

Low-Power Communication Circuits for Net-Zero-Energy IoT Nodes

by

Jaeyoung Jung

Bachelor of Science in Electrical Science and Engineering,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

© Massachusetts Institute of Technology 2022. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 6th, 2022

Certified by.....
Anantha P. Chandrakasan
Vannevar Bush Professor of Electrical Engineering and Computer
Science
Thesis Supervisor

Accepted by
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Low-Power Communication Circuits for Net-Zero-Energy IoT Nodes

by

Jaeyoung Jung

Submitted to the Department of Electrical Engineering and Computer Science
on May 6th, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

While Internet of Things (IoT) devices are increasingly widespread thanks to the lower cost of semiconductors, the reach of IoT is limited by the fundamental need of IoT devices for power. Currently, IoT devices are powered by a battery, which must be periodically replaced or recharged, or powered from the electrical grid, which limits mobility and requires a permanent wired connection. This thesis presents a solution in the form of the Net-Zero-Energy Device, an IoT node that can power itself by harvesting energy from 5G radiation. To transmit data from weak 5G signals, the energy harvester was optimized for sensitivity, and the active blocks were optimized to minimize their leakage power. The focus of this thesis is the design of the encoder block and backscatter circuit, which enable the IoT node to transmit data.

Thesis Supervisor: Anantha P. Chandrakasan

Title: Vannevar Bush Professor of Electrical Engineering and Computer Science

Acknowledgments

Thank you to my advisor, Professor Anantha Chandrakasan, as well as the MIT lead of this project, Professor Tomás Palacios. I am grateful to have had the opportunity to work on such an exciting and cutting-edge project under both of their guidance.

I would like to thank Ericsson for their financial and technical support of this project. Thank you especially to Dr. Jonas Hansryd and Dr. Miguel Lopez from Ericsson.

I would also like to thank Deniz Umut Yildirim and Dr. Mohamed Radwan Abdelhamid, members of Professor Chandrakasan's group who collaborated with me on this project. I appreciated their help and guidance throughout the course of the project. I would also like to thank Dr. Ahmad Zubair, Jiadi Zhu, and Sharon Hsia, members of Professor Palacios' group with whom we collaborated.

I would also like to thank the other members of Professor Chandrakasan's group, who have been welcoming and helpful throughout my time in the group, especially Maitreyi Ashok, Ray Chen, Dr. Yeseul Jeon, Eunseok Lee, and Saurav Maji. I could not have made the progress on this project that I did without their help.

I would also like to thank Dr. Muhammad Ibrahim Wasiq Khan, who helped us with the RF elements of the project.

Finally, I would like to thank my friends and family for their support and encouragement throughout my time at MIT.

Contents

1	Introduction	13
2	System Design	17
2.1	Energy Harvesting	18
2.2	Active Blocks	21
2.2.1	Supply-independent clock generator	21
2.2.2	Digital encoder	21
3	Encoder Design	23
3.1	Communication Protocol Design	23
3.1.1	Tail-biting Convolutional Code	24
3.1.2	Manchester Code	24
3.1.3	Cyclic Redundancy Check	25
3.2	Digital Block	25
3.2.1	Leakage-minimizing Design Techniques	25
3.2.2	Top Level	26
3.2.3	CRC Calculator	31
3.2.4	Convolutional Code Encoder	33
3.3	Miscellaneous Circuits	35
3.3.1	Power On Reset Circuit	35
3.3.2	Clock Driver	35
3.3.3	Summary	36

4	Simulation Results	39
4.1	RTL Simulation	39
4.2	Post-layout Analog Simulation	39
4.2.1	Simulated Performance	39
4.2.2	Full System Simulation	41
5	Backscatter Circuit Design	45
6	Conclusion and Future Work	51
A	Code Appendix	53
A.1	Digital Block RTL	53
A.1.1	zed_top.v	53
A.1.2	crc_reg.v	57
A.1.3	preamble.v	59
A.1.4	tbt_encoder.v	61
A.2	Digital Block Verification Testbenches and Scripts	64
A.2.1	zed_top_comprehensive_tb.v	64
A.2.2	verify_zed_top.m	65
A.2.3	crc_vecgen.py	67
A.2.4	crc_reg_verification_tb.v	68
A.2.5	tbt.py	70
A.2.6	tbt_encoder_verification_tb.v	71

List of Figures

1-1	Potential application for self-powering IoT nodes	14
2-1	System-level circuit diagram	18
2-2	RF power received by IoT node	19
2-3	Duty-cycled operation	20
3-1	Packet format	24
3-2	Top-level state diagram	27
3-3	Architecture of digital block	27
3-4	Manchester encoder with glitches	28
3-5	Output signal with glitches	29
3-6	Glitch-free Manchester encoder	30
3-7	Output signal without glitches	30
3-8	CRC calculator	31
3-9	CRC calculator waveforms	32
3-10	Convolutional code encoder	33
3-11	Convolutional code encoder waveforms	34
3-12	Power on reset circuit	36
3-13	Simulation of power on reset circuit	37
3-14	Clock generator, clock driver, and digital block	38
4-1	RTL simulation waveforms	40
4-2	Analog simulation waveforms	42
4-3	Full system simulation waveforms	43

5-1	Backscatter circuit diagram	47
5-2	Reflection coefficient vs. change in input impedance	49
5-3	Input impedance, reflection coefficient, modulation factor vs. number of fingers (assuming $\text{Im}\{Z_{ant}\} = -\text{Im}\{Z_{IC,off}\}$)	50

List of Tables

3.1	Specifications of packet components	24
4.1	Simulated performance metrics	41
5.1	Simulated impedance and reflection coefficient of integrated circuit in on and off states (assuming $Z_{ant} = 1 + 262j$)	49

Chapter 1

Introduction

The increasing availability and decreasing cost of semiconductor devices has enabled the rise of small, low-cost sensors with wireless communication capabilities. These sensors, known as Internet of Things (IoT) nodes, have applications in many areas, including environmental monitoring, agriculture, healthcare, surveillance, and the military [1]. However, with current technology, the reach of IoT nodes is limited by their need for power. As of now, IoT nodes can be powered from a battery, whose energy is limited and must be periodically replaced, or from the electric grid, which requires a wired connection. Creating an IoT node that can harvest its own power would greatly increase the reach of the Internet of Things and extend existing applications as well as enable new ones. Figure 1-1 shows a potential application for such an IoT node.

The problem of energy harvesting for self-powered sensors is a widely studied topic. Many approaches have been explored, including harvesting chemical energy stored in soil [2], harvesting thermal energy from the human body [3], harvesting kinetic energy of vibrations from industrial processes [4], and harvesting energy from received RF signals. The approach that is explored in this project is energy harvesting from RF signals because RF signals can be available regardless of the environment as long as a suitable RF transmitter is within the range of the IoT node, unlike other options for energy harvesting. While the limitation that an RF transmitter must be within the range of the IoT node still exists, it can be mitigated by increasing the range

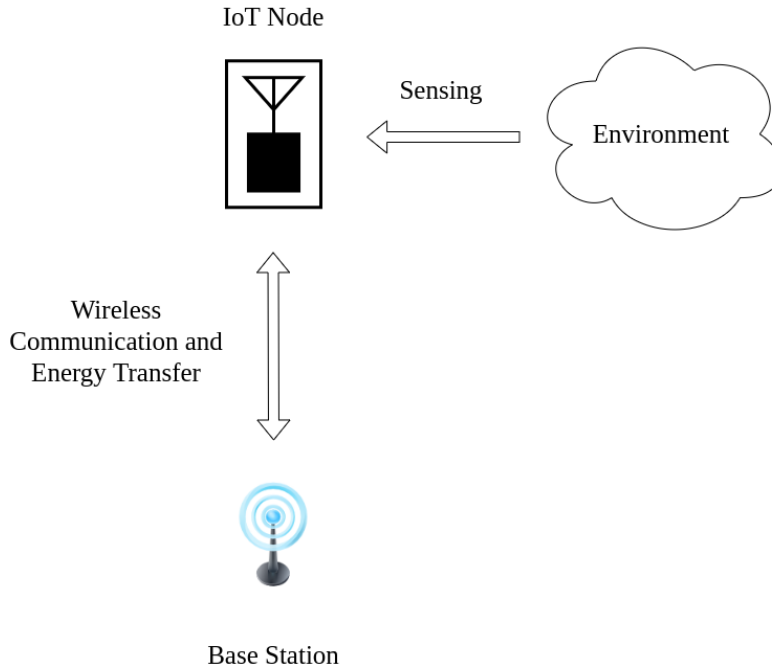


Figure 1-1: Potential application for self-powering IoT nodes

from the RF transmitter at which the IoT node can operate. As the power received by a client decreases with increasing distance between the transmitter and the client, improving the sensitivity, or minimum power level at which the IoT devices can turn on, increases the range.

In an RF energy harvesting system, a rectifier is used to turn the incoming cellular signals into a DC voltage, which can be used by circuits to perform useful work, such as computation and communication. Energy harvesting of RF signals has been a popular research topic in the circuits field [5], [6], [7]. However, previous works, which have invariably used silicon devices to implement the RF energy harvester, have not achieved the sensitivity required for this application. Because the peak AC voltage that is received by the rectifier is smaller than the threshold voltage of silicon transistors, the subthreshold swing of silicon transistors is the key limitation of using silicon transistors for RF energy harvesting. A transistor that can achieve a lower subthreshold swing could harvest energy from smaller voltages, and therefore RF signals of lower power levels. Professor Tomás Palacios' group at MIT has developed such a device: the negative capacitance field-effect transistor (NCFET). [8]

Integrating this novel device to develop a rectifier with state-of-the-art sensitivity is one of the goals of this project.

This thesis presents an integrated circuit for a proof-of-concept IoT node that is powered by 5G signals and sends data through a custom wireless communication protocol. Negative capacitance field-effect transistors will be integrated into the rectifier in order to improve the sensitivity of the energy harvesting system beyond what is possible with silicon transistors. In addition, to further improve sensitivity, efficient power management blocks, including a charge pump, comparator, and power switches were implemented.

The focus of this thesis is the active circuits that enable data transmission to a base station. A duty-cycled system architecture was designed with two phases: the idle phase, during which an energy storage capacitor is charged by the energy harvester, and the active phase, during which the active blocks of the system are powered. The active digital block of this system samples data from the input pins of the chip, computes a checksum, and sends an ID, the sampled data, and the checksum to the RF receiver through backscattering. These blocks were optimized to achieve two objectives: First, the leakage current while in the idle state should be minimized because high leakage current will reduce the rate at which energy is stored and potentially prevent the system from entering the active phase. Second, the blocks should consume minimal energy per bit of data sent while in the active state. By minimizing idle leakage and minimizing energy per data bit sent, the average data rate in periodic steady-state operation was maximized.

In this thesis, an overall system diagram is presented, followed by a detailed discussion of the active communication circuits.

Chapter 2

System Design

The following are the objectives of the project, which influenced the design of the overall system as well as its individual components.

- The system harvests RF energy with cutting-edge sensitivity, thanks to the integration of novel devices.
- The system uses the stored energy to transmit data.
- The average rate of data transmission is maximized through minimization of leakage power in the idle phase and energy per bit in the active phase.
- The system performs all of its operations autonomously with no external stimulus.

These objectives were considered when designing the architecture for the system, which is shown in figure 2-1. In the following sections, the design considerations to achieve these objectives for each of the major parts of the system will be discussed. The energy harvesting block, power management block, and supply-independent clock generator, which were designed by Deniz Yildirim, are briefly described in sections 2.1, 2.1, and 2.2.1, respectively, for context.

