);
469       GPIO_write(CONFIG_GPIO_LED_1,1);
470   }
```

...

```
499             // Message
500             dealloc = HostTestApp_processStackMsg((hciPacket_t *)
                   ↪ pMsg, &masterSpi);
```

...

```
539   SPI_close(masterSpi);
```

...

```
610     return(HCI_TL_processStructuredEvent((ICall_Hdr *)pBuf,
          ↪ spiHandle));
```

A.5 icall_hci_tl.c

```
52  #include <string.h>
53  #include <ti/drivers/SPI.h>
54  #include <ti/drivers/GPIO.h>
```

...

```
3910  uint8_t HCI_TL_processStructuredEvent(ICall_Hdr *pEvt, SPI_Handle *
        ↪ spiHandle)
3911  {
3912    return(processEvents(pEvt,spiHandle));
3913  }
```

...

```
6491  static uint8_t processEvents(ICall_Hdr *pMsg, SPI_Handle *spiHandle
        ↪ )
```

...

```
6520      case GATT_MSG_EVENT:
6521        pBuf = processEventsGATT((gattMsgEvent_t *)pMsg, out_msg, (
              ↪ uint8_t *)&msgLen, &allocated, spiHandle);
```

...

```
6534    if (msgLen)
6535    {
6536      if (!(pMsg->event == GATT_MSG_EVENT && Msg->method ==
            ↪ ATT_HANDLE_VALUE_NOTI)){
6537         HCI_TL_SendVSEvent(pBuf, msgLen);
```

```
6538      }
6539  //    HCI_TL_SendVSEvent(pBuf, msgLen);
6540      //HCI_SendControllerToHostEvent(HCI_VE_EVENT_CODE,  msgLen,
              ↪ pBuf);
6541    }
```

...

```
7412  static uint8_t *processEventsGATT(gattMsgEvent_t *pPkt, uint8_t *
          ↪ pOutMsg,
7413                                    uint8_t *pMsgLen, uint8_t *
                                             ↪ pAllocated, SPI_Handle *
                                             ↪ spiHandle)
7414  {
7415    uint8_t msgLen = 0, attHdrLen = 0, hdrLen = HCI_EXT_HDR_LEN + 1;
           ↪ // hdr + event length
7416    uint8_t *pBuf, *pPayload = NULL;
7417    uint8_t status = pPkt->hdr.status;
7418
7419    uint16_t        masterTxBuffer[1];
7420    SPI_Transaction transaction;
```

...

```
7499        case ATT_READ_RSP:
7500          {
7501            attReadRsp_t *pRsp = &pPkt->msg.readRsp;
7502
7503            msgLen = ATT_BuildReadRsp(&pOutMsg[hdrLen], (uint8_t *)
                    ↪ pRsp);
7504            pPayload = pRsp->pValue;
7505
7506            uint16_t i;
7507                for (i=0; i<msgLen; i=i+2){
7508                    masterTxBuffer[0] = ((uint16_t)pPayload[i] << 8)
                           ↪ | pPayload[i+1];
7509                    transaction.count = 1;
7510                    transaction.rxBuf = (void *) NULL;
```

```
7511                    transaction.txBuf = (void *) masterTxBuffer;
7512
7513                    GPIO_write(CONFIG_GPIO_SS, 0);
7514                    SPI_transfer(*spiHandle, &transaction);
7515                    GPIO_write(CONFIG_GPIO_SS, 1);
7516                }
7517          }
```

...

```
7524        case ATT_READ_BLOB_RSP:
7525          {
7526            attReadBlobRsp_t *pRsp = &pPkt->msg.readBlobRsp;
7527
7528            msgLen = ATT_BuildReadBlobRsp(&pOutMsg[hdrLen], (uint8_t
                  ↪ *)pRsp);
7529            pPayload = pRsp->pValue;
7530
7531            uint16_t i;
7532
7533            for (i=0; i<msgLen; i=i+2){
7534                masterTxBuffer[0] = ((uint16_t)pPayload[i] << 8) |
                      ↪ pPayload[i+1];
7535                transaction.count = 1;
7536                transaction.rxBuf = (void *) NULL;
7537                transaction.txBuf = (void *) masterTxBuffer;
7538
7539                GPIO_write(CONFIG_GPIO_SS, 0);
7540                SPI_transfer(*spiHandle, &transaction);
7541                GPIO_write(CONFIG_GPIO_SS, 1);
7542            }
7543          }
```

...

```
7626        case ATT_HANDLE_VALUE_NOTI:
7627          {
7628            attHandleValueInd_t *pInd = &pPkt->msg.handleValueInd;
```

