13.56 MHz RFID Emulator
Final Report

1st Matthew Cox
Department of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
coxm@mit.edu

2nd Nathan Shwatal
Department of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
nshwatal@mit.edu

***Abstract*—We present a design for an FPGA-based 13.56 MHz RFID utility, integrated with a PCB analog front-end. It can emulate an RFID card, successfully interfacing with an actual RFID system. The design allows for selection between a number of different cards. It also includes the digital logic for emulating a card reader, and future work might endow the system with a more capable analog front-end that can support the card reader function.**

***Index Terms*—RFID, FPGA, 13.56MHz, PCD, PICC**

I. Introduction

Radio Frequency Identification (RFID) is a system by which mobile items (such as ID cards, microchips in pets, and containers transporting items on a factory floor) communicate small amounts of data to readers (such as a tap pad or a scanner), where the energy to power the mobile item's circuitry comes exclusively from an electromagnetic field put out by the reader. Previous 6.205 projects have implemented a 125kHz version of RFID on an FPGA. We instead implemented the more modern, 13.56MHz version of RFID (described in the ISO 14443A specification). The ISO 14443A specification provides for more data throughput and modes of communication than 125kHz, and also supports modern systems, such as MIT's current access control system.

We aimed to emulate both sides of the system, to be able to act as a mobile tag and also as a reader. Due to shortcomings of our analog front-end, the physical system is only able to act as a mobile tag, but the digital side implements the logic of both sides.

II. Terminology

- **PCD** (Proximity Coupling Device): the RFID card reader, which reads the data stored on the PICC
- **PICC** (Proximity Integrated Circuit Card): a RFID card or other (usually passive) proximity device that holds data
- **UID**: the Unique ID of a PICC, which can be 4, 7, or 10 bytes in size

III. Summary of RFID Specification: RF Interface

The RF interface for 13.56MHz RFID is defined by ISO specification 14443-2 [1]. There are two versions of the RF interface: A and B. Because MIT's access control systems use type A, we constrain our project to type A: supporting both type A and B increases complexity of the analog front-end but has practically no impact on the complexity of the digital system, so because this is a digital design class, we prefer to make a minimum viable analog front-end which supports only type A.

The ISO specification supports multiple data rates. For purposes of this explanation, we look at the $f_c/128$ data rate (that is, each bit lasts for 128 cycles of the 13.56MHz carrier frequency), which is the slowest data rate in the specification and the one which all devices are required to support. There are modes with increased data rates (such as $f_c/64$, $f_c/32$, and $f_c/16$), but we chose not to implement them because we preferred to avoid the added complexity, and we found no evidence that MIT's access control system supports faster data rates.

Communication from PCD to PICC is done with on-off keying of the 13.56MHz carrier signal. Bits are encoded using a modified Miller encoding:

- a 1 is represented by the signal being off between halfway and 3/4 of the way through the bit, and otherwise on
- a 0 is represented by the signal being steadily on (for a zero preceded by a one) or by the signal being off for the first 1/4 of the bit and otherwise on (for a zero that begins a message, or for a zero preceded by a zero)

A few bits of a typical communication are shown in Figure 1.

Fig. 1. Communication from PCD to PICC: typical RF envelope.

Communication from PICC to PCD is done by modulating the load characteristics of the PICC. The PCD sends out a continuous 13.56MHz carrier, and it detects the small variations in load from the PICC. Bits are encoded using a Manchester encoding: a 0 is represented by modulating the load in the second half of the bit, alternately 8 cycles at low impedance and 8 cycles at high impedance; and a 1 is represented by the same load modulation in the first half of the bit. A few bits of a typical communication are shown in Figure 3.

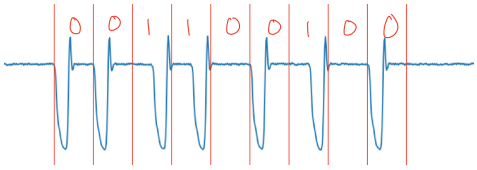

Fig. 2. Communication from PCD to PICC: envelope of measured waveform. (External PCD, sending to our system acting as PICC.)

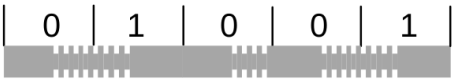

Fig. 3. Communication from PICC to PCD: typical RF envelope.