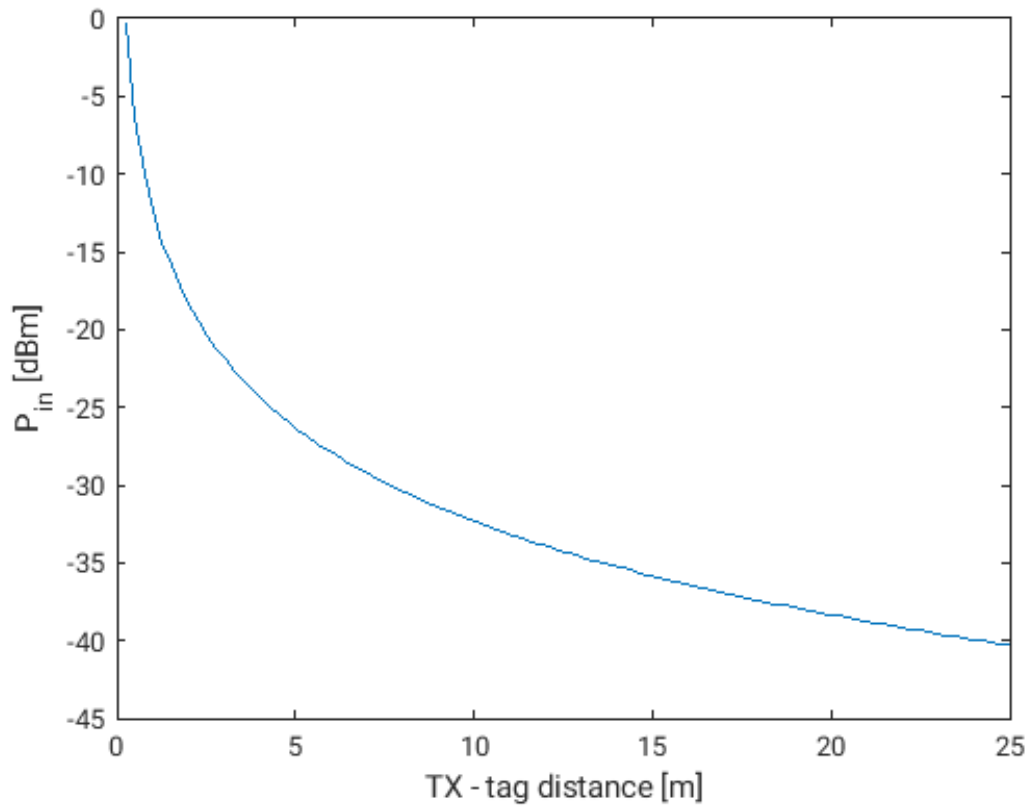


Figure 2-2: RF power received by IoT node

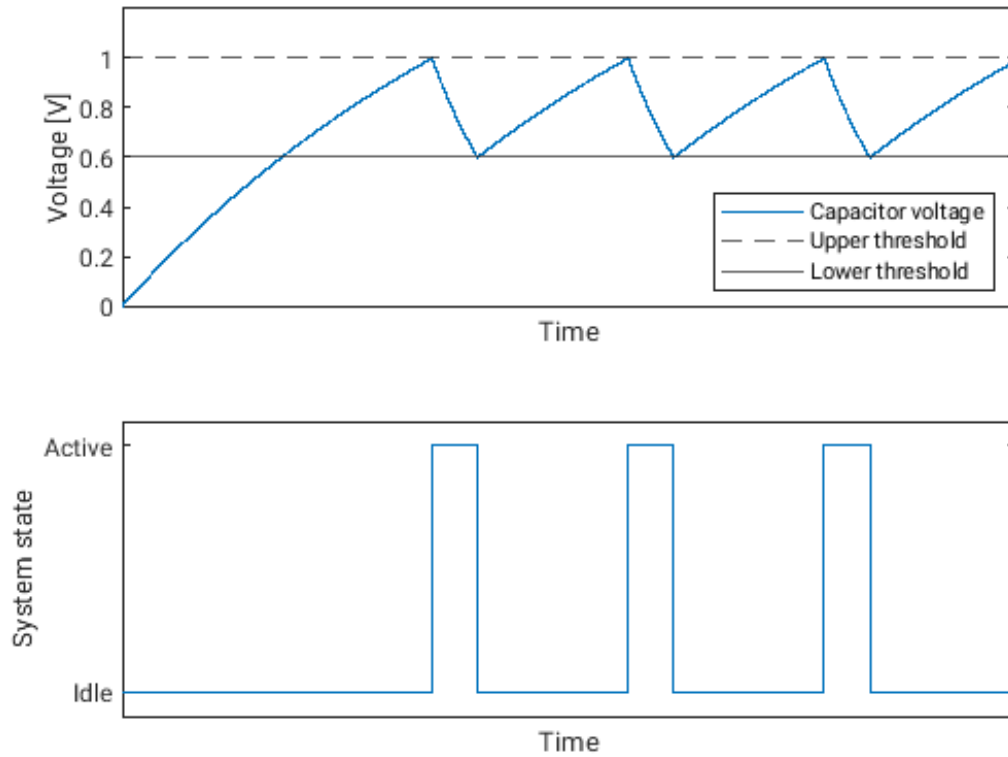


Figure 2-3: Duty-cycled operation

connects those circuits to the off-chip energy storage capacitor. Because the load is disconnected, the charge pump is able to charge the capacitor on net over time. When the voltage reaches a certain upper threshold, the comparator switches its output, closing the switch and powering the circuits from the energy stored in the capacitor. As the circuits use the stored energy, the voltage across the capacitor decreases. When the voltage reaches a lower voltage threshold, the comparator switches its output back, once again opening M1 and preventing the outflow of current from the capacitor. See figure 2-3 for a visual representation of this process. The duty-cycled nature of this system architecture allows for instantaneous power consumption that is higher than the instantaneous power extracted from the RF signal, allowing for operation from RF signals with lower power.

2.2 Active Blocks

The power-consuming active blocks in the system consist of a supply-independent clock generator, digital encoder, and backscatter switches. Note that there is no voltage regulator in this system. This implementation of the circuit consumes less energy than an one with a voltage regulator, due to the high power consumption of the voltage regulator. A voltage regulator is not necessary for the digital encoder in this circuit, since digital circuits can operate from a wide range of supply voltages. The digital circuit does, however, require a clock signal with a constant frequency to transmit data.

2.2.1 Supply-independent clock generator

In order to generate a clock signal with a constant frequency, Deniz Yildirim implemented a supply-independent clock generator based on a ring oscillator. Simply powering a ring oscillator from a varying supply voltage would cause the frequency of the output clock signal to vary as well, which is unacceptable for this system. To achieve a ring oscillator circuit whose output frequency is independent of the supply voltage, first, a reference, which produces a bias voltage, was implemented. A PMOS transistor whose gate is connected to this bias voltage produces a known, constant current. Then, a PMOS transistor, M3, was connected between the voltage supply and the supply terminal of the ring oscillator, with its gate connected to the bias voltage from the current reference. This approach regulates the current of the ring oscillator to a known value, which keeps the frequency of the generated clock signal constant.

2.2.2 Digital encoder

The digital circuit in this system samples parallel data bits from an external data source and encodes it into a bitstream. The coding scheme used to encode the data uses codes that increase the resiliency of the signal. This output bitstream is used to drive backscatter switches. When the gate voltage of the backscatter switches is

low, the switches are open, the matching between the external antenna and the chip is preserved, and incident RF power is absorbed. However, when the gate voltage of the backscatter switches is high, the switches are closed, the matching between the antenna and the chip is disturbed, and incident RF power is reflected. This effect is used to transmit an RF signal in an energy-efficient manner.

Chapter 3

Encoder Design

In order for the system to be able to send data, an encoder that samples data and encodes the data into a bitstream that can be transmitted as an RF signal is required. While designing this block, the goal of minimizing energy consumption per bit of data transmitted was considered. At the same time, the resiliency of the transmitted signal was maximized, since, due to the use of the backscatter effect for RF transmission, the effective transmitted power is inherently limited to the incident power.

The communication protocol used for data transmission and the digital circuit implementation of the encoder were co-designed in order to achieve these dual objectives.

3.1 Communication Protocol Design

The communication protocol was designed in collaboration with Dr. Miguel Lopez from Ericsson in order to maximize the resiliency of the signal while minimizing the energy required by the node to compute and transmit the packet. [10] The format of the packet is shown in figure 3-1, and the specifications of its components are shown in table 3.1.

The first component of the packet is the preamble, which is an 11-bit Barker code used by the receiver in order to identify the beginning of a packet. It is transmitted as a simple on-off signal without any encoding.



Figure 3-1: Packet format

Component	Preamble	Device ID	Data	CRC checksum
Data bit count	11	8	8	8
Encoding	None	Convolutional code and Manchester code		

Table 3.1: Specifications of packet components

The next components of the packet are the device ID, data, and the CRC checksum of the device ID and data. Since it is crucial that the data bits in these components of the packet are interpreted correctly by the receiver, they are encoded with a convolutional code and Manchester code. The convolutional code is used to increase tolerance for error in the reception of the signal. The Manchester code is used to create a self-clocking signal, which is necessary because the clock signal used by the digital encoder is generated by a ring oscillator that generates a frequency that varies by device, meaning that the receiver does not have prior knowledge of the frequency of the received digital signal.

3.1.1 Tail-biting Convolutional Code

The convolutional code used for this system is a tail-biting convolutional code, which increases the effective rate of data transmission compared to a zero-tail convolutional code by eliminating rate loss. The specific tail-biting code chosen comes from [9] and has the following parameters.

$$K = 8, m = 3, G = (54, 60)$$

3.1.2 Manchester Code

The signal is also encoded with a Manchester code as per the IEEE 802.3-2015 standard [11], where a logical "1" is encoded as a rising edge and a logical "0" is encoded as a falling edge. This means that an encoder can be implemented by simply performing an exclusive or (XOR) of data and clock signals.

3.1.3 Cyclic Redundancy Check

A Cyclic Redundancy Check (CRC) checksum is included as one of the components of the packet format. This checksum is used by the receiver to verify that the data of the received packet were received correctly. The format of the CRC is based on the format used by the IEEE 802.11 standard. [12]

- All registers are initialized to 1.
- The CRC generating polynomial is $G(D) = D^8 + D^2 + D + 1$.
- The output bits are inverted.

3.2 Digital Block

The digital block was designed such that it generates a bitstream from the sampled data according to the communication protocol defined in the previous section while consuming the minimal amount of energy per data bit transmitted.

3.2.1 Leakage-minimizing Design Techniques

Due to the low operating clock frequency of this block (17 kHz), its power consumption is dominated by leakage power rather than dynamic power. The following sections describe techniques to reduce the leakage current of the digital block.

Using High Threshold Voltage (HVT) Transistors

Leakage current decreases exponentially as threshold voltage, V_T , increases.

$$I_{leak} \propto \exp\left(\frac{-V_T}{n\frac{kT}{q}}\right)$$

Therefore, implementing a digital design with transistors with a higher threshold voltage reduces its leakage current. However, increasing the threshold voltage decreases the overdrive voltage of the transistors $V_{DD} - V_T$, with which the on-current scales linearly, assuming that the device is velocity saturated:

$$I_{on} \propto (V_{DD} - V_T)$$

This scaling creates a tradeoff between the leakage current and the speed of the block. Due to the low frequency of the clock signal, the constraints on timing and the overdrive voltage are very relaxed. Therefore, this block was implemented with transistors with the highest threshold voltage possible.

Transistor Count Reduction

To the first order, leakage current and power scale linearly with the number of transistors in a design. Therefore, the number of transistors in a digital block should be minimized to minimize its leakage power. To minimize the number of transistors, elements of the digital block were reused wherever possible, and the number of flip-flops used was minimized. Creating multiple instances of the same design in the block increases the number of transistors in the design, of course, so it was avoided wherever possible. Minimizing the number of flip-flops in the design was important for minimizing transistor count because RTL synthesis tools can replace combinational logic with simpler but logically equivalent combinational logic to minimize power and cell count, but they cannot reduce the number of sequential logic cells, including flip-flops. These techniques were used for every level of design of the digital block.

3.2.2 Top Level

Design

This block was designed as a state machine with five states, shown in figure 3-2, with one idle state and one state for each of the components of the data packet. The architecture of the digital implementation is shown in figure 3-3. Since only one state can occur at any time, an operation that is used by multiple states can be performed by the same hardware. This is the case for convolutional code encoding, which is used when sending the device ID, sampled data, and CRC. Therefore, there is only one convolutional code encoder in this design, whose input is connected to a multiplexer

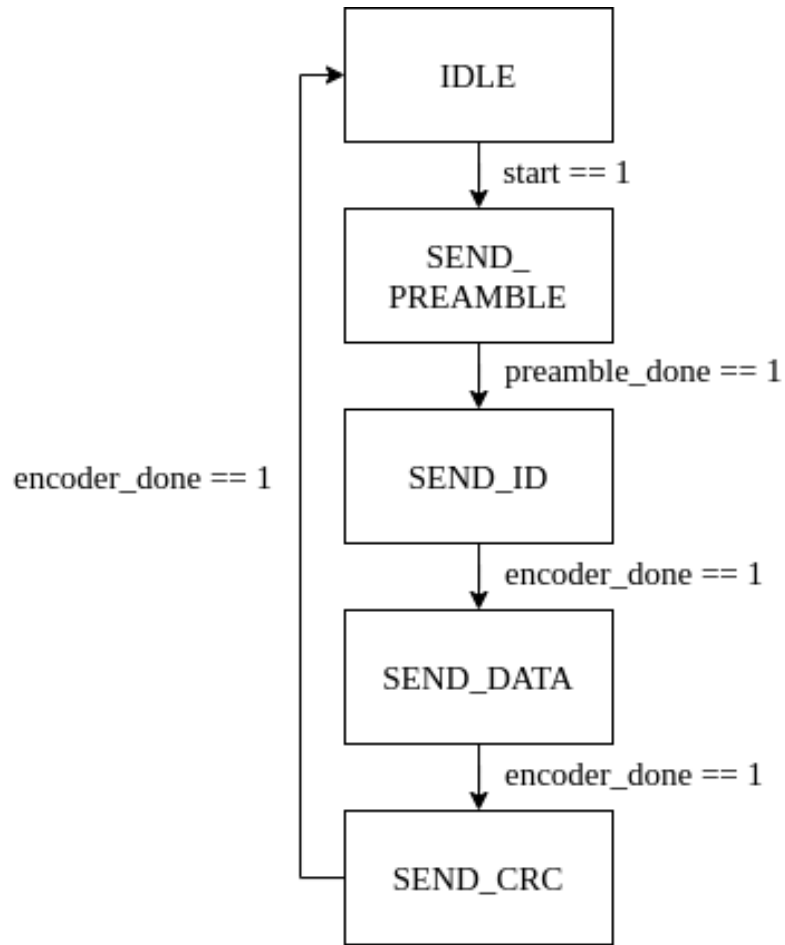


Figure 3-2: Top-level state diagram

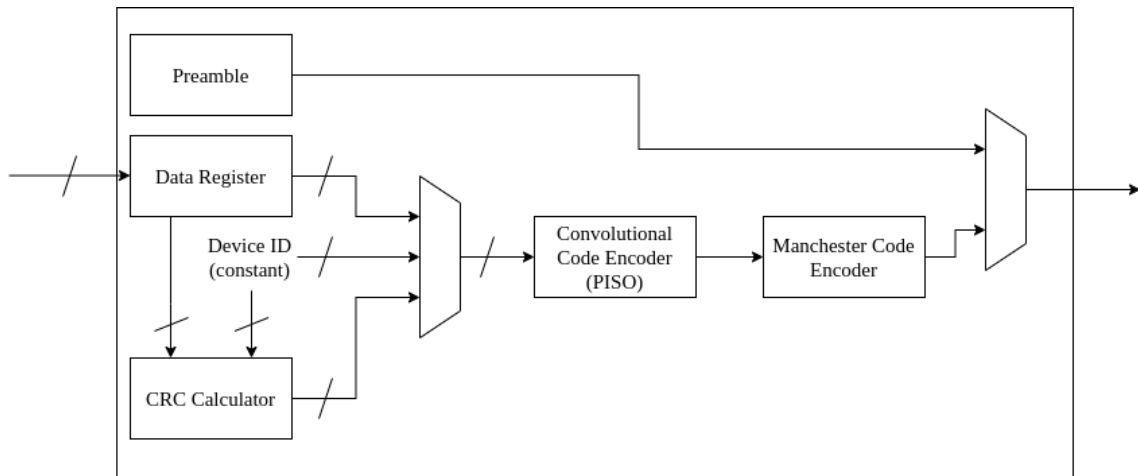


Figure 3-3: Architecture of digital block

with three inputs: ID, data, and CRC. One of these inputs is selected depending

on the state, and the same convolutional code encoder is used to encode all of these serially.

