BS4LIES: Backscatter 4 Low-power IoT Environmental Sensing

Jason Cox, Rahul Bulusu, Eric Greenlee, and Aadesh Madnaik

CS 8803: MCI Fall 2023; Final Project Report

Introduction

Many environmental sensing applications require deploying a dense network of sensors to achieve a high-resolution understanding of the environment. One such application is the study of urban heat islands -- the high variation of temperature between nearby points within an urban area -- which is especially relevant in Atlanta (Zhou and Shepherd, 2010). Deploying such high-density sensor networks is difficult because of the physical and human infrastructure needed to maintain these systems: conventionally, each sensor requires either a connection to mains power or a replaceable battery.

Backscatter presents an opportunity to solve the power problem by enabling wireless communication with the sensors while keeping their power consumption incredibly low, on the order of tens of microwatts, allowing sensors to be powered by environmental energy harvesting. However, current backscatter systems tend to prioritize throughput over range, achieving distances of only a few meters between the tag and the transmitter/receiver when configured in a monostatic setup where the transmitted and receiver are co-located (Xu et al. , 2018). Such short ranges do not significantly reduce the infrastructure requirements of a practical system.

In this project, we aim to build an ultra-low power backscatter device for monitoring Atlanta's urban heat islands, using digital communications techniques to extend the range between the tag and transmitter/receiver by sacrificing throughput. Specifically, we build on the work by Arora et al.'s analog, 915MHz backscatter tag (2021) by modifying the system to communicate temperature readings as digital data. We explore digital range-increasing techniques such as forward error correction (FEC) and investigate low-power digitization and signal generation. We build a system consisting of a co-located transmitter and receiver capable of reading temperature data from a single backscatter tag. We achieve a range of 7.5+ meters between the transmitter/receiver and a battery-powered tag and also design a tag that could theoretically run on less than 100 uW of power. We leave the actual implementation of this low-power tag, the energy harvesting system, and system scalability to future work.

Problem Statement

The goal of this project is to design and implement a backscatter device in the 915 MHz band that measures ambient temperature at least every 5 minutes, achieves at least 10 meters of range, and consumes less than 100uW of power.

Design

In this section, we discuss the high-level design of our backscatter device. To achieve the goal laid out in our problem statement, we designed a simple packet to communicate the temperature that requires only 56 bits: 16 for the preamble and 40 for the payload. The payload consists of the 6 bits of temperature data, corresponding from 32 to 95 degrees Fahrenheit, which have an added 2 bit XOR checksum and then are repetition encoded 5 times. The entire 8 bits of data and checksum are repeated, rather than repeating each bit individually, to account for the fact that errors tend to occur in bursts. The details of a packet are shown in Figure 1.

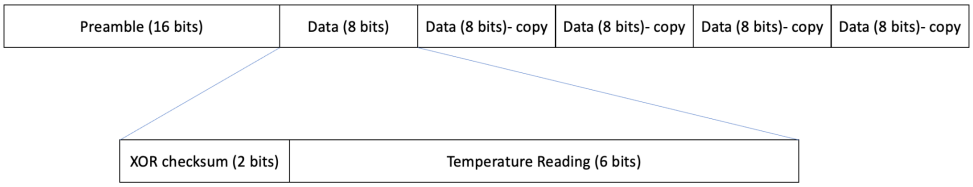

Figure 1: Packet Contents

To communicate this packet over backscatter, we use three components: a carrier wave transmitter, a tag that modulates the data onto the carrier, and a receiver that demodulates the packet and displays the temperature. The 915MHz carrier wave is generated by a Software Defined Radio (SDR). The tag, shown in Figure 2, consists of (1) an Arduino that reads the temperature from an I2C device and then encodes this into the packet using Binary Phase Shift Keying (BPSK) and (2) a custom Printed Circuit Board (PCB) that couples the Arduino's signal onto the carrier wave. The receiver uses a separate SDR to digitize the signal and GNURadio to filter, demodulate, and decode the packet. GNURadio also served as a useful debugging tool. Further details on these components are explained in the "Implementation" section.