IV. Summary of RFID Specification: Logic Flow

The logical flow of 13.56MHz RFID communications is defined by ISO specifications 14443-3 [2] and 14443-4 [3]. 14443-3 details the initial handshake between PCD and PICC, and the anticollision protocol to be followed should multiple PICCs be in range of a PCD; and 14443-4 details the method for subsequent exchange of information after the initial handshake. For the purposes of this project, we focus primarily on 14443-3, as the initial handshake involves the transfer of the PICC's UID, often the sole piece of information used for access control. Subsequent data is usually encrypted and used for transit card balances, among other things.

The handshake begins with an initial request from the PCD and an acknowledgment from the PICC that indicates the size of its UID. The PCD then begins the process of requesting the UID, which may take multiple steps even in the case of no collisions among multiple PICCs depending on the size of the UID. Once a complete UID is obtained, the PCD sends a final select signal, which is acknowledged by the PICC. Further data transfer will be initiated if both parties implement the same protocol (either 14443-4 or a proprietary one).

All transmissions are put into frames with specified lengths. Frames are organized in the following ways:

- PCD to PICC
  1) Frame starts with a start bit of 0
  2) Can include either 7 data bits and no parity bit (a "short frame"), or 1 to 9 data bytes, each of which has a parity bit (a "standard frame")
  3) Frame ends with an end bit of 0
- PICC to PCD
  1) Frame starts with a start bit of 1
  2) Includes 1 to 5 data bytes, each of which has a parity bit
  3) No end bit – transmission just ceases after the last parity bit

MIT's access control system only checks the UID of a PICC and doesn't check any further data, so we only implemented 14443-3 and not 14443-4.

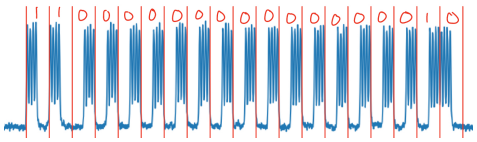

Fig. 4. Communication from PICC to PCD: envelope of measured waveform. (Transmitted by our system.)

V. Emulation Overview (Nathan)

In order to implement PCD and PICC functionality, the system is split into two main components: one for the PCD, and one for the PICC. A high-level block diagram abstracting out these components is presented in figure 5. These two components share the analog front-end. Control of the analog front-end's mode is implemented using a number of control signals. The output from the ADC is decoded and sent to the PCD and PICC emulation components by the ADC decoder unit.

VI. PCD & PICC Emulation (Nathan)

The organization of both main emulation components is quite similar. They each have a control unit implementing a FSM handling their respective side of the handshake protocol, and input and output units. The basic structure of a transaction between the PCD and PICC are indicated in figures 6 and 7, but the actual state machines are significantly more involved, since they have to meet timing specifications and handle errors. Each of REQA, ATQA, etc. is a specific command. The longest command sent from the PCD to the PICC in the initial handshake is 9 bytes, so the size of the connection between the PCD control unit and the PCD output unit is 72 bits. In order to receive such a command, the size of the connection between the PICC input unit and the PICC control unit is 72 bits. The PICC sends at most 5 bytes of data at time, so the size of the connection between the PICC control unit and the PICC output unit is 40 bits. Likewise, the connection between the PCD controller unit and PICC input unit is also 40 bits. These signals are accompanied by single-cycle valid signals, so that the corresponding units know when to start processing the data.

As the PICC only sends 5 bytes of data at a time, for UIDs of 7 or 10 bytes, multiple rounds of communication have to take place, which are called cascades. We have implemented the protocol to start for 4 byte UIDs, and cascading functionality could be a future extension. In order to ensure that errors accumulated in transmission are not incorrectly assumed to be part of a valid message, multiple forms of error detection are used in the protocol. The 5th byte of a PICC transmission is the XOR of the bytes before it, allowing for a single error in each bit to be detected. The PCD command repeating the UID to the PICC appends two bytes of CRC, and so does the command from the PICC to the PCD confirming that this UID is indeed correct. The CRC used is a common variant of CRC_16, which was implemented in hardware using a LFSR.