When the start input signal is asserted in the IDLE state, the state machine enters the first data transmission state, SEND_PREAMBLE, and starts the calculation of the CRC. After the transmission of the preamble, which is stored on the chip as a constant, is complete, it goes through the three states that use the convolutional code encoder - SEND_ID, SEND_DATA, and SEND_CRC. While in any of these three states, the state machine transitions to the next state when the convolutional code encoder is done encoding its input. This way, a continuous stream of encoded data bits, whose sequence matches the packet format, is output by the block. The CRC calculator calculates the CRC by the time the state machine reaches the SEND_CRC block, which, as the name suggests, uses the output of the CRC calculator.

This block also has a Manchester encoder used to turn the output signal into a self-clocking signal, which is used for the last three states, SEND_ID, SEND_DATA, and SEND_CRC. It was implemented as an XOR gate. Simply performing an XOR operation with clock as one input and the data as the other results in an output with glitches due to differences in signal timing. See figure 3-4, where the difference in timing between the red and blue signal paths causes glitches in the output signal. Figure 3-5 shows the output waveform of the Manchester encoder, which contains glitches.

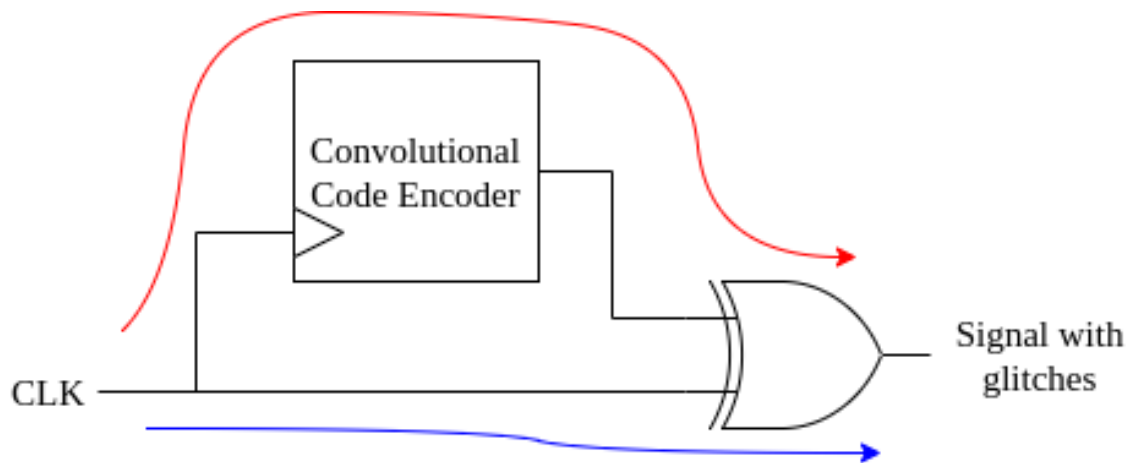


Figure 3-4: Manchester encoder with glitches

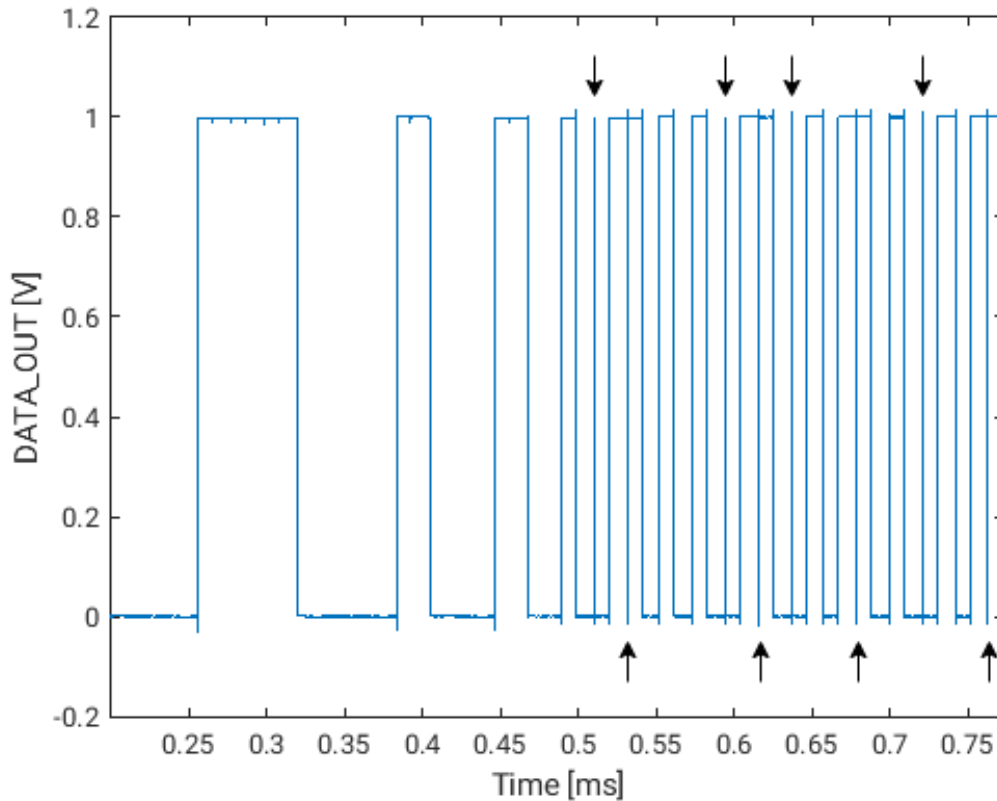


Figure 3-5: Output signal with glitches

Because the output signal changes on both the rising and falling edges of the clock, removing the glitches is not as simple as placing a flip-flop at the output. The solution was to generate a secondary clock whose frequency is half that of the original clock. The original clock was used as the clock for the flip-flop at the output of the encoder, while the slower secondary clock was used as the clock signal for the Manchester encoder. In addition, the state machine of the convolutional code encoder was changed so that it would only transition on every other cycle to match the timing of the Manchester encoder, as shown in figure 3-6. Figure 3-7 shows the glitch-free output waveform of this new implementation.

Verification

This block outputs the packets defined in the previous section serially as a bitstream. Therefore, to verify the functionality of this block, it was necessary to verify that the

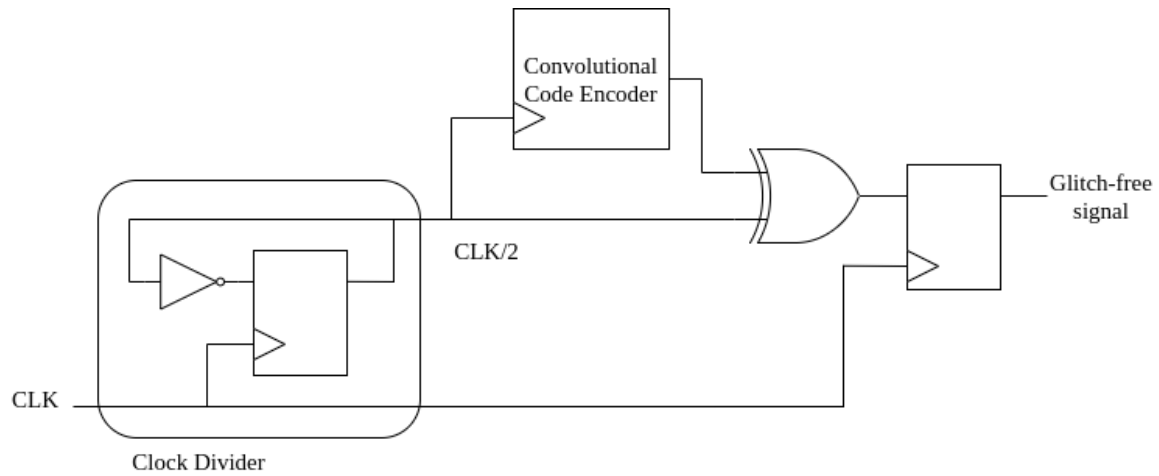


Figure 3-6: Glitch-free Manchester encoder

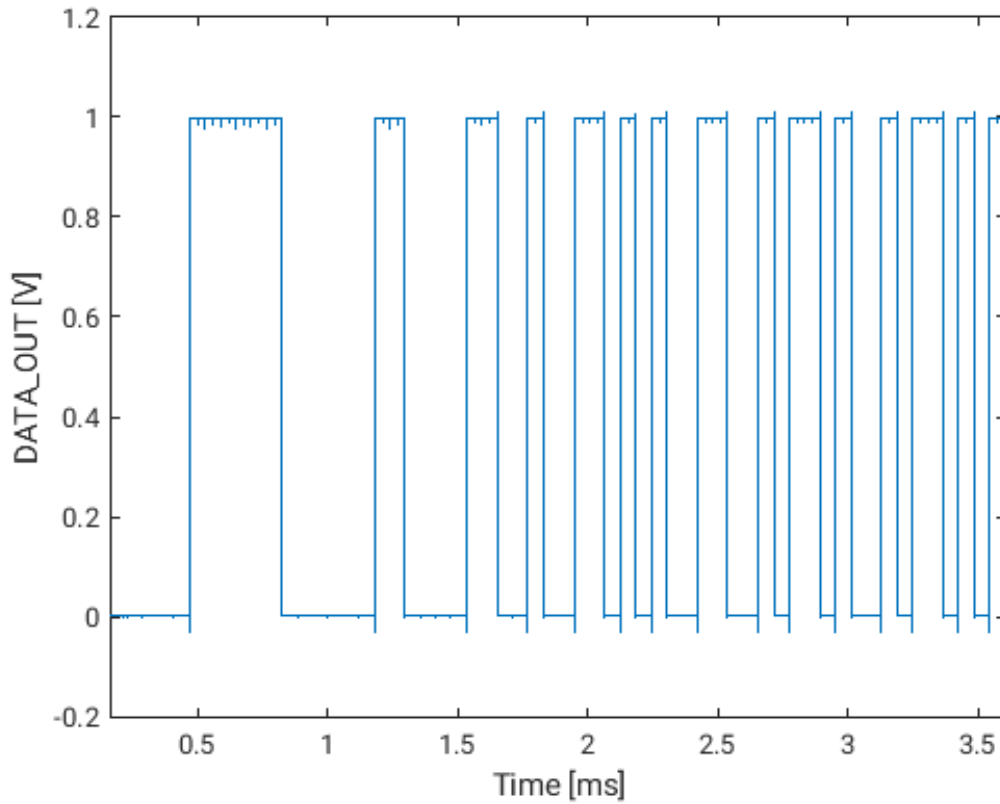


Figure 3-7: Output signal without glitches

output packet was consistent with the input data bits, and that the output packet was internally consistent.

The testbench for the block input all possible values, ranging from 0 to 255, into

the block, and asserted the start signal for each value. The block output a packet for each input value. A MATLAB script, shown in appendix section A.2.2, was used to verify the correctness of each of the output packets. This script looked for the uncoded 11-bit Barker code, which signifies the beginning of a packet. Since the rest of the packet, the ID, data, and CRC, is encoded with the Manchester code and convolutional code, the script decoded it into an uncoded signal, and extracted the ID, data, and CRC. The script verified that the ID and data were consistent with what it expected. It then calculated the CRC from the ID and data and verified it with the CRC from the waveform.

This verification script was run on the digital block before it was synthesized, implemented, and integrated into the silicon.

3.2.3 CRC Calculator

Design

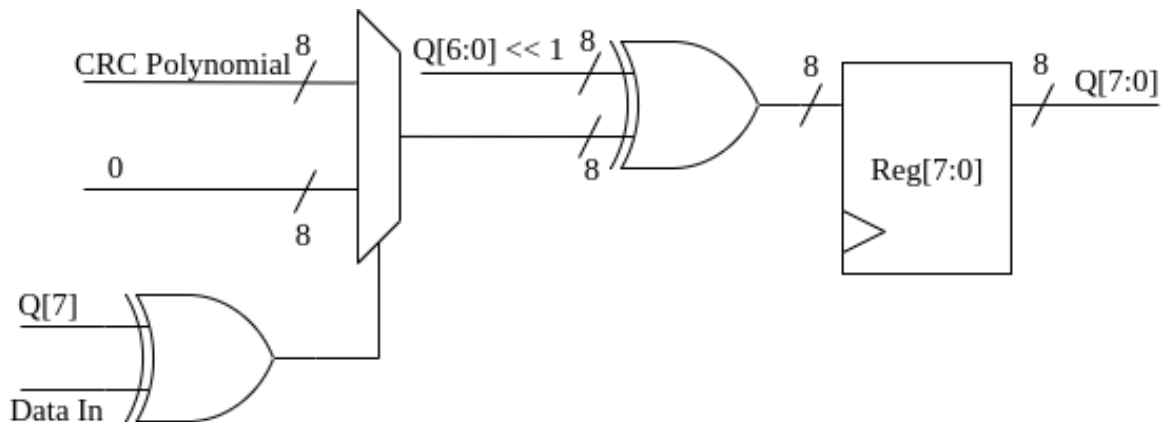


Figure 3-8: CRC calculator

Since the packet has CRC as one of its components, a block that computes the CRC of the ID and data was necessary. The inputs to the CRC calculator block are the ID and data. When the start signal is asserted, the process of computing the CRC is begun.