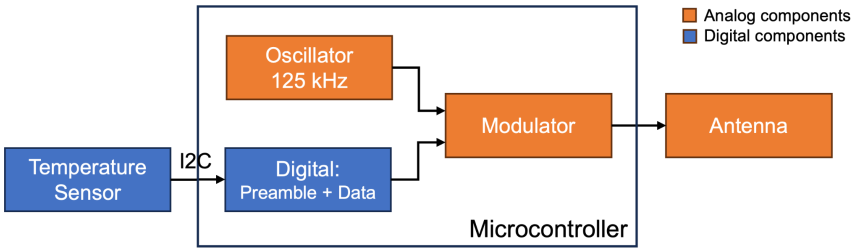

Figure 2: Block Diagram (Tag)

Over the course of the project, we made several design decisions to simplify the prototype. Rather than implementing a 100uW solution, which would have unreasonably expanded the time, cost, and complexity required for our prototype because we would not have been able to use an Arduino, we focused on ensuring our design could directly translate into a 100uW solution, shown in the "Results" section. Additionally, we implemented simple encoding rather than more sophisticated methods that are more robust to noise. Finally, due to hardware constraints, we transmit data packets over 5 ms instead of using the full 5 minute interval between measurements laid out in our problem statement. These changes mean that we do not achieve the range or power from our problem statement, but this prototype and the associated analysis provide a proof-of-concept and a path forward for doing so.

Implementation

Our hardware consists entirely of components that we already had access to or that we could easily create with capabilities at Georgia Tech. As pictured in Figure 3, our system consists of a carrier transmitter, composed of a HackRF One SDR and a RFMAX RFID antenna, a tag, composed of a BME280 temperature sensor, SparkFun Thing Plus ESP32 Arduino, a custom backscatter tag, and a Time-7 A7030c-71584 Slimline RFID antenna, and a receiver, composed of a N210 USRP, a RFMAX RFID antenna, and a Lenovo ThinkPad running GNURadio. The tag is powered by a 5V Anker power bank.

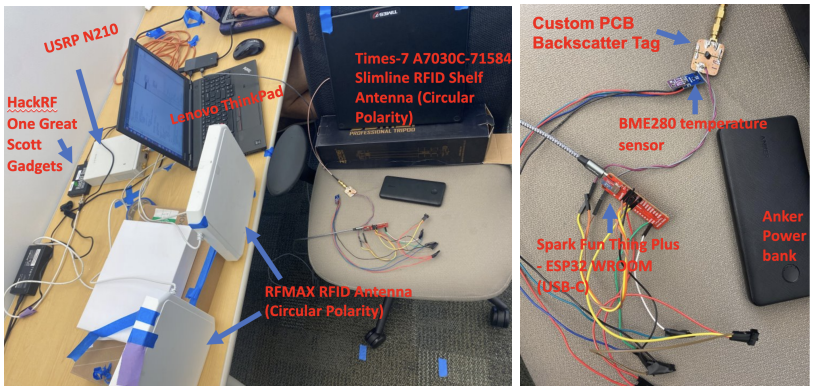

Figure 3: Hardware setup

The carrier wave transmitter simply uses a command line interface to transmit at a specified frequency and power level.

The tag consists primarily of two components: (1) an Arduino running custom code to take a reading from the temperature sensor and then convert it into a BPSK signal via the Digital to Analog Converter (DAC) and (2) a custom PCB to mix the BPSK signal onto the 915MHz carrier wave. While the Arduino, which we ran off a 5V USB power bank, required no hardware modifications besides soldering wires to the temperature sensor and tag, the embedded code it runs, attached as “TempSensoTag_Final.ino”, incorporates several complex components. Substantial code is dedicated to enabling and configuring the phase-shiftable sinusoid, which has no Arduino library and thus was implemented by altering registers in the microcontroller’s memory. Another custom function, “with_checksum()”, adds the 2 bit XOR checksum to the integer temperature reading. The main loop continuously reads the I2C temperature value, encodes it into a packet, and then transmits it one bit at a time, with a non-inverted sinusoid output for a “1” and an inverted output for a “0”. To implement the 40us symbol period, the loop includes a 39 microsecond delay at the end since the rest of the loop takes around 1us to run. Debugging the code proved challenging because adding print statements drastically increased the amount of time for each loop.