```
7629
7630            attHdrLen = ATT_HANDLE_VALUE_IND_FIXED_SIZE;
7631
7632            // Copy request header over
7633            msgLen = ATT_BuildHandleValueInd(&pOutMsg[hdrLen], (
                   ↪ uint8_t *)pInd) - attHdrLen;
7634            pPayload = pInd->pValue;
7635
7636            uint16_t i;
7637                for (i=0; i<msgLen; i=i+2){
7638                    masterTxBuffer[0] = ((uint16_t)pPayload[i] << 8)
                           ↪ | pPayload[i+1];
7639                    transaction.count = 1;
7640                    transaction.rxBuf = (void *) NULL;
7641                    transaction.txBuf = (void *) masterTxBuffer;
7642
7643                    GPIO_write(CONFIG_GPIO_SS, 0);
7644                    SPI_transfer(*spiHandle, &transaction);
7645                    GPIO_write(CONFIG_GPIO_SS, 1);
7646                }
7647        }
```

B Low Noise Amplifier Layout and Testing

There are three circuit implementations within the chip: one with the output matching network (circuit 1), one with only transistors and capacitors (circuit 2), and one with only transistors (circuit 3). Fig. 1 shows the pin-out of the chip and Table 3 summarizes the corresponding connection to each numbered pad.

B.1 Pin-Out

Table 3: Pad Connections

Circuit	Pad Number	Connection
Circuit 1	1	VDD
	2	RF Input
	3	RF Output
	4	Source of Input NMOS
	5	GND
Circuit 2	1	VDD
	6	RF Output
	7	Capacitor for ISM OMN
	8	Source of Input NMOS
	9	RF Input
	10	Capacitor for MICS OMN
Circuit 3	1	VDD
	11	Gate of Input NMOS
	12	Source of Input NMOS
	13	Drain of Second NMOS

In circuit 2, the output matching network inductor for the MICS band should be connected between pad 1 and pad 10. The output matching network inductor for the ISM band should be connected between pad 7 and pad 10.

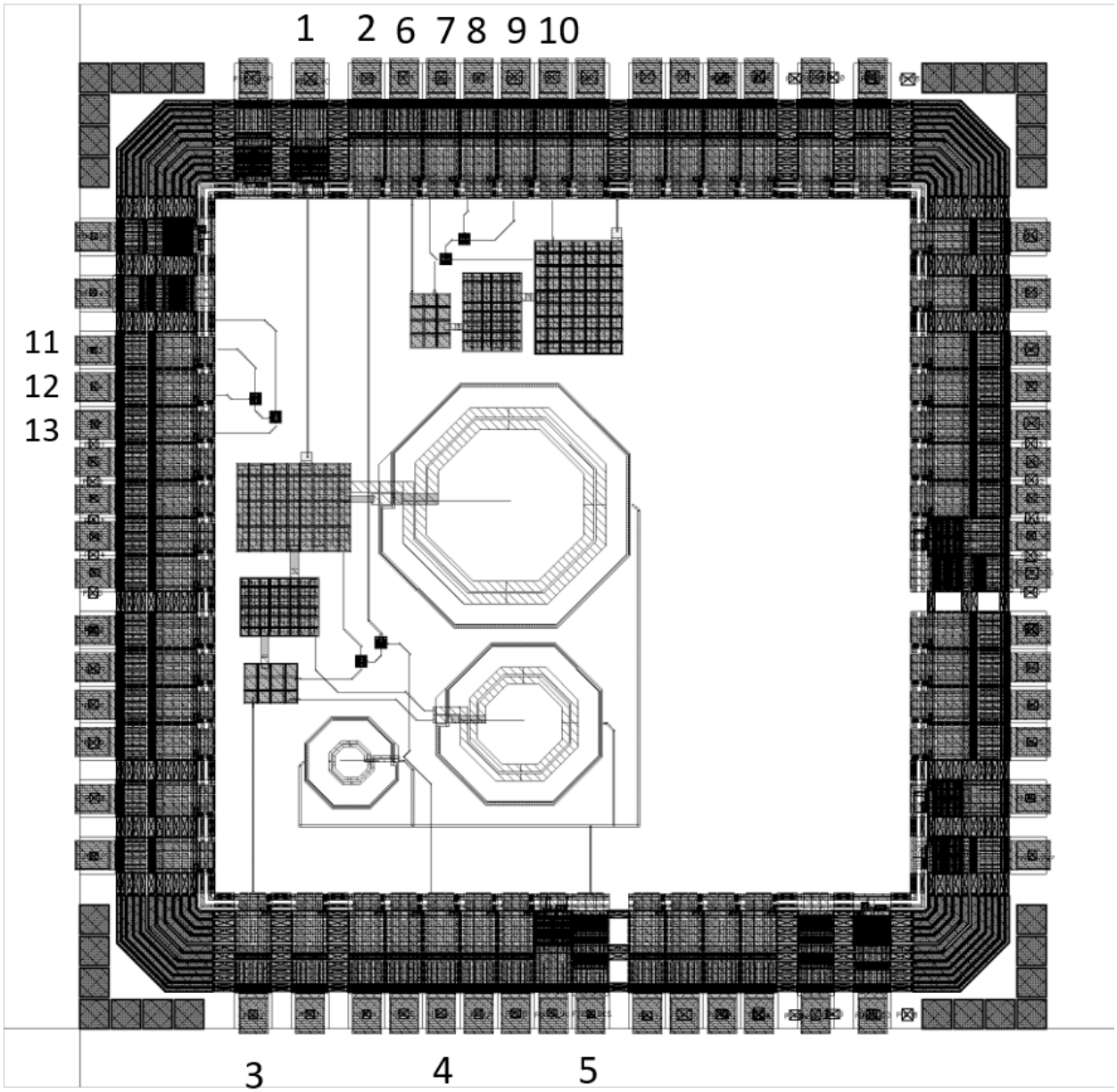


Figure 1: Pin-out of the LNA chip.

B.2 A Note for Future RFIC Chips

Wire Bonding: If you want to wire bond the IC directly to the PCB, make sure the PCB has an ENIG or ENIPIG finish.

Probe Station: The RF probes at the probe station have 3 connections: ground, signal, ground. The connections are spaced 100 μm apart. Pads with 60 μm width and 40 μm apart would work well for this. There is a limit to two RF probes.

Vita

Kendra Anderson received her B.S. degree in electrical engineering from the University of Tennessee in 2019. Later in 2019, she joined the Mlab group at the University of Tennessee as a research and teaching assistant pursuing her M.S. degree in electrical engineering. Her current research interests include analog integrated circuit design, radio frequency integrated circuit design (RFIC), RF/microwave circuits and systems, microelectronics, and wireless body area networks.