The CRC calculator was implemented as a modified shift register, shown in figure 3-8. The operation is explained below.

1. Perform XOR (1-bit addition) of the input data bit and the MSB in the register array.
2. Perform 1-bit left-shift of the register array values, discarding the MSB.
3. If the result of (1) was zero, perform a bitwise XOR of the result of (2) and zero. Otherwise, perform a bitwise XOR of the result of (2) and the CRC polynomial.
4. Save the output of (3) in the register array.
5. Repeat until all data bits are shifted in.

For eight cycles, bits of the ID are sequentially shifted into the "Data In" input. Then, for the next subsequent eight cycles, bits of the sampled data are shifted in. When all bits are shifted in after sixteen clock cycles, the computation of the CRC checksum is complete, and the valid signal is asserted. See figure 3-9 for example output waveforms of the CRC calculator. The eight output bits of the register array can be accessed by the convolutional code encoder, which transforms the CRC into an encoded serial bitstream.

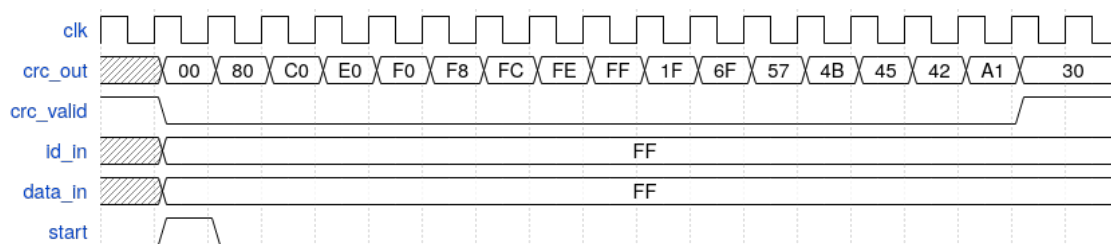


Figure 3-9: CRC calculator waveforms

Verification

This block computes an 8-bit CRC from two 8-bit input signals: the ID and data. To verify this functionality, the testbench, shown in appendix section A.2.4, input all $2^{(8)(2)} = 65536$ possible combinations of 2 8-bit values and verified that the output CRC matched the precomputed value in each case.

3.2.4 Convolutional Code Encoder

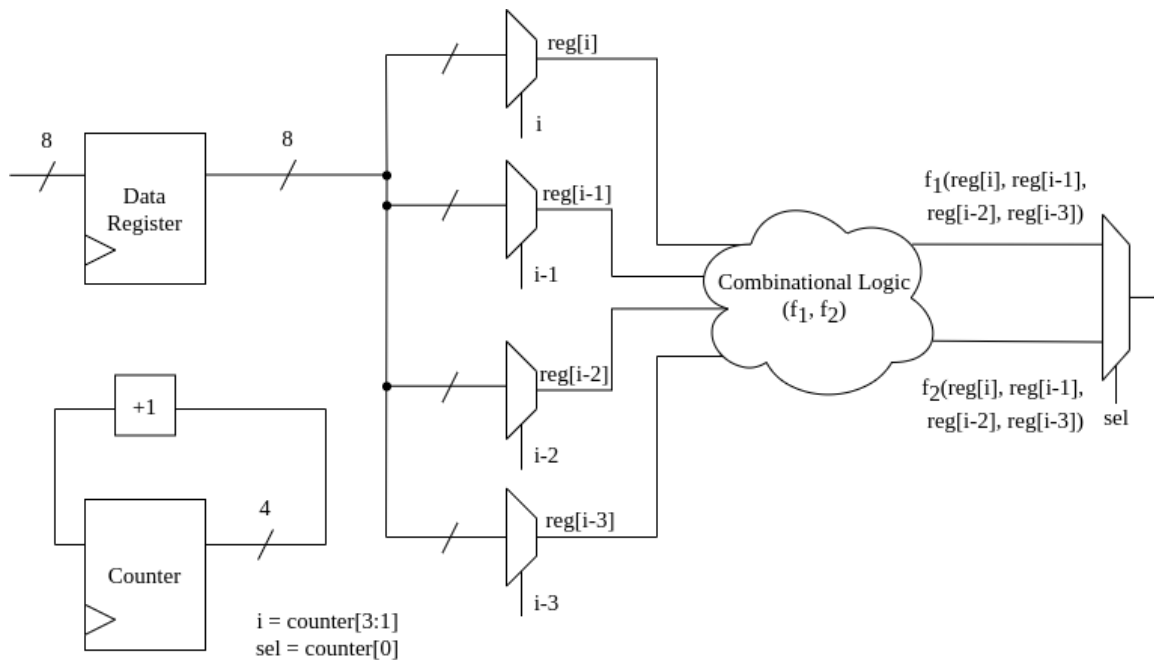


Figure 3-10: Convolutional code encoder

Design

When designing the convolutional code encoder, care was taken to minimize logic and flip-flop count. In fact, this block only has five flip-flops in total: four for the counter, and one for the state machine.

Recall that a tail-biting convolutional code with the following parameters was chosen as the encoding scheme.

$$K = 8, m = 3, G = (54, 60)$$

Encoding 8 bits of data u_0, \dots, u_7 with this convolutional code results in 16 bits of encoded data v_0, \dots, v_{15} . For integers $i \in [0, 7]$:

$$v_{2i} = u_i \oplus u_{i-2} \oplus u_{i-3}$$

$$v_{2i+1} = u_i \oplus u_{i-1}$$

(For $j < 0$, let $u_j = u_{j+8}$.)

This was implemented in hardware by setting the functions f_1 and f_2 as:

$$f_1(\text{reg}[i], \text{reg}[i - 2], \text{reg}[i - 3]) = \text{reg}[i] \oplus \text{reg}[i - 2] \oplus \text{reg}[i - 3]$$

$$f_2(\text{reg}[i], \text{reg}[i - 1]) = \text{reg}[i] \oplus \text{reg}[i - 1]$$

As the counter is incremented, the *sel* signal alternates between 0 and 1, alternating the outputs between the two signals generated by the two combinational logic functions f_1 and f_2 . When i is incremented, it changes which data bits are input into the combinational logic functions f_1 and f_2 . This results in a serial stream of data bits that are encoded with the tail-biting trellis convolutional code specified in the packet format. See figure 3-11 for example waveforms of the convolutional code encoder. Notice that data transitions only occur when the phase signal is high, which happens every other cycle. This is to accommodate for the half-frequency clock used by the Manchester encoder, which uses the output of this block.

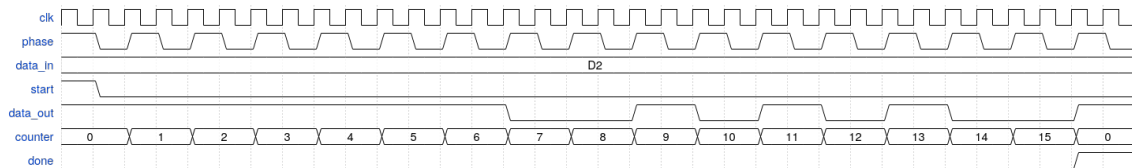


Figure 3-11: Convolutional code encoder waveforms

Verification

A Python script, shown in appendix section A.2.5, was used to encode each of the 256 possible 8-bit input data. The verification testbench, shown in appendix section A.2.6, input all possible input data into the block and compared the serial output data with the precomputed encoded data.

3.3 Miscellaneous Circuits

Additional elements were required to integrate the digital block into the integrated circuit. They are described below.

3.3.1 Power On Reset Circuit

When a digital block which was previously off is powered on, the states of the flip-flops in the block are undefined. This is problematic, as the desired behavior of the digital block is that it begins transmitting data when it is powered on. Therefore, it is necessary to reset all of the flip-flops in the design to a known state as soon as it is powered on. In order to accomplish this, a power on reset circuit was implemented.

The power on reset circuit was implemented as an RC circuit, shown in figure 3-12. When the circuit is first powered on, the voltage across the capacitor is zero due to the pulldown diode-connected NMOS transistors, which discharges the capacitor during the charging phase. The buffer outputs a "low" signal, asserting the reset signal. Over time, the capacitor is charged by the diode-connected PMOS transistor. The pulldown network consists of three diode-connected transistors in series so that the pulldown network is weaker than the pullup network. When the voltage across the capacitor reaches the switching point of the buffer, the output of the buffer switches from low to high, deasserting the reset signal and starting the operation of the digital block. Simulated waveforms of the "RESET_UNBUFF" and "RESET" nets are shown in figure 3-13.

3.3.2 Clock Driver

The clock net is the net with the highest fanout in the digital block; therefore, it is important that the net be driven with a strong driver. This clock driver was implemented as an inverter, whose input is driven by the supply-independent clock generator.

In order to prevent glitches from cross-coupled nets from causing glitches on the clock net and disrupting the operation of the digital block, a capacitor was connected

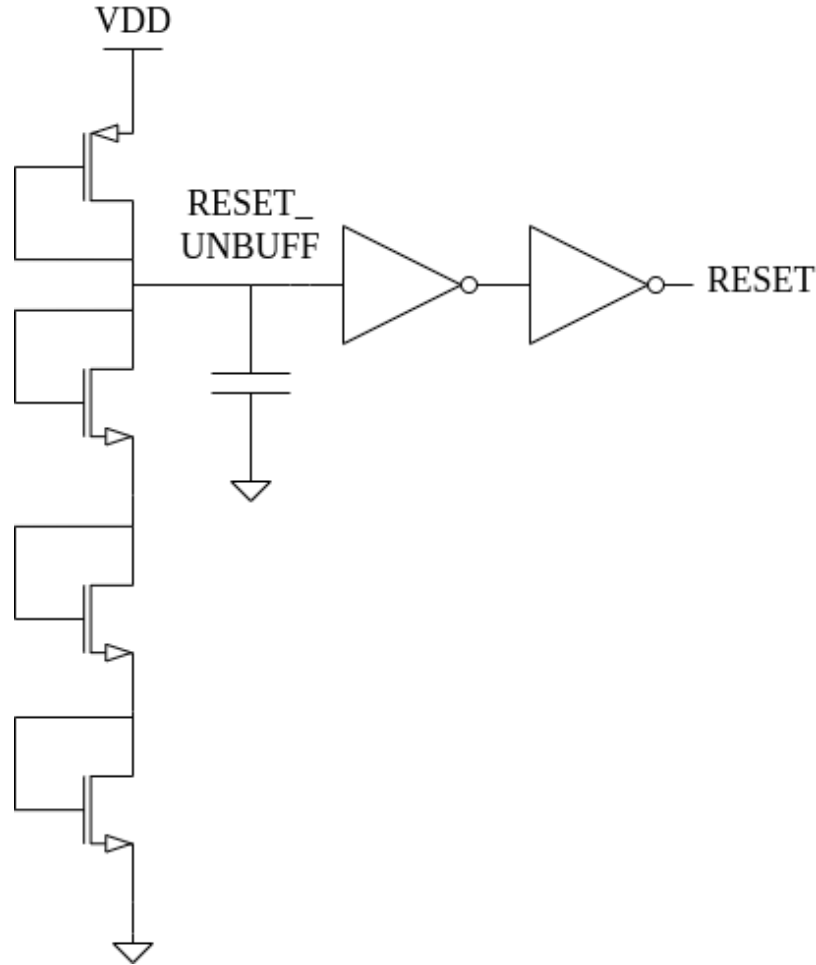


Figure 3-12: Power on reset circuit

to the clock net in order to mitigate the effect from cross-coupled nets. The clock driver circuit is shown in figure 3-14.

3.3.3 Summary

In this chapter, the protocol used for data transmission, which consists of a preamble, ID, data, and CRC checksum, the latter three of which are encoded with a convolutional code and Manchester code, was introduced. The implementation of the digital encoder, which includes a CRC calculator and convolutional code encoder, was described. In order to minimize the leakage power of this block, high threshold voltage transistors were used, and the number of transistors was minimized by reducing the number of flip-flops and reusing digital elements whenever possible. Finally, the power

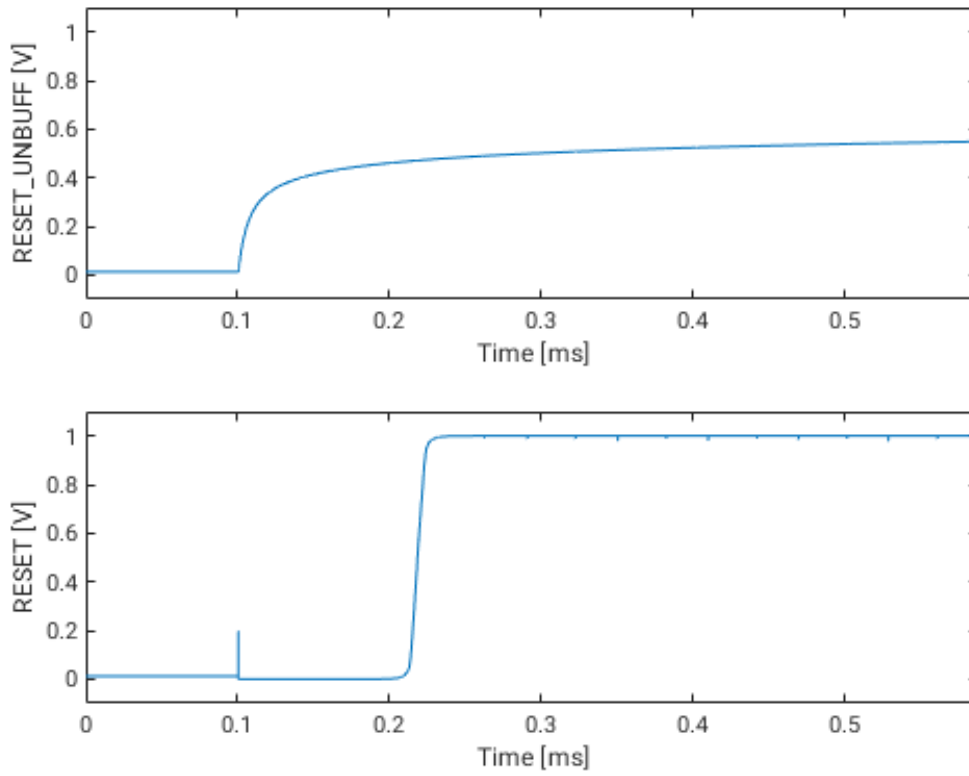


Figure 3-13: Simulation of power on reset circuit

on reset circuit and clock driver, which are required for the operation of the digital block, were described.