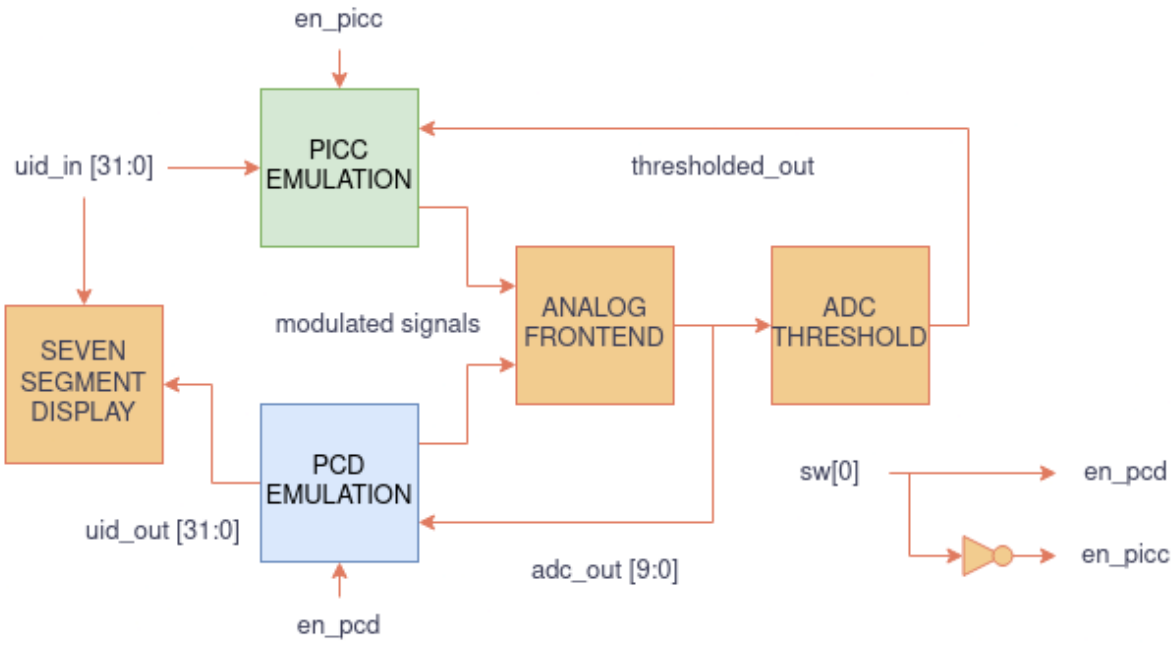

Fig. 5. Simplified block diagram.

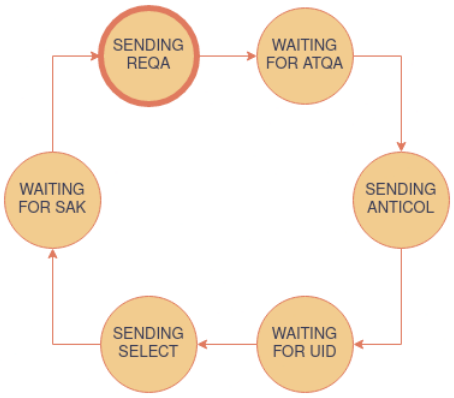

Fig. 6. Base PCD state machine.

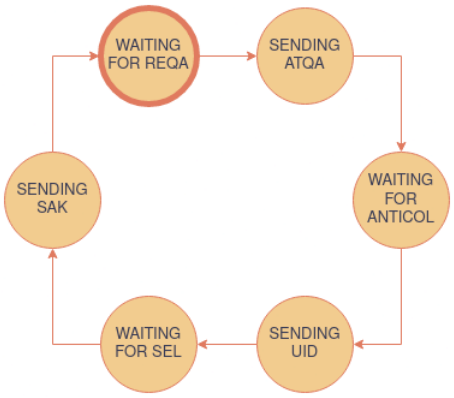

Fig. 7. Base PICC state machine.

VII. Encoding Output Signals (Nathan)

As mentioned in section III, the PICC and PCD transmit data using distinctly different encodings. However, the implementation of both the PCD and PICC output modules involves a very similar state machine (Figure 8), with transitions based on the next bit to transmit, including the odd parity bits. These bits are added in-flight after each byte instead of through pre-processing, since the it only involves a single bit of history. Rather than producing a modulated signal that gets fed directly into the analog frontend, each output module produces a signal that indicates when to modulate the carrier or subcarrier, improving modularity and allowing for a more unified interface. The two modules differ in the modulation pattern produced in each of these states, corresponding to their distinct protocols.