The PCB, whose schematic and image are shown in Figure 4, is a very simple circuit consisting of a JFET transistor, two DC-blocking capacitors, and two connectors. The Arduino BPSK output connects to J2, while the antenna that both receives the carrier wave and retransmits the modulated backscatter signal is connected to J1. The JFET, Q1, mixes these two signals together at its port 1, making backscatter possible. We designed the PCB in KiCAD, included in the attached files “DigBackscatter_MCI_Arduino_rev0.kicad_sch” and “DigBackscatter_MCI_Arduino_rev0.kicad_pcb”, and we manufactured the PCB in Georgia Tech’s HIVE Makerspace.

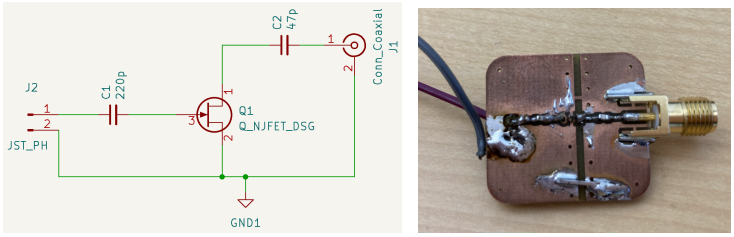

Figure 4: PCB schematic and image

The receiver is an SDR that uses GNU Radio to decode the received signal. The basic outline of the GNU Radio flow is shown in Figure 5 below, and the full GNU Radio file, including source code for the custom blocks, is attached as “receiver-decode-final.grc”.

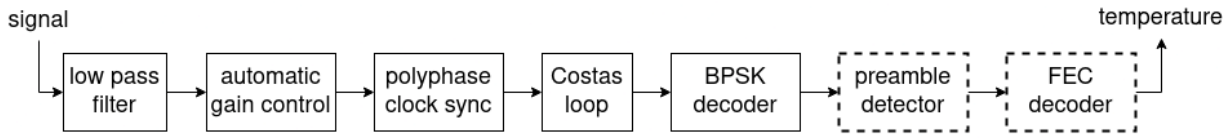

Figure 5: GNU Radio receiver flow. Custom blocks are shown with dashed borders.

The signal first passes through a low pass filter to isolate the data signal from the carrier wave. We then use automatic gain control to increase the amplitude of the signal. The polyphase clock sync and Costas loop blocks synchronize the receiver’s clock with the tag and recover the phase-modulated carrier frequency. The BPSK decoder transforms the phase-modulated signal into binary data.

Once the signal has been converted to binary data, our custom preamble detector waits to receive the preamble bits and then passes the following 40 bits onto the FEC decoder. We found that sometimes the previous blocks swapped 1s and 0s, so the preamble detector also watches for the inverse of the preamble and inverts the following 40 bits as needed.

Finally, the custom FEC decoder decodes our repetition encoding scheme. It checks each repeated bit value and chooses the value that occurs most frequently. It then verifies the checksum, discarding corrupted packets, and outputs the temperature value from the packet.

The outputs of the full demodulation path are shown in Figure 6. The left image displays the final decoded and adjusted temperature in degrees fahrenheit (“decoded with threshold”: 71), the input to the decoder (“coded”: 39), and the spectrum after it has been filtered and adjusted with automatic gain control. The middle image shows the constellation, which has clean clusters around -1+0j and 1+0j as would be expected for BPSK, from the output of the Costas loop. The right image also shows the output from the Costas loop but instead conveys data in the time domain, whereby the repeated pattern of packets can be clearly seen.