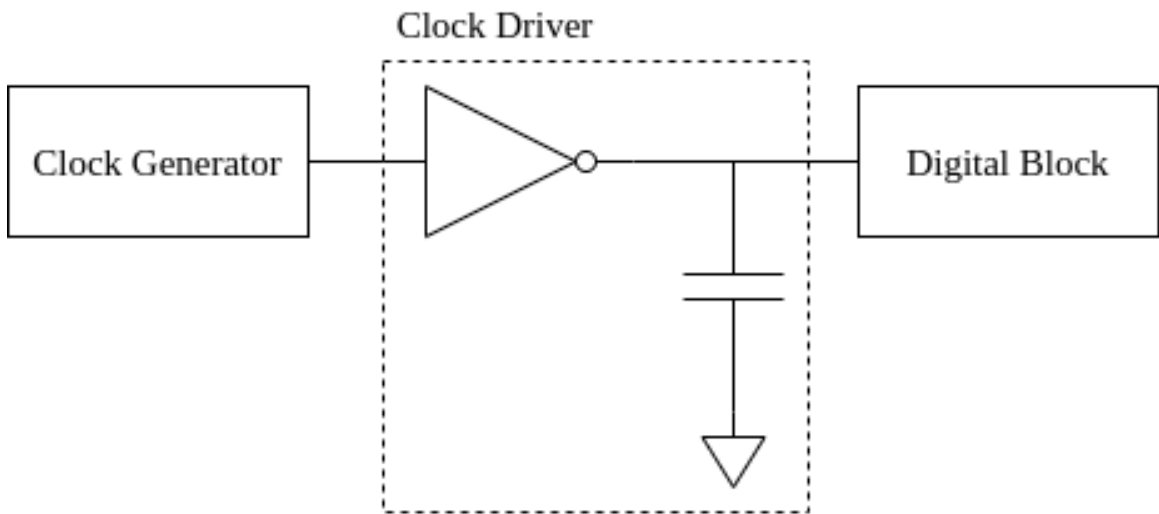


Figure 3-14: Clock generator, clock driver, and digital block

Chapter 4

Simulation Results

4.1 RTL Simulation

Figure 4-1 shows the result of an RTL simulation of the digital block with an input data of D2. The output (DATA_OUT) and state (STATE) signals are shown. The simulation testbench is given in A.2.1.

4.2 Post-layout Analog Simulation

The digital block, after synthesis, implementation, and integration with the clock generator and driver and power-on-reset circuit, was simulated with Cadence Virtuoso ADE XL with an input data of D2. The reset (RESET), output (DATA_OUT), and state (STATE[2], STATE[1], and STATE[0]) signals are shown in figure 4-2. Note that the output and state signals are consistent with the result of the RTL simulation.

4.2.1 Simulated Performance

As mentioned earlier, the two measures of performance of this block are the leakage power consumption in the idle state and the energy consumed per transmitted bit in the active state. The following performance metrics were calculated from the analog simulation described earlier, where $V_{DD} = 1V$.

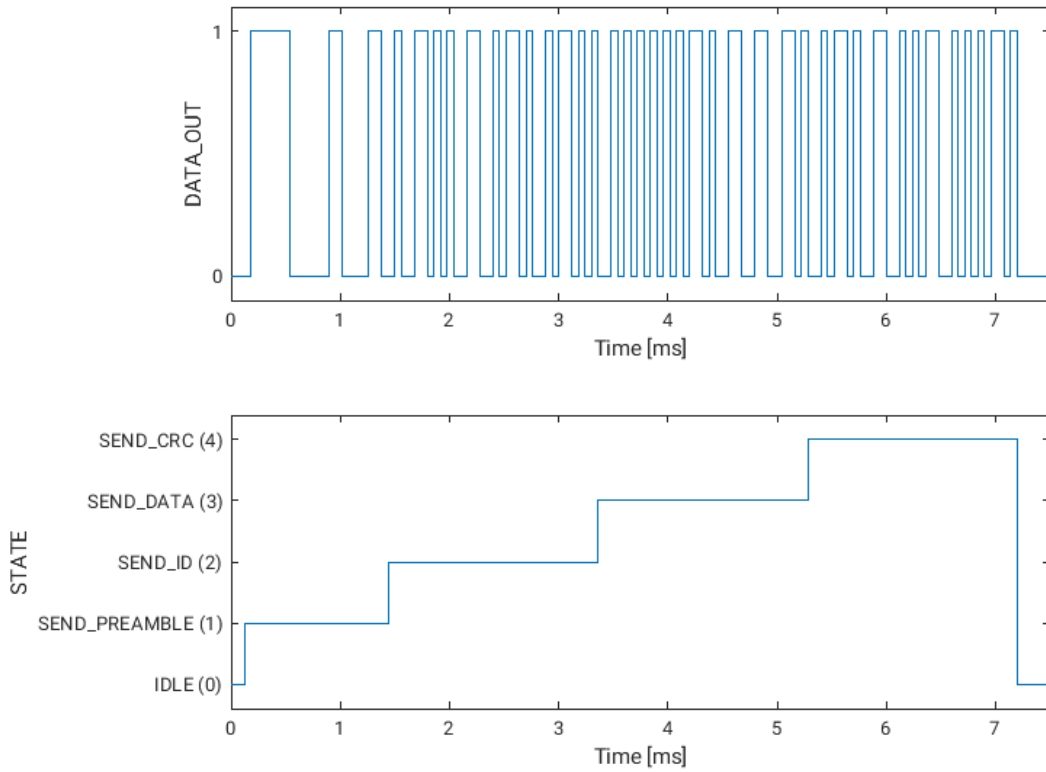


Figure 4-1: RTL simulation waveforms

The leakage power was calculated by simply multiplying the supply voltage by the supply current of the block while the block is in the idle state.

$$P_{leak} = V_{DD}I_{leak}$$

The energy consumed per transmitted bit was calculated by integrating the instantaneous power consumption of the block over the time period required to send a single packet, and dividing by 8, the number of bits in a packet. The instantaneous power is given by the supply voltage multiplied by the supply current.

$$E_{bit} = \frac{1}{8} \int_{packet} V_{DD}I_{active}(t)dt$$

Leakage power	127 pW
Energy per bit	0.70 nJ

Table 4.1: Simulated performance metrics

4.2.2 Full System Simulation

Figure 4-3 shows the results of a full system simulation with an input data of E5. The two waveforms shown are \sim COMMEN, which is the active-low enable signal for the system, and the DATA_OUT signal. \sim COMMEN remains high until the stored voltage reaches the upper threshold, at which point it is switched to low, and the digital block begins encoding data. The decay in the stored voltage is visible in the DATA_OUT waveform. Despite this decay, the operation of the block is unaffected thanks to the supply-independent clock generator.

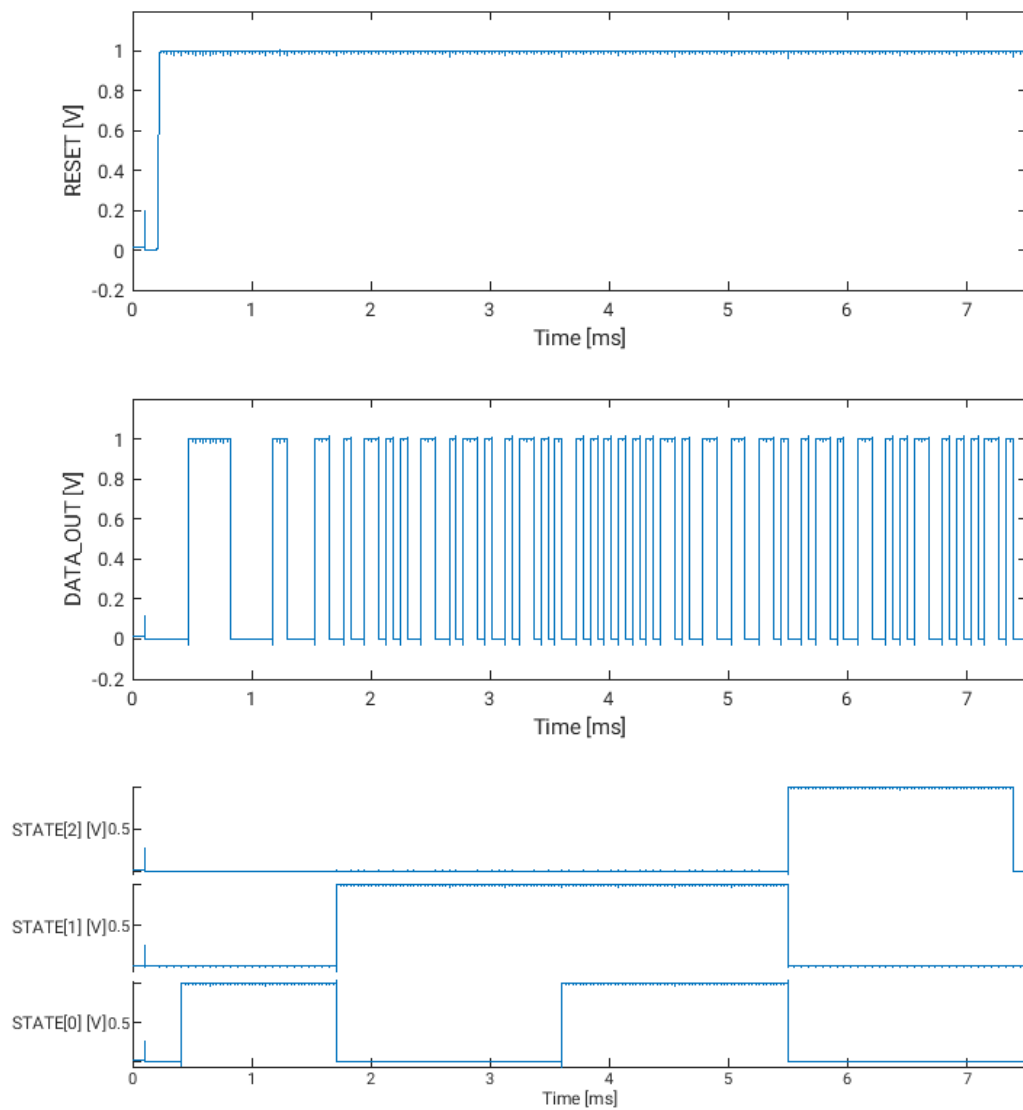


Figure 4-2: Analog simulation waveforms

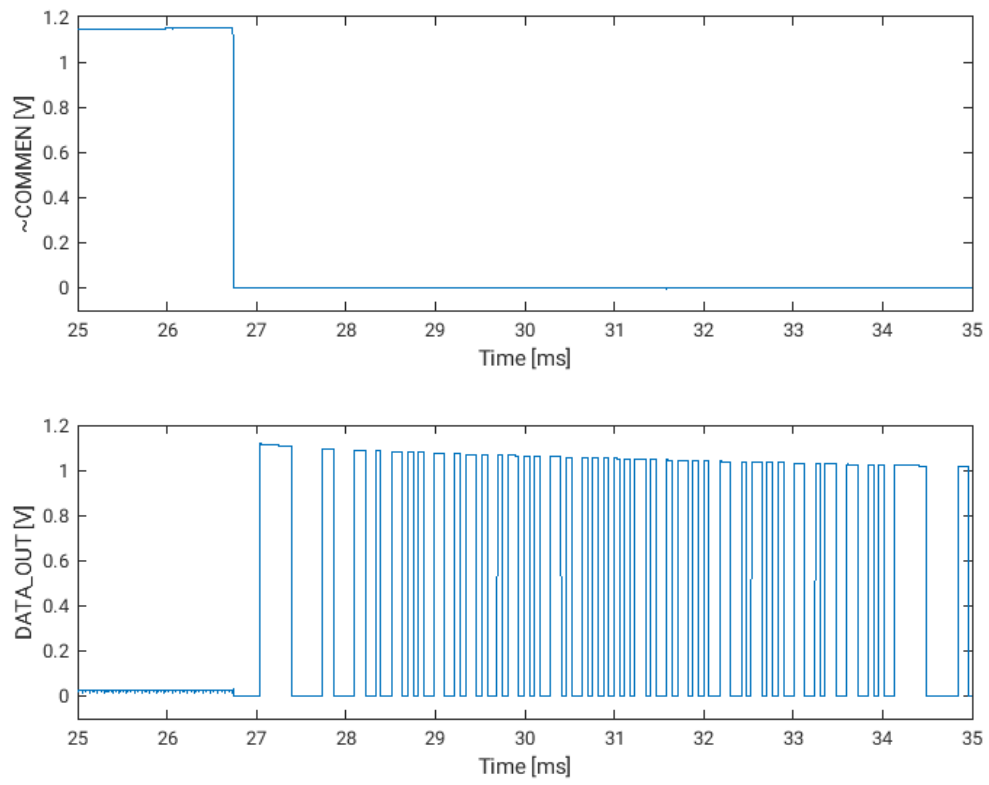


Figure 4-3: Full system simulation waveforms

Chapter 5

Backscatter Circuit Design

Backscatter communication is a method of wireless data transmission commonly used in applications with very low energy constraints, including passive radio-frequency identification (RFID) tags. The basic theory of operation is as follows: Consider an antenna connected to an integrated circuit. The input impedances of the antenna and integrated circuit are respectively denoted Z_{ant} and Z_{IC} . The reflection coefficient Γ for this antenna-integrated circuit system is given by the following expression. [13]

$$\Gamma = \frac{Z_{IC} - Z_{ant}^*}{Z_{IC} + Z_{ant}}$$

Recall that, in order to utilize passive RLC resonance and maximize power transfer, the impedance of the antenna and the impedance seen by the antenna looking into the RF input port of the integrated circuit should have opposite imaginary components, and the imaginary components of the impedances should be larger than their real components in order to maximize Q . Ideally, for maximum power transfer, the real parts of the two impedances should equal, but this is difficult to achieve while also satisfying the constraint on the imaginary components of the impedances. While the real parts are not exactly equal, they are on the same order of magnitude.