VIII. Decoding Input Signals (Matthew)

The FPGA's only input from the analog front-end is a raw stream of 10-bit samples from the ADC that indicate the current amplitude of the RF envelope. We need to decode this raw input to parse out the incoming bit stream coming in through the RF channel.

A. PCD In

1) Detecting Modulation: The input to the PCD is a sequence of bits, modulated with a subcarrier (whose period is 16 cycles of the 13.56MHz carrier) either in the first or second

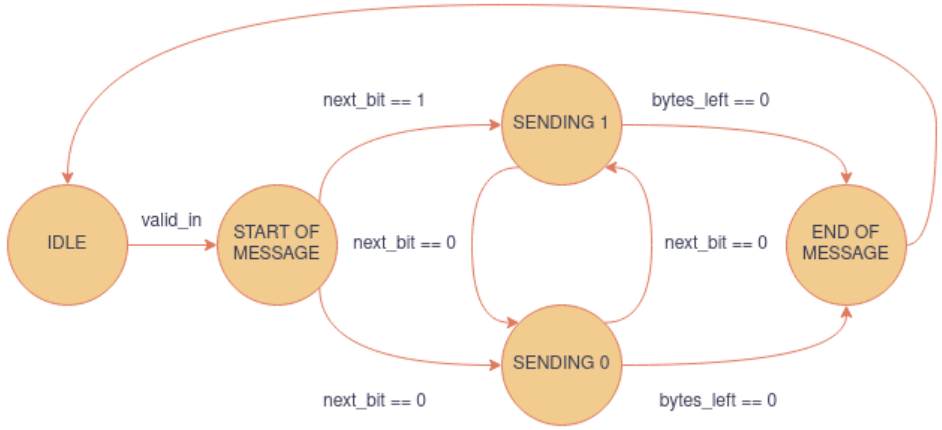

Fig. 8. Basic state machine for picc_out and pcd_out modules.

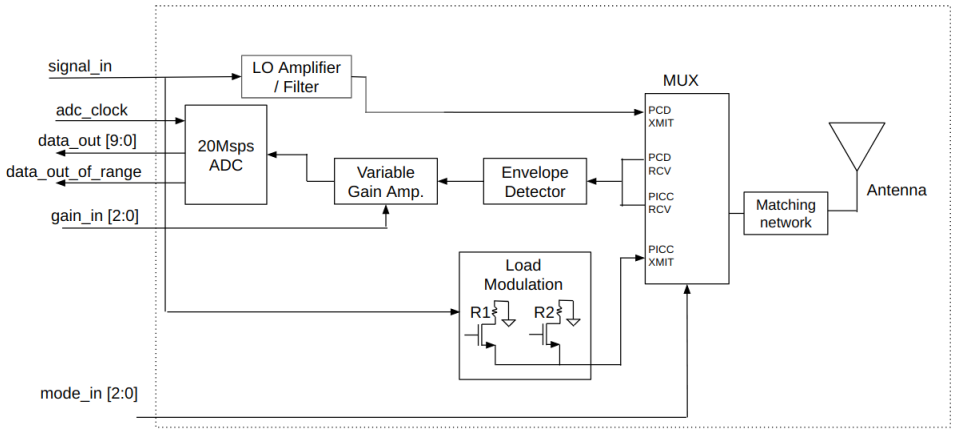

Fig. 9. Block diagram of the analog front-end.

half of the bit (Figure 3). We need to detect the subcarrier's presence or absence.

To do so, we implemented a correlator that compares the input with the subcarrier frequency to see when the subcarrier is present. The correlator takes the envelope samples from the last 32 cycles, take the average of that 32-cycle segment, and threshold each of those 32 samples by whether it's above or below the average (casting it to 1 or -1). Then, we take the dot-product of that segment with a square wave at the subcarrier frequency (the square wave also taking a value of 1 or -1 at each point), then we output the absolute value of that dot product. The correlator output will range from 0 (for an input that's completely uncorrelated with the subcarrier) to 32 (for an input that exactly matches the subcarrier and is in-phase or 180° out of phase).

We determine that the input is currently modulated if, for at least 20 of the last 32 correlator outputs, the output is at least 10.