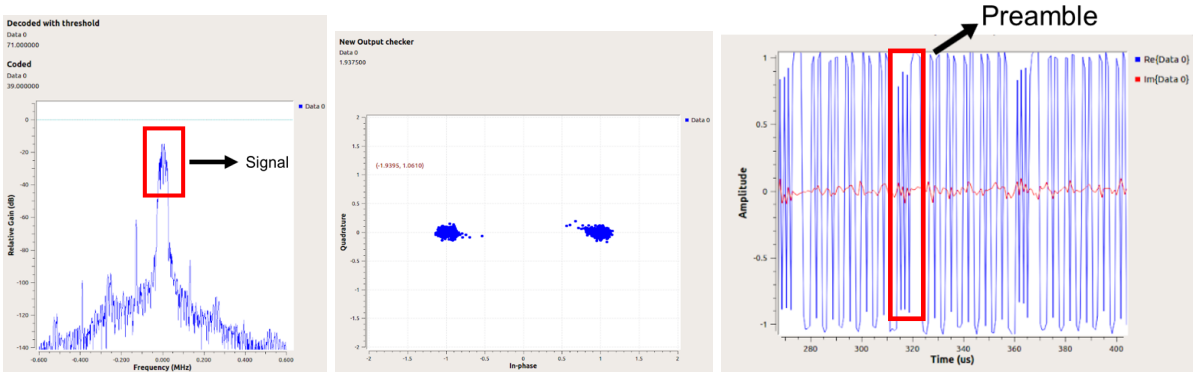

Figure 6: GNURadio demodulating a packet

Results

A recorded demonstration of the full system reporting temperature from 7.65 meters away is available at this link.

Range, Throughput, and Temperature Measurement

Each BPSK symbol takes 40us to transmit, resulting in a throughput of 1/40us = 25 kbps. Attempts to reduce this throughput to achieve greater ranges presented challenges as GNURadio did not support the sharper digital filters that would be required to downsample and demodulate the signal. Exchanging throughput for range will be a focus of future work.

With our current implementation transmitting the carrier wave at -16 dBW (14dBm), we achieve a range of greater than 7.65 meters with line of sight. This is less than our goal of 10 meters but is close enough to count as a preliminary success. Additionally, this range is highly sensitive to multipath and fading. Preliminary bit error rate (BER) measurements showed that moving from 1.03 meters to 1.20 meters increased the BER from 0.00031% to 25%, while further increasing the range caused the BER to go back down to negligible levels. We observed that the system was very sensitive to tag location. Moving the tag a centimeter further from the carrier and receiver would unpredictably sometimes result in a total loss of signal while other times would significantly improve the signal strength and result in a much cleaner constellation. Selecting tag placement and making the system robust to multipath and fading will also be a focus of future work.

Our system communicates temperature with less than three seconds of latency. We tested this by putting a finger in contact with the temperature sensor and observing how long it took the GNURadio display to start increasing. This accounts for the latency across the full chain, including the temperature sensor, the Arduino, the tag, and the demodulator. This delay is well within the range of permissible values for measuring ambient environmental temperature.

Link Budget

From Griffin and Durgin, it is theorized that the received signal strength of a backscatter system is given by the equation $P_R = \frac{P_T G_T G_R G_t^2 \lambda^4 X^2 M}{(4\pi r)^4 \Theta^2 B^2 F_\alpha}$. We describe what each of these terms is and assign it a value in Table 1 in both Raw and decibel values for two difference distances, noted as “reader-to-tag separation”: a 1 meter reference and a 7.65 meter maximum distance with demodulation. For each, we compute the theoretical received power from the above equation. We also measured the received power for the 1 meter reference (the signal was too unsteady to get an accurate measurement for the 7.65 meter setup), which was substantially lower than the value predicted by the equation. This is most likely due to a non-ideal free space loss factor, which is typically modeled as 2 but has been documented to be up to 4 even with line of sight. We adjusted this value until the theory and our measurement agreed, landing on a value of 3.3. This means that every time read-to-tag separation doubles, the signal strength decreases by a factor of $2^{(2*3.3)} = 97$ rather than 16. Adjusting the 7.65 meter distance, this suggests we received and demodulated a -152 dBW (-122 dBm) signal. It is also likely that multipath played a large role in the signal strength since our experiments showed that moving the tag further away from the transmitter and receiver would sometimes cause the signal strength to improve.