$$\text{Re}\{Z_{IC}\} \approx \text{Re}\{Z_{ant}\}$$

Based on this, the reflection coefficient Γ is relatively close to 0, meaning that

incident RF waves will be largely absorbed by the antenna-rectifier system.

$$\Gamma = \frac{(\text{Re}\{Z_{IC}\} + j\text{Im}\{Z_{IC}\}) - (\text{Re}\{Z_{ant}\} - j\text{Im}\{Z_{ant}\})}{(\text{Re}\{Z_{IC}\} + j\text{Im}\{Z_{IC}\}) + (\text{Re}\{Z_{ant}\} + j\text{Im}\{Z_{ant}\})} = \frac{\text{Re}\{Z_{IC}\} - \text{Re}\{Z_{ant}\}}{\text{Re}\{Z_{IC}\} + \text{Re}\{Z_{ant}\}} \approx 0$$

On the other hand, consider the case in which $Z_{IC} = 0$. Then, the reflection coefficient takes the following form.

$$\Gamma = \frac{-Z_{ant}^*}{Z_{ant}} = \frac{-\text{Re}\{Z_{ant}\} + j\text{Im}\{Z_{ant}\}}{\text{Re}\{Z_{ant}\} + j\text{Im}\{Z_{ant}\}} \approx \frac{\text{Im}\{Z_{ant}\}}{\text{Im}\{Z_{ant}\}} = 1$$

In this case, incident RF waves will be largely reflected by the antenna. The two states, RF absorption and reflection, can represent the two possible states of a single bit: for instance, a high bit could map to reflection, and a low bit could map to absorption. The receiver can detect whether the RF wave was reflected and accordingly determine which state was transmitted by the node. The modulation factor, or the effective difference between the reflection coefficients in the two states A and B , is given by the following equation. [13]

$$M = \frac{1}{4} |\Gamma_A - \Gamma_B|^2$$

The effective power of the transmitted backscattered signal is proportional to the modulation factor M . In the ideal case, Γ_A and Γ_B equal to 1 and -1 , which results in $M = 1$, meaning that 100% of received power will be reflected. In practice, this is difficult to achieve, so only a fraction of received power can be reflected through backscattering. The backscatter circuit was designed to achieve a large value of M .

The backscatter circuit implementation used for this work is shown in figure 5-1. Each input terminal of the chip has a MOSFET connected between the terminal and ground. The gates of the two MOSFETs are connected together so that the states of the two MOSFETs are controlled together. When the gate voltage is low, the transistors do not conduct any current, making a minimal impact on the input impedance of the chip. When the gate voltage is high, the transistors conduct, providing a path for

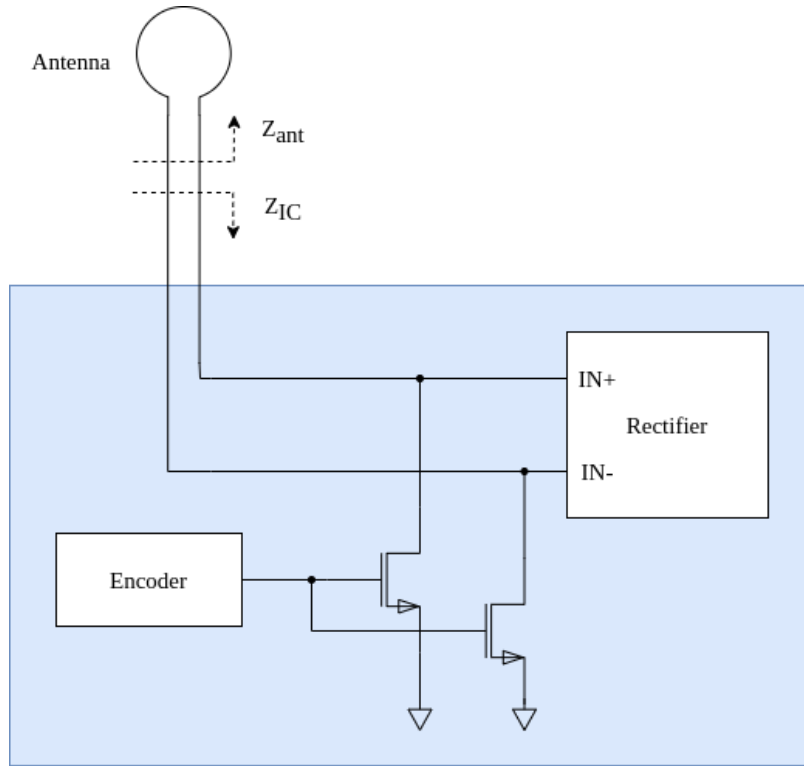


Figure 5-1: Backscatter circuit diagram

current between the two input terminals, which is not quite a short circuit as was assumed in the explanation, but sufficiently close that the reflection coefficient is nearly equal to 1. The gate voltage is controlled by the output bit of the digital encoder discussed in the previous chapter. This way, the digital encoder controls whether the system absorbs or reflects incident RF waves, and thus the backscattered RF signal that is transmitted.

Since the RF power collected by the transistors is very small, the leakage current of the transistors in the off state has a non-negligible impact on the performance of the rectifier. To minimize the leakage, HVT (high threshold voltage) transistors were used in the backscatter circuit. To decide the size of the transistors used for the backscatter circuit, the tradeoff between the ideality of the short circuit state, the parasitic capacitance added by the transistors in the open circuit state, and the energy required to switch the backscatter switches, was considered. To implement switches that are close to "ideal:" that is, switches that present as low of a resistance as possible, the width of the transistors should be maximized. However, because

the drain-source parasitic capacitance of a transistor scales linearly with its width, increasing the width is detrimental to the performance of the antenna-rectifier system. This is because a rectifier with a larger capacitance requires a physically smaller antenna in order to preserve conjugate impedance matching, which reduces the AC voltage received by the rectifier. Increasing the width of the transistors also increases their gate capacitance, increasing the energy required to switch the transistors, given by the following equation.

$$E = C_{gates} V_{DD}^2$$

The matching between the antenna and the integrated circuit is very sensitive to the impedances of each of the elements; in other words, a small change in the input impedance of the integrated circuit causes a significant disruption to the matching. See figure 5-2, which shows the reflection coefficient as a function of the relative change in the magnitude of the input impedance of the integrated circuit. Notice that only a small change in the impedance is required to cause a large change in the reflection coefficient. Therefore, the backscatter circuit needs to create only a small change in the input impedance of the integrated circuit between the "on" and "off" states. This means that increasing the effective width (number of fingers times width per finger) of the transistors used in the backscatter circuit gives a small benefit in the modulation factor, as shown in figure 5-3, which shows values from pre-layout simulations. Based on these results, HVT MOSFETs with 3 fingers were chosen for the backscatter circuit, since the marginal benefit of increasing the effective width beyond this point is small compared to the marginal cost of increased switching energy.

The post-layout simulated input impedances of the integrated circuit in both the off and on states are shown in table 5.1. It can be seen that the reflection coefficient is close to 1 in the "on" state, meaning that almost all incident RF power will be reflected. In the "off" state, the reflection coefficient shows that some of the power will be absorbed by the integrated circuit. The difference in the reflected power can

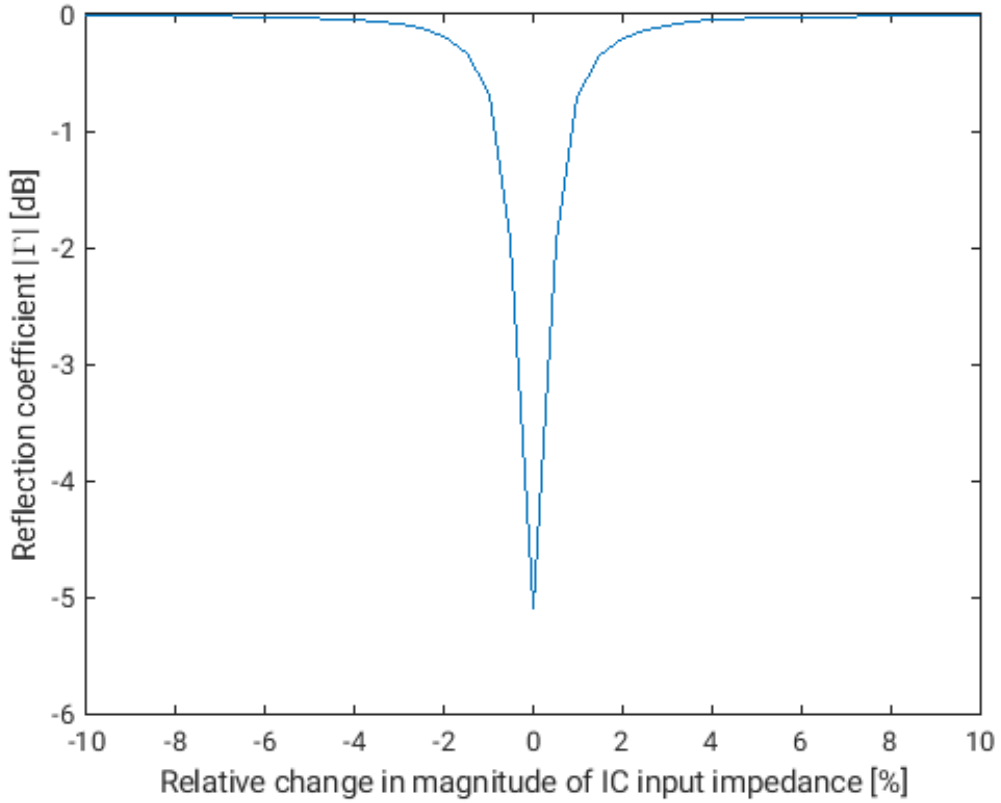


Figure 5-2: Reflection coefficient vs. change in input impedance

be measured by the receiving station and converted to a stream of bits, which is then decoded to extract and verify the data.

State	Off	On
Impedance Z_{IC}	$0.5 - 261.6j$	$88.0 - 226j$
Reflection coefficient Γ	$-0.245 + 0.332j$	$0.981 + 0.008j$
Magnitude of ref. coefficient $ \Gamma $	-7.7 dB	-0.2 dB
Modulation factor M	0.402	

Table 5.1: Simulated impedance and reflection coefficient of integrated circuit in on and off states (assuming $Z_{ant} = 1 + 262j$)

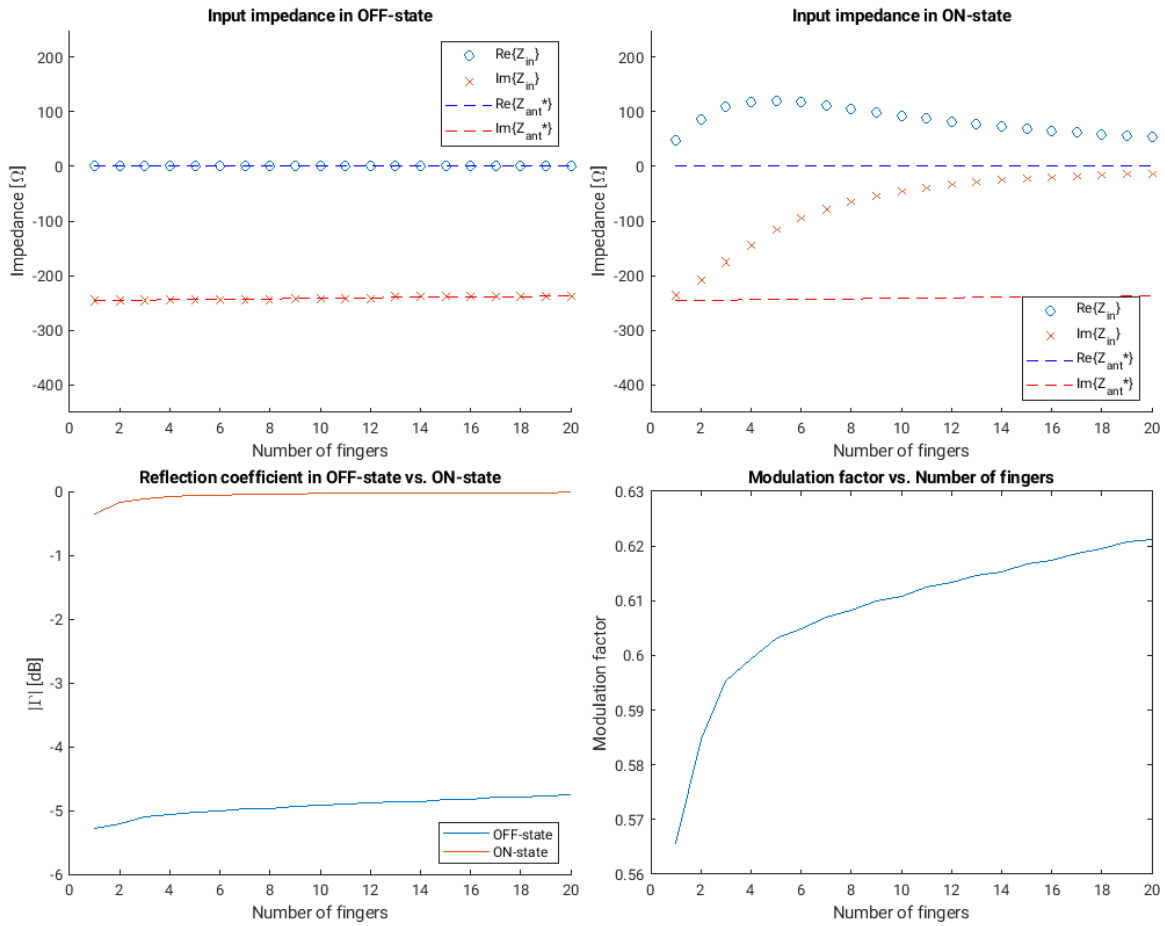


Figure 5-3: Input impedance, reflection coefficient, modulation factor vs. number of fingers (assuming $\text{Im}\{Z_{ant}\} = -\text{Im}\{Z_{IC,off}\}$)

Chapter 6

Conclusion and Future Work

This work presents a data transmission circuit that has been integrated into the integrated circuit for an RF energy harvesting IoT node. By utilizing power-minimizing techniques such as maximizing threshold voltage, minimizing flip-flop count, and block reuse, a circuit with sufficiently low power consumption such that it can be powered by weak wireless RF signals was implemented. This chip will be integrated onto a small PCB with an antenna to realize a self-powering IoT node, which will serve as a proof of concept for future self-powering IoT nodes.