2) Parsing Bitstream: We can detect transitions from one bit to the next by the spacing of the modulated portions. If the same bit is repeated (1 to 1, or 0 to 0), the time difference in the onset of modulation from one bit to the next will be 128 cycles. On a transition from 1 to 0, that will be 192 cycles. And on a transition from 0 to 1, the modulation from the first bit continues into the next. We detect the time difference between the start of one modulation and the start of the next, and also detect double-length periods of modulation, to compose the input bitstream.

3) Parity Check and Output: Once we haven't detected modulation for significantly longer than the bit spacing, we check the input to validate it has a valid length, and if so, we also validate its parity bits. If everything checks out, the PCD input module processes the input by removing the start bit and parity bits, then passes it along to the PCD controller.

B. PICC In

1) Detecting Modulation: The input to the PICC is a sequence of bits where the carrier is halted for a 32-cycle portion of the bit, or for the first 0 in a sequence of 0s, the carrier continues for the entire bit (Figure 1). To detect when the carrier is halted, we implemented a thresholder. The thresholder has a running average of the ADC value from the last 128 cycles, and as each bit comes in, the thresholder

returns a 0 if the new bit is less than the running average by 20 LSBs or more, or a 1 otherwise.

We determine that the input is currently modulated if, for at least 20 of the last 32 thresholder outputs, the ADC value is below the threshold.

2) Parsing Bitstream: Similarly to the PCD input, we detect transitions from one bit to the next by the spacing from the onset of one modulated section to the onset of the next. If the same bit is repeated (1 to 1, or 0 to 0), the time difference in the onset of modulation from one bit to the next will be 128 cycles. On a transition from 1 to 0 to 0, or from 0 to 1, that will be 192 cycles; and on a transition from 1 to 0 to 1, it will be 256 cycles. We detect the time difference between the start of one modulation and the start of the next (checking whether they match the theoretical values to within a reasonable tolerance) to compose the input bitstream.

3) Parity Check and Output: Once we haven't detected modulation for significantly longer than the bit spacing, we check the input to validate it has a valid length. If the length matches a short frame, then we strip off the start and end bits and pass the data to the PICC controller. If the length matches a standard frame, we validate its parity bits, then strip the start, parity, and end bits, and pass it along to the PICC controller.

IX. Clocking (Nathan)

As we need to manipulate signals at a rate of 13.56 MHz, we have chosen our main clock signal to be 135.6 MHz, 10x this rate. Fortunately, Vivado's Clock Wizard can produce this frequency quite precisely. This gives us 10 cycles to perform any necessary calculations per change in the output state. In order to generate the 13.56 MHz signal, we flip its state every 5 clock cycles, using additional bits to signal rising and falling edges. Additionally, PICC load modulation requires a subcarrier signal at $fc/16$, or 0.8475 MHz; we generate this by flipping its state every 8th cycle of the 13.56MHz signal.

X. Analog Front End (Matthew)

This project requires a significant analog front-end portion to process the 13.56MHz signals. Whereas previous similar projects, operating at 125kHz, could directly sample at the operational frequency, 13.56MHz is too fast for a reasonably-priced ADC to directly sample with sufficient samples per cycle. We can directly produce a 13.56MHz output signal, but not read an incoming 13.56MHz signal. Therefore, we required a more complex analog front-end which could pre-process the 13.56MHz signal, to reduce the sample-rate requirement of the ADC and the digital processing.

We designed a printed circuit board to implement the analog front-end. It is a 4-layer PCB, with ground planes on the inner layers and signals and power routed on the outer layers. Figure 14 is a photo of the final PCB, which includes some modifications off the circuit board that we had to make to correct various design defects.

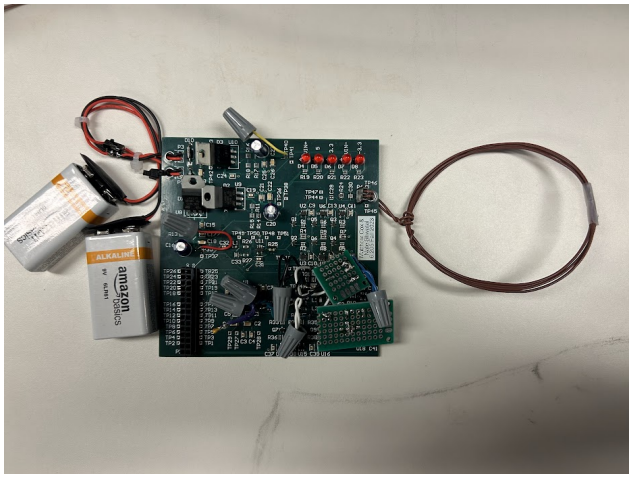

Fig. 10. Photo of Analog Front-End

A. Functional Blocks

Figure 9 is a block diagram of the analog front-end. The diagram shows the following blocks:

- The LO Amplifier / Filter is a block that takes a 13.56MHz square wave on the signal_in pin, filters out the harmonics to get a sine wave, and buffers the output.
- The Envelope Detector block takes a modulated 13.56MHz signal coming in, rectifies it, and averages the rectified version, to produce a lower-frequency signal from which the modulation can be recovered.
- The Variable Gain Amplifier block is an amplifier whose gain is set by the gain_in pins, to provide an avenue for the FPGA to request a higher gain on the ADC's input if needed.
- The ADC is a 10-bit analog-to-digital converter capable of up to 20Msps. It samples on a 13.56MHz clock which comes from the FPGA, and has a pipeline with a latency of 3 cycles.
- The Load Modulation block switches between two different loads, and is used in PICC mode.
- The Antenna is an inductive loop antenna.
- The Matching Network block is an impedance-matching network which cancels the reactive component of the antenna's impedance, and has a dissipative component to reduce its quality factor to around 2 (to prevent excessive ringing, which would make it difficult to parse an input signal which is being turned on and off quickly)
- The Multiplexer (MUX) block switches the antenna between various other blocks, depending on what is appropriate to connect in which mode.

B. Modes

When acting as a PCD in transmit mode, the FPGA directly generates the desired 13.56MHz output signal, modulated with on-off keying as a square wave on the signal_in pin. It's filtered to remove harmonics, and the multiplexer is configured to pass it through to the antenna.

When acting as a PCD in receive mode, the FPGA generates a constant 13.56MHz square wave on the signal_in pin, and the multiplexer connects both the envelope detector and the filtered 13.56MHz signal to the antenna.

When acting as a PICC in transmit mode, the FPGA uses the signal_in pin to control the load modulation output, which the multiplexer connects to the antenna.

When acting as a PICC in receive mode, the multiplexer connects the antenna directly to the envelope detector.

C. Errata and Modifications

There were some issues with the functionality of the PCB, some of which we were able to fix and some of which we weren't. This is a comprehensive list of the issues we ran into with our PCB and the modifications we made, intended to help any future group hoping to do a similar project.

1) PCD Output Amplitude: The signal path from signal_in through the LO amplifier/filter has significant attenuation. At the design phase, we had mis-estimated the amplitude that would be required at the coil to power an external PICC, so we did not prioritize passing a strong carrier to the output. The buffer output is several volts peak-to-peak, but the other side of the series resistor shows an amplitude of just a few hundred millivolts, so we expected that resistor saw a low impedance to ground somewhere afterwards, and couldn't figure out where that was. We tried changing this 820Ω series resistor to 220Ω, but this did not increase the output amplitude enough to make a difference. (We couldn't short this resistor entirely, since we need some way to sense the changes in current caused by the PCD's load modulation.) A future revision could increase this amplitude by adding some gain to the LO buffer, which might also require a higher supply voltage than the +5V and ±3.3V supplies we used.

2) Load Modulation Input: The PCB was designed to have the one signal_in pin used both to control load modulation in the PICC output mode, and to supply a 13.56MHz modulated signal in PCD input and output modes. However, when in PCD mode, it appeared that the load modulation input was loading down the 13.56MHz signal and attenuating it. Because of this, we cut the trace connecting signal_in to load modulation, and instead added a wire to route an unused pin to the input of load modulation. Ultimately, we did not measure that this change had any effect on the output amplitude.

3) Bypass Capacitor Values: While soldering the PCB, we were unable to immediately source a sufficient quantity of 100nF bypass capacitors, so we substituted all 100nF capacitors on the board with 220nF. We expect that this should have had minimal effect on the board's function.