	PT	GT	GR	Gt	λ	X	M	r	Θ	B	Fα	PR: Received Power (W)		
Description	Transmitted power (W)	Transmitter antenna gain	Receiver antenna gain	Tag antenna gain	Wavelength (m)	Antenna polarization mismatch	Modulation factor	Reader-to-tag separation (m)	Tag's on-object gain penalty	Path-blockage loss	Fade margin for 1E-3 Error rate with 6 dB SNR	Theoretical	Measured	Theoretical: Adjusted for loss factor of 3.3
Raw value	0.03	3.98	3.98	3.98	0.33	1.00	1.00	1.00	1.23	1.00	6.31	3.07E-07	3.98E-10	4.25E-10
dB value	-16 dBW	6 dBi	6 dBi	6 dBi		0	0		0.9	0	8 dB	-65.13	-94	-93.71
Raw value	0.03	3.98	3.98	3.98	0.33	1.00	1.00	7.65	1.23	1.00	6.31	8.95E-11	n/a	6.26E-16
dB value	-16 dBW	6 dBi	6 dBi	6 dBi		0	0		0.9	0	8 dB	-100.48	n/a	-152.03

Table 1: Backscatter link budget and path loss analysis

Power Budget Calculations

As seen in Figure 2 our tag hardware has five major components: sensor, antenna, oscillator, digital logic circuits, and the modulator. We know that our current tag, which is battery powered and uses an Arduino, exceeds our power goal of 100uW since we did not have the time to build and test all of the custom hardware for a digital system. Instead, we designed a digital tag with discrete components that is functionally equivalent to our prototype tag.

The following measurements are based on measured and datasheet values for the digital tag designed with discrete components. The digital logic circuit which controls the preamble, data, and coding consumes about 5 μW of power. The modulator is made using low-power RF switches, and consumes about 5 μW of power too. The sensing hardware consumes less than 20 μW of power. However, since the temperature sensor is a commodity device, we do not have control over the power. Lastly, the first iteration of the 125 kHz Clapp oscillator is based on BJT, so the power consumption was 500 μW which significantly overshoots the budget. By updating the oscillator circuit by simply changing the transistor to a MOSFET, we expect the power consumption to drop below 500 nW, which is a 1000x reduction. This design was first implemented in Arora et al. All in all, we have achieved a reasonable power consumption of 30 μW for the tag. The power consumption is well within the designated budget of 100 μW. It is possible to further reduce the average power consumption to a few μWs by making the data transmissions intermittent, since the reported power consumption is for continuous transmission.

Future Work

Although our backscatter system is able to achieve greater than 7.65 meters with line of sight this is still not really practical for outdoor applications. In the future, we will explore further extending the range to the 100s+ meters using three primary approaches: (1) we will lower the data rate by narrowing the bandwidth and thus decrease the noise we received, (2) we will increase the output power to the maximum allowable -6 dBW using a power amplifier that we have already selected, and (3) we will explore interleaving the preamble with data bits so that we can correct for changes in the channel over the longer periods of time so we can increase the duration of out transmission. Second, we will reduce the power consumption by implementing the design with discrete components or an ASIC and switching our 125 kHz BJT Oscillator to a MOSFET Oscillator, decreasing the total power consumption to less than 30 uW. Lastly, we also want to improve practical usability of our backscatter system, meaning we want to do two things: use spread spectrum to create coding channels for scaling to many devices and deploying the system in the field to measure how robust it is and how it responds in different physical environments.

References

Arora, Nivedita, et al. "Mars: Nano-power battery-free wireless interfaces for touch, swipe and speech input." *The 34th Annual ACM Symposium on User Interface Software and Technology*. 2021.

Griffin, Joshua D., and Gregory D. Durgin. "Complete link budgets for backscatter-radio and RFID systems." *IEEE Antennas and Propagation Magazine* 51.2 (2009): 11-25.

Van Huynh, Nguyen, et al. "Ambient backscatter communications: A contemporary survey." *IEEE Communications surveys & tutorials* 20.4 (2018): 2889-2922.

Xu, Chenren, Lei Yang, and Pengyu Zhang. "Practical backscatter communication systems for battery-free Internet of Things: A tutorial and survey of recent research." *IEEE Signal Processing Magazine* 35.5 (2018): 16-27.

Zhou, Yan, and J. Marshall Shepherd. "Atlanta's urban heat island under extreme heat conditions and potential mitigation strategies." *Natural hazards* 52 (2010): 639-668.