Future generations of IoT nodes which build upon this work may include additional features, making them more practical devices. One potential feature is two-way communication. Being able to send commands, as well as power, to IoT nodes scattered in a large area would allow for more flexibility in the use of such devices. For example, this would allow the base station to command specific IoT nodes to transmit data, which may be useful in a situation where there is interest in the data being collected by a subset of IoT nodes within the range of the base station.

Another feature is the integration of environmental sensing into the IoT node. As environmental sensing is one of the most common applications of IoT, this feature would greatly extend the reach of many IoT applications. Since the world is analog, most environmental sensing electronics require an analog to digital converter (ADC). Therefore, a key challenge of implementing this feature is integrating an energy-efficient ADC. Fortunately, energy-efficient ADCs are a widely studied research topic;

recent works such as [14] could serve as the basis for the ADC of a future energy harvesting IoT node.

A key limitation of the presented IoT node is that it has no ability to tune its resonant frequency, meaning that it can only harvest energy from signals within a very narrow range of frequencies. Worse, due to variations in manufacturing, the resonant frequency may vary between devices. An IoT node with the ability to tune its resonant frequency in a similar manner to [7] would allow it to harvest energy from a wider range of frequencies as well as eliminate the dependence of performance on device variations.

Appendix A

Code Appendix

A.1 Digital Block RTL

A.1.1 zed_top.v

The top-level state machine described in 3.2.2, as well as the connections between the lower-level blocks, are implemented by this Verilog file. ("ZED" stands for **Z**ero **E**nergy **D**evelopments.)

```
1 'timescale 1ns / 1ps
2 module zed_top
3 (
4     input wire CLK,
5     input wire RESET,
6
7     input wire [7:0] DATA_IN,
8     input wire START,
9
10    output wire PHASE_DEBUG,
11    output wire [2:0] STATE_DEBUG,
12    output wire TBT_ENCODER_DATA_OUT_DEBUG,
13
14    output reg DATA_OUT
15 );
16
```

```

17 parameter ID = 8'hE4;
18
19 localparam IDLE = 0, SEND_PREAMBLE = 1, SEND_ID = 2, SEND_DATA =
    3, SEND_CRC = 4;
20
21 reg [2:0] state;
22 reg start_reg;
23
24 reg phase;
25 reg phase_1d;
26 wire data_out_1p;
27
28 wire preamble_start;
29 wire preamble_data_out;
30 wire preamble_done;
31 wire preamble_done_1p;
32
33 preamble preamble (.clk(CLK), .phase(phase), .reset(RESET), .start
    (preamble_start), .data_out(preamble_data_out), .done(
    preamble_done), .done_1p(preamble_done_1p));
34
35 wire [7:0] tbt_encoder_data_in;
36 wire tbt_encoder_start;
37 wire tbt_encoder_data_out;
38 wire tbt_encoder_done;
39 wire tbt_encoder_done_1p;
40
41 tbt_encoder tbt_encoder (.clk(CLK), .phase(phase), .reset(RESET),
    .data_in(tbt_encoder_data_in), .start(tbt_encoder_start), .
    data_out(tbt_encoder_data_out), .done(tbt_encoder_done), .done_1p
    (tbt_encoder_done_1p));
42
43 wire crc_reg_start;
44 wire [7:0] crc_reg_data_out;
45 wire [7:0] crc_reg_crc_out;
46 wire crc_reg_crc_valid;

```

```

47
48 crc_reg #(.CRC_POLY(8'h07), .INPUT_REFLECTED(1), .OUTPUT_REFLECTED
    (1), .CRC_REG_INITIAL(8'hFF), .CRC_REG_XOR_OUT(8'hFF)) crc_reg (.
    clk(CLK), .reset(RESET), .id_in(ID), .data_in(DATA_IN), .start(
    crc_reg_start), .data_out(crc_reg_data_out), .crc_out(
    crc_reg_crc_out), .crc_valid(crc_reg_crc_valid));
49
50 assign crc_reg_start = (state == IDLE) & start_reg;
51 assign preamble_start = (state == IDLE) & start_reg;
52 assign tbt_encoder_start = (state == SEND_PREAMBLE) &
    preamble_done_1p | ((state == SEND_ID | state == SEND_DATA) &
    tbt_encoder_done_1p);
53
54 assign tbt_encoder_data_in = (state == SEND_ID) ? ID : (state ==
    SEND_DATA) ? crc_reg_data_out : (state == SEND_CRC) ?
    crc_reg_crc_out : 8'b0;
55
56 assign data_out_1p = (state == SEND_PREAMBLE) ? preamble_data_out
    : ((state == SEND_ID | state == SEND_DATA | state == SEND_CRC) ?
    (phase_1d ^ tbt_encoder_data_out) : 1'b0);
57
58 assign PHASE_DEBUG = phase;
59 assign STATE_DEBUG = state;
60 assign TBT_ENCODER_DATA_OUT_DEBUG = tbt_encoder_data_out;
61
62 always @(posedge CLK or negedge RESET) begin
63     if (RESET == 1'b0) begin
64         phase <= 1'b0;
65         phase_1d <= 1'b1;
66         DATA_OUT <= 1'b0;
67     end else begin
68         phase <= ~phase;
69         phase_1d <= phase;
70         DATA_OUT <= data_out_1p;
71     end
72 end

```

```

73
74 always @(posedge CLK or negedge RESET) begin
75     if (RESET == 1'b0) begin
76         state <= IDLE;
77         start_reg <= 1'b0;
78     end else if (phase == 1'b1) begin
79         start_reg <= START;
80         case (state)
81             IDLE: begin
82                 if (start_reg) begin
83                     state <= SEND_PREAMBLE;
84                 end
85             end
86             SEND_PREAMBLE: begin
87                 if (preamble_done_1p) begin
88                     state <= SEND_ID;
89                 end
90             end
91             SEND_ID: begin
92                 if (tbt_encoder_done_1p) begin
93                     state <= SEND_DATA;
94                 end
95             end
96             SEND_DATA: begin
97                 if (tbt_encoder_done_1p & crc_reg_crc_valid) begin
98                     state <= SEND_CRC;
99                 end
100            end
101            SEND_CRC: begin
102                if (tbt_encoder_done_1p) begin
103                    state <= IDLE;
104                end
105            end
106            default: begin
107                state <= IDLE;
108            end

```

```

109     endcase
110   end
111 end
112
113 endmodule

```

A.1.2 crc_reg.v

This Verilog file implements the CRC checksum calculator described in 3.2.3.

```

1  `timescale 1ns / 1ps
2
3  module crc_reg
4    #(
5      parameter CRC_POLY = 8'h07,
6      parameter INPUT_REFLECTED = 1,
7      parameter OUTPUT_REFLECTED = 1,
8      parameter CRC_REG_INITIAL = 8'hFF,
9      parameter CRC_REG_XOR_OUT = 8'hFF
10   )
11   (
12     input wire clk,
13     input wire reset,
14
15     input wire [7:0] id_in,
16     input wire [7:0] data_in,
17     input wire start,
18
19     output wire [7:0] data_out,
20     output wire [7:0] crc_out,
21     output wire crc_valid
22   );
23
24
25   localparam IDLE = 0, CRC_ENCODE = 1;
26

```

```

27  reg state;
28  reg [4:0] counter;
29  reg [7:0] data_reg;
30  reg [7:0] crc_reg;
31
32  wire [15:0] crc_reg_shift_in;
33  wire [7:0] crc_out_prexor;
34
35  genvar i;
36
37  generate
38      if (INPUT_REFLECTED) begin
39          for (i = 0; i < 8; i = i + 1) begin
40              assign crc_reg_shift_in[i+8] = id_in[7-i];
41              assign crc_reg_shift_in[i] = data_in[7-i];
42          end
43      end else begin
44          assign crc_reg_shift_in[15:8] = id_in;
45          assign crc_reg_shift_in[7:0] = data_in;
46      end
47
48      if (OUTPUT_REFLECTED) begin
49          for (i = 0; i < 8; i = i + 1) begin
50              assign crc_out_prexor[i] = crc_reg[7-i];
51          end
52      end else begin
53          assign crc_out_prexor = crc_reg;
54      end
55  endgenerate
56
57  assign crc_out = crc_out_prexor ^ CRC_REG_XOR_OUT;
58
59  always @(posedge clk or negedge reset) begin
60      if (reset == 0) begin
61          state <= IDLE;
62          counter <= 4'b1111;

```

```

63     end else begin
64         case (state)
65             IDLE: begin
66                 if (start) begin
67                     state <= CRC_ENCODE;
68                     data_reg <= data_in;
69                     crc_reg <= CRC_REG_INITIAL;
70                     counter <= 4'b1111;
71                 end
72             end
73             CRC_ENCODE: begin
74                 counter <= counter - 1'b1;
75                 crc_reg <= {crc_reg[6:0], 1'b0} ^ ((crc_reg[7] ^
76                 crc_reg_shift_in[counter]) ? CRC_POLY : 8'b0);
77                 if (counter == 4'b0) begin
78                     state <= IDLE;
79                 end
80             end
81             default: begin
82                 state <= IDLE;
83                 counter <= 4'b1111;
84             end
85         endcase
86     end
87
88     assign data_out = data_reg;
89     assign crc_valid = (state == IDLE);
90
91 endmodule

```

A.1.3 preamble.v

This Verilog file implements a block which outputs the preamble, which is the 11-bit Barker code.

```

1  'timescale 1ns / 1ps
2  module preamble
3      #(
4          parameter integer SEQUENCE = 11'b11100010010,
5          parameter SEQUENCE_LENGTH = 11
6      )
7      (
8          input wire clk,
9          input wire phase,
10         input wire reset,
11
12         input wire start,
13
14         output wire data_out,
15         output wire done,
16         output wire done_1p
17     );
18
19     localparam IDLE = 0, DATA_OUT = 1;
20
21     reg [${clog2(SEQUENCE_LENGTH)-1:0}] counter;
22     reg state;
23
24     always @(posedge clk or negedge reset) begin
25         if (reset == 1'b0) begin
26             state <= IDLE;
27             counter <= 0;
28         end else if (phase == 1'b1) begin
29             case (state)
30                 IDLE: begin
31                     if (start) begin
32                         state <= DATA_OUT;
33                         counter <= 0;
34                     end
35                 end
36                 DATA_OUT: begin

```

```

37     if (counter == SEQUENCE_LENGTH - 1) begin
38         state <= IDLE;
39         counter <= 0;
40     end else begin
41         counter <= counter + 1;
42     end
43 end
44 default: begin
45     state <= IDLE;
46     counter <= 0;
47 end
48 endcase
49 end
50 end
51
52 assign data_out = SEQUENCE[SEQUENCE_LENGTH - 1 - counter];
53 assign done = (state == IDLE);
54 assign done_1p = (state == DATA_OUT & counter == SEQUENCE_LENGTH -
55     1);
56 endmodule

```

A.1.4 tbt_encoder.v

This Verilog file implements the convolutional code encoder described in 3.2.4.

```

1  `timescale 1ns / 1ps
2  module tbt_encoder
3      (
4          input wire clk,
5          input wire phase,
6          input wire reset,
7
8          input wire [7:0] data_in,
9          input wire start,
10

```

```

11     output wire data_out ,
12     output wire done ,
13     output wire done_1p
14 );
15
16 // States
17 localparam IDLE = 0, DATA_ENCODE = 1;
18
19 // Registers
20 reg state;
21 reg [3:0] counter;
22
23 // Wires
24 //wire shift_reg_gclk;
25 wire [2:0] index_0;
26 wire [2:0] index_1;
27 wire [2:0] index_2;
28 wire [2:0] index_3;
29 wire data_sel;
30
31 always @(posedge clk or negedge reset) begin
32     if (reset == 1'b0) begin
33         state <= IDLE;
34         counter <= 0;
35     end else if (phase == 1'b1) begin
36         case (state)
37             IDLE: begin
38                 if (start) begin
39                     state <= DATA_ENCODE;
40                     counter <= 0;
41                 end
42             end
43             DATA_ENCODE: begin
44                 if (counter == 4'b1111 & !start) begin
45                     state <= IDLE;
46                 end

```

```

47         counter <= counter + 1;
48     end
49     default: begin
50         state <= IDLE;
51         counter <= 0;
52     end
53 endcase
54 end
55 end
56
57 assign index_0 = counter [3:1];
58 assign index_1 = counter [3:1] - 1;
59 assign index_2 = counter [3:1] - 2;
60 assign index_3 = counter [3:1] - 3;
61
62 assign data_0 = data_in[index_0];
63 assign data_1 = data_in[index_1];
64 assign data_2 = data_in[index_2];
65 assign data_3 = data_in[index_3];
66
67 assign data_sel = counter [0];
68
69 assign data_out = data_sel ? (data_0 ^ data_1) : (data_0 ^ data_2
    ^ data_3);
70
71 assign done = (state == IDLE);
72 assign done_1p = (state == DATA_ENCODE & counter == 4'b1111);
73
74 endmodule