4) Envelope Detector Input AC Coupling: The connection from the MUX to the envelope detector should be AC coupled, centered at ground. We cut the trace that connected them directly, and added a 100nF coupling capacitor, and a 1kΩ resistor to pull the envelope detector's DC input to ground.

5) Envelope Detector Low Pass Filter: The low-pass filter on the envelope detector's output that we put on the PCB did not adequately filter out the 13.56MHz component coming into its input. When we cut the traces at the low pass filter's input and output, and wired in another filter made of components with identical values, the filter worked. We don't know why the filter did not originally work. One suspicion we have is that the inductors we used had a self-resonant frequency below or near 13.56MHz, so we were seeing them either in a capacitative regime or their inductance was substantially lowered. Another possibility is that we soldered the wrong value of inductor on the PCB.

6) Variable Gain Amplifier Input Tuning: When we configured it with no gain or a low gain ratio, we saw significant overshoot (several times as strong as the signal of interest) in the variable gain amplifier's output, which would have made it hard to parse the output. Adding a 22pF compensation capacitor in parallel with the 10kΩ feedback resistor reduced this overshoot.

7) Input Power Switch: We neglected to include an on-off switch on the PCB, so we added it at the input on the 9V battery connector leads.

D. Final Functionality

The analog front-end is capable of supporting the FPGA acting as a PICC. Unfortunately, it's unable to support PCD mode, because of the inadequate output carrier amplitude.

XI. Testing and Demonstration (Matthew)

We tested our PICC emulation with several PCDs.

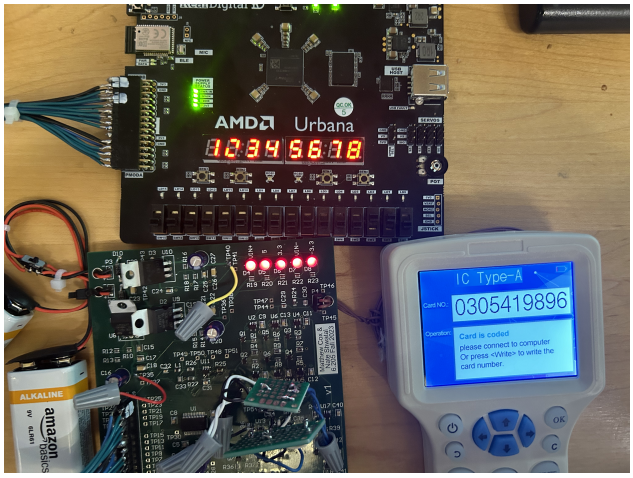

Fig. 11. Our PICC transmits 32'h12345678, and reader confirms that it receives 32'd030541896 (which equals 32'h12345678)

Additionally, even though the analog front-end didn't support PCD mode, we still were able to test the digital side of PCD mode by wiring up two FPGAs to each other and sending the internal modulation signal (i.e. the signal that would have been modulating the 13.56MHz carrier in the full system) directly to the other one's ADC input, to bypass the analog front-end; the PCD was able to read the UID that we'd configured the PICC to send. It's not as good of a test as it would be to test the PCD with a real PICC, but it's the best we could do given the state of the analog front-end.

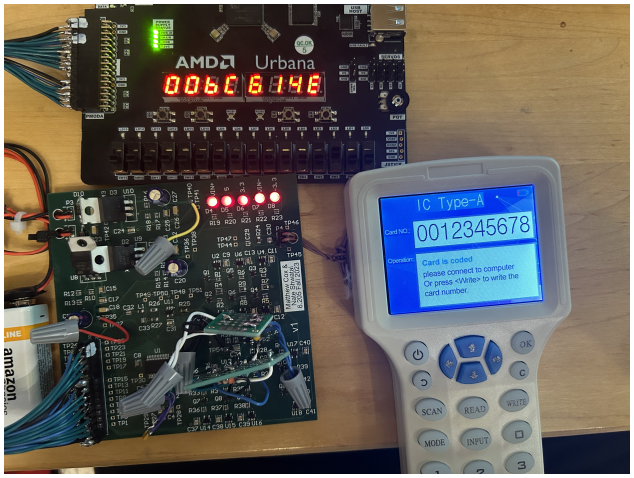

Fig. 12. Our PICC transmits 32'h00BC614E, and reader confirms that it receives 32'd0012345678 (which equals 32'h00BC614E)