```

A.2 Digital Block Verification Testbenches and Scripts

A.2.1 zed_top_comprehensive_tb.v

This Verilog file implements a testbench for the entire digital block. It sets the input data to every possible value, so that the output can be saved and verified by the verification script that follows.

```
1  `timescale 1us / 1ns
2
3  module zed_top_comprehensive_tb;
4
5      reg clk, reset, start;
6      reg [7:0] data_in;
7      wire data_out;
8
9      zed_top uut (.CLK(clk), .RESET(reset), .DATA_IN(data_in), .START(
        start), .DATA_OUT(data_out));
10
11
12     initial begin
13         $sdf_annotate("../synth2/core/current/out/zed_top_syn.sdf");
14         $sdf_annotate("../par2/core/current/out/zed_top_par.sdf");
15         $dumpfile("zed_top.vcd");
16         $dumpvars(0, zed_top_comprehensive_tb.uut);
17         $dumpall;
18     end
19
20     integer i;
21
22     always #30 clk = !clk;
23
24     initial begin
25         clk = 1;
26         reset = 0;
27         start = 0;
28         #60;
```

```

29     reset = 1;
30     #60;
31
32     for (i = 0; i < 256; i = i + 1) begin
33         data_in = i;
34         start = 1;
35         #120;
36         start = 0;
37         #12000;
38     end
39     $finish;
40 end
41 endmodule

```

A.2.2 verify_zed_top.m

This MATLAB script takes the input and output of the top level digital block and verifies that the output is as expected.

```

1 waveforms = readmatrix("zed_top.csv", "OutputType", "string");
2
3 data_in = waveforms(:,3); % Data in (00-FF)
4 data_out = waveforms(:,4); % Data out waveform
5
6 id_ref = [0 0 1 0 0 1 1 1]; % Tag ID
7
8
9 for i=0:255
10     i_hex = dec2hex(i, 2);
11     data_out_i = data_out(data_in == i_hex); % Select data out
12     % waveform where data in equals current iteration
13     data_out_i = data_out_i(1:2:end); % Every other bit
14     data_out_i = strjoin(data_out_i, "");
15
16     data_out_after_barker = extractAfter(data_out_i,
17     "1111110000001100001100"); % Select bits after Barker code

```

```

16 % Decode Manchester encoded signal
17 data_array = regexp(data_out_after_barker, '\w{1,2}', 'match');
18 bits_array = [];
19 for j=data_array
20     if (j == "01")
21         bits_array = [bits_array 1];
22     elseif (j == "10")
23         bits_array = [bits_array 0];
24     else
25         break;
26     end
27 end

28
29 % Decode tail-biting trellis encoded data: ID, data, CRC
30 id_i = tbt_decoder(bits_array(1:16));
31 data_i = tbt_decoder(bits_array(17:32));
32 data_i_dec = bin2dec(num2str(data_i(end:-1:1)'));
33 crc_i = tbt_decoder(bits_array(33:48));
34 crc_i_ref = calculate_crc([id_i; data_i]');
35
36
37 % Assertions for verification
38 assert (isequal(id_i, id_ref'));
39 assert (isequal(data_i_dec, i));
40 assert (isequal(crc_i, crc_i_ref));
41
42 end

43
44 function data = tbt_decoder(encoded_data)
45
46     K = 8;
47     m = 3;
48     constraint_length = m + 1;
49     tblen = K;
50
51     generator_poly = {'x3+x1+1', 'x3+x2'};

```

```

52     trellis = poly2trellis(constraint_length, generator_poly);
53
54     y = 1 - 2 * encoded_data';
55     y = kron(ones(3,1),y);
56     w = vitdec(y, trellis, tblen, 'trunc', 'unquant');
57     data = w(end-K+1:end);
58
59 end
60
61 function checksum = calculate_crc(data)
62     crc8 = comm.CRCGenerator('Polynomial','z^8 + z^2 + z + 1', ...
63         'InitialConditions',1,'DirectMethod',true,'FinalXOR',1);
64     codeword = crc8(data);
65     checksum = codeword(end-8+1:end)';
66 end

```

A.2.3 crc_vecgen.py

This Python script calculates a CRC checksum for every possible input value and saves the input and output into a file readable by a Verilog testbench.

```

1 from crc import CrcCalculator, Configuration
2
3 width = 8
4 poly = 0x07
5 init_value = 0xFF
6 final_xor_value = 0xFF
7 reverse_input = True
8 reverse_output = True
9
10 configuration = Configuration(width, poly, init_value,
11     final_xor_value, reverse_input, reverse_output)
12
13 use_table = True
14
15 crc_calculator = CrcCalculator(configuration, use_table)
16

```

```

15 def to_hex(x):
16     return "{:02x}".format(x)
17
18
19 lines = []
20
21 for i in range(0, 2**8):
22     for j in range(0, 2**8):
23         data = [i, j]
24         crc = crc_calculator.calculate_checksum(bytes(data))
25         lines.append(to_hex(i) + to_hex(j) + to_hex(crc) + "\n")
26
27 for i in range(0, 4):
28     with open("crc_vector_" + str(i) + ".mem", "w") as f:
29         f.writelines(lines[i*2**14:(i+1)*2**14])

```

A.2.4 crc_reg_verification_tb.v

This Verilog file implements a testbench for the CRC calculator. It confirms that the CRC calculator module generates the same outputs as the Python script.

```

1 module crc_reg_verification_tb;
2
3     reg [23:0] vector [0:16383];
4     reg [13:0] index;
5     wire [7:0] crc_out_expected;
6
7     reg clk;
8     reg reset;
9     wire [7:0] id_in;
10    wire [7:0] data_in;
11    reg start;
12    wire [7:0] data_out;
13    wire [7:0] crc_out;
14    wire crc_valid;
15

```

```

16
17 crc_reg uut (.clk(clk), .reset(reset), .id_in(id_in), .data_in(
    data_in), .start(start),
18 .data_out(data_out), .crc_out(crc_out), .crc_valid(crc_valid));
19
20 always #5 clk = !clk;
21
22 assign id_in = vector[index][23:16];
23 assign data_in = vector[index][15:8];
24 assign crc_out_expected = vector[index][7:0];
25
26 integer h;
27 integer i;
28
29 initial begin
30     clk = 1;
31     reset = 0;
32     start = 0;
33     #10;
34     reset = 1;
35     #10;
36     for (h = 0; h < 4; h = h + 1) begin
37         $readmemh($sformatf("/homes/jungj/digital/zed-flow/zed-rtl/
verification/crc_vector_%0d.mem", h), vector);
38         for (i = 0; i < 16384; i = i + 1) begin
39             index = i;
40             #10;
41             start = 1;
42             #10;
43             start = 0;
44             while (!crc_valid) begin
45                 #10;
46             end
47             if (crc_valid & crc_out != crc_out_expected) begin
48                 $display("Test failed");
49                 $display(h);

```

```

50         $display(i);
51         //$finish;
52     end
53     #10;
54 end
55 end
56 $finish;
57 end
58
59 endmodule

```

A.2.5 tbt.py

This Python script encodes every possible 8-bit data input into 16-bit outputs and saves the input and output into a file readable by a Verilog testbench.

```

1 import numpy as np
2
3 input_vectors = np.zeros([256,8])
4
5 for u7 in [0,1]:
6     for u6 in [0,1]:
7         for u5 in [0,1]:
8             for u4 in [0,1]:
9                 for u3 in [0,1]:
10                    for u2 in [0,1]:
11                        for u1 in [0,1]:
12                            for u0 in [0,1]:
13                                input_vectors[u0+u1*2+u2*4+u3*8+u4
14                                    *16+u5*32+u6*64+u7*128,:] = [u0, u1, u2, u3, u4, u5, u6, u7]
15 #print(input_vectors)
16
17 matrix = np.array(
18     [
19         [1,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0],

```

```

20         [0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0],
21         [0,0,0,0,1,1,0,1,1,0,1,0,0,0,0,0],
22         [0,0,0,0,0,0,1,1,0,1,1,0,1,0,0,0],
23         [0,0,0,0,0,0,0,0,1,1,0,1,1,0,1,0],
24         [1,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0],
25         [1,0,1,0,0,0,0,0,0,0,0,0,1,1,0,1],
26         [0,1,1,0,1,0,0,0,0,0,0,0,0,0,1,1]
27     ])
28
29 encoded_result = np.flip(np.mod(np.matmul(input_vectors, matrix), 2)
30     , axis=1)
31 concatenated_vector = np.concatenate((np.flip(input_vectors, axis=1)
32     , encoded_result), axis=1)
33
34 with open("tbt_vector.mem", "w") as f:
35     for line in concatenated_vector:
36         f.write("".join([str(int(element)) for element in line]))
37         f.write("\n")

```

A.2.6 tbt_encoder_verification_tb.v

This Verilog file implements a testbench for the convolutional code (tail-biting trellis) encoder. It verifies that the module generates the same outputs as the Python script.

```

1  `timescale 1ns / 1ps
2
3  module tbt_encoder_verification_tb;
4
5      reg [23:0] vector [0:255];
6      reg [7:0] index;
7      wire [15:0] data_out_expected;
8
9      reg clk;
10     reg reset;

```

```

11  reg phase;
12
13  wire [7:0] data_in;
14  reg start;
15  wire data_out;
16  wire done;
17
18  reg [15:0] data_out_reg;
19
20  tbt_encoder uut
21  (.clk(clk), .reset(reset), .data_in(data_in),
22  .start(start), .phase(phase),
23  .data_out(data_out), .done(done));
24
25  assign data_in = vector[index][23:16];
26  assign data_out_expected = vector[index][15:0];
27
28  integer i;
29  integer reg_index;
30
31  always #2.5 clk = !clk;
32  always #5 phase = !phase;
33  initial begin
34      $readmemb("/homes/jungj/digital/zed-flow/zed-rtl/verification/
35      tbt_vector.mem", vector);
36      clk = 1;
37      reset = 0;
38      phase = 1;
39      #10;
40      reset = 1;
41      for (i = 0; i < 256; i = i + 1) begin
42          index = i;
43          start = 1;
44          #10;
45          start = 0;
46          reg_index = 0;

```

```
46     data_out_reg = 16'b0;
47     while (!done) begin
48         data_out_reg[reg_index] = data_out;
49         #10;
50         reg_index = reg_index + 1;
51     end
52     if (data_out_reg != data_out_expected) begin
53         $display("Test failed");
54         $finish;
55     end
56 end
57 $finish;
58 end
59
60
61 endmodule
```


Bibliography

- [1] D. Puccinelli and M. Haenggi, “Wireless sensor networks: applications and challenges of ubiquitous sensing,” *IEEE Circuits and Systems Magazine*, vol. 5, no. 3, pp. 19–31, 2005.
- [2] F.-T. Lin, Y.-C. Kuo, J.-C. Hsieh, H.-Y. Tsai, Y.-T. Liao, and H.-C. Lee, “A self-powering wireless environment monitoring system using soil energy,” *IEEE Sensors Journal*, vol. 15, pp. 3751–3758, July 2015.
- [3] F. Parás-Hernández, A. Fabián-Mijangos, M. Cardona-Castro, and J. Alvarez-Quintana, “Enhanced performance nanostructured thermoelectric converter for self-powering health sensors,” *Nano Energy*, vol. 74, p. 104854, 2020.
- [4] V. Ostasevicius, V. Jurenas, V. Markevicius, R. Gaidys, M. Zily, M. Cepenas, and L. Kizauskiene, “Self-powering wireless devices for cloud manufacturing applications,” *The International Journal of Advanced Manufacturing Technology*, vol. 83, no. 9, pp. 1937–1950, 2016.
- [5] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, “Ambient backscatter: Wireless communication out of thin air,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, p. 39–50, aug 2013.
- [6] M. Stoopman, S. Keyrouz, H. J. Visser, K. Philips, and W. A. Serdijn, “Co-design of a cmos rectifier and small loop antenna for highly sensitive rf energy harvesters,” *IEEE Journal of Solid-State Circuits*, vol. 49, no. 3, pp. 622–634, 2014.
- [7] M. R. Abdelhamid, R. Chen, J. Cho, A. P. Chandrakasan, and F. Adib, “Self-reconfigurable micro-implants for cross-tissue wireless and batteryless connectivity,” *ACM MobiCom 2020*, 2020.
- [8] A. Nourbakhsh, A. Zubair, S. Joglekar, M. Dresselhaus, and T. Palacios, “Sub-threshold swing improvement in mos2 transistors by the negative-capacitance effect in a ferroelectric al-doped-hfo2/hfo2 gate dielectric stack,” *Nanoscale*, vol. 9, pp. 6122–6127, 2017.
- [9] P. Stahl, J. Anderson, and R. Johannesson, “Optimal and near-optimal encoders for short and moderate-length tail-biting trellises,” *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2562–2571, 1999.

- [10] M. M. Lopez, "Private communication." Telefonaktiebolaget LM Ericsson.
- [11] "Ieee standard for ethernet," *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–4017, 2016.
- [12] "Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - redline," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016) - Redline*, p. 5363, 2021.
- [13] J. D. Griffin and G. D. Durgin, "Complete link budgets for backscatter-radio and rfid systems," *IEEE Antennas and Propagation Magazine*, vol. 51, no. 2, pp. 11–25, 2009.
- [14] Y. Song, Z. Xue, Y. Xie, S. Fan, and L. Geng, "A 0.6-v 10-bit 200-ks/s fully differential sar adc with incremental converting algorithm for energy efficient applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 4, pp. 449–458, 2016.