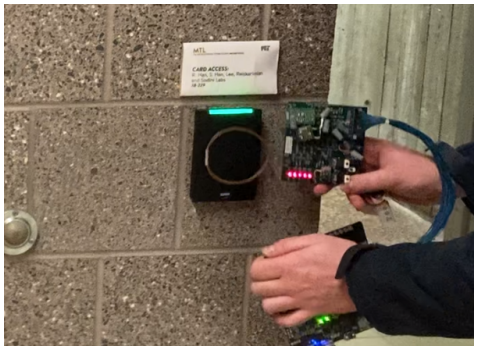

Fig. 13. Our PICC, emulating Matthew's MIT ID, successfully opens the door of a lab that Matthew has access to.

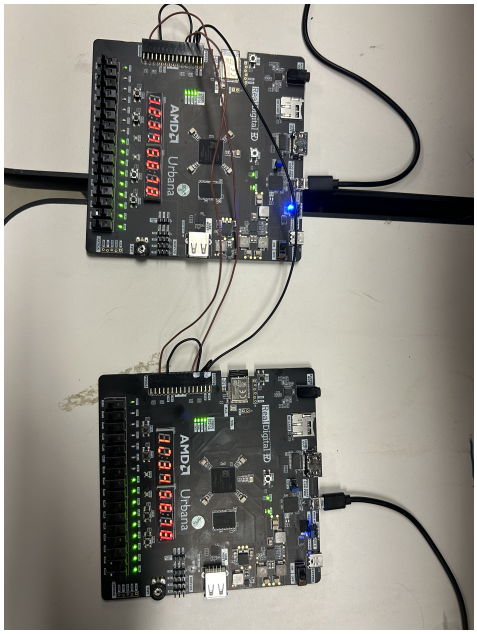

Fig. 14. FPGA acting as PCD reads UID 32'h12345678 from FPGA acting as PICC

XII. Timing & Resource Evaluation (Nathan)

Due to the nature of this protocol, clock-cycle latency is not a major concern of our system. The smallest transaction sent consists of a start bit plus 7 data bits, which lasts for 1024 carrier periods at a data rate of $f_c/128$. Since the carrier frequency is one-tenth of our main clock, any message will take at least 10,240 cycles to completely send. Additionally, the specification mandates waiting time of at least 1172 carrier periods (11,720 cycles) from the end of a received transmission to the start of the next output transmission. As such, these lengthy waiting periods dwarf any amount of time spent waiting for calculations to complete. The timing constraints enforced by the specification (on message length and waiting time) mean that a successful transaction can only take place approximately once every 300,000 clock cycles, about 2 ms.

To improve this throughput, the waiting times can be shortened, or the data can be transmitted faster. As significantly shorter wait times are not within spec, it is unclear how much they can be altered to maintain functionality. However, as mentioned above, ISO 14443 supports a collection of data rates faster than $f_c/128$. Implementing these in a future iteration would substantially increase the throughput, but there is a minimum 7000 carrier period delay (0.5 ms) between two requests from the PCD for any data rate. Again, this could be altered, but it would be outside the specification.

The slack for our final design fluctuates around zero (+0.036 ns on the most recent build). We had to pipeline a few of the calculations in the input modules to maintain positive slack, but we did not necessarily need to run the system at 135.6 MHz, either. Extra cycles are always useful, but they were not generally necessary. However, it would have been difficult to change the clock late in the design process.

Our design uses 10.14% of the board's slice look-up tables, and a very similar 9.77% of the board's slice registers. No DSPs or BRAM tiles were used. Both of these statistics indicate that there is plenty of room for future iterations to add new functionality on top of working logic for PICC/PCD authentication.

XIII. Source Files

All of our Verilog code, as well as the Altium files for the analog front-end PCB, are in our Github repository: https://github.mit.edu/coxm/6205-final-project

References

[1] Identification cards — Contactless integrated circuit cards — Proximity cards — Part 2: Radio frequency power and signal interface, ISO 14443-2, International Organization for Standardization, 2020.

[2] Identification cards — Contactless integrated circuit cards — Proximity cards — Part 3: Initialization and anticollision, ISO 14443-3, International Organization for Standardization, 2018.

[3] Identification cards — Contactless integrated circuit cards — Proximity cards — Part 4: Transmission protocol, ISO 14443-4, International Organization for Standardization, 2018.