MENG 471a Final Report

Special Projects I Dec. 20, 2017

Applications in RFID Technology

Tim Foldy-Porto

Advisor: Dr. Lawrence Wilen

Acknowledgements

We would like to acknowledge the Yale School of Engineering and Applied Science dean's office for their generous financial support.

Table of contents

1. Introduction

Radio-Frequency Identification (RFID) technology has proliferated in the past twenty years: it is currently employed in a variety of industries, ranging from commerce, to transportation logistics, to infrastructure management and protection. The technology uses small, passive tags to store information, generally an identification code, which can be accessed by a reader. There are currently many companies which offer RFID products, for various applications: the London transportation authority uses RFID Oyster Cards to calculate fares for its passengers, who can swipe into any form of public transportation; Zebra Technologies, in a partnership with the NFL, uses RFID to track players movements on the field, allowing for real-time, in-game statistics; HID Global produces high quality identification cards, including the ones distributed by universities, that can be used for physical access control.

Passive, radio-based technology dates back to World War II, with the invention of a covert listening device called “The Thing” by Leon Theremin for the Soviet Union. Although not exactly the same as modern day RFID, “The Thing” employed backscatter modulation of an incident radio wave to transmit audio signals to a listener, which is the essential operating principle of RFID technology today. At the same time, the German army used a crude version backscatter modulation to identify whether incoming aircraft on their radar were friend or foe. German pilots found that if they did a barrel roll while flying towards a radar station, they would cause a disturbance in the incident radio wave that would be identifiable to radar operators.

Today RFID is ubiquitous, appearing in a variety of places. There has been a great amount of academic research done into the properties of this technology and its potential applications in the future. Private corporations are also very interested in RFID, and have used their own R&D labs to develop numerous advances in this technology, including many (albeit proprietary) revolutionary collision detection processes, which allow a single reader to process multiple RFID tags in its vicinity simultaneously. In addition, much corporate research is being done into the security and encryption algorithms behind RFID tags, including by the company HID Global, the manufacturer of Yale’s own ID cards.

2. Background

Due to RFID's widespread usage as a physical access device, there has been a great amount of interest in the system from the electronics and security hobbyist community. There has been a significant amount of research conducted and published online about hacking RFID systems. Several people have successfully been able to clone an HID proximity card, as well as exploit other features in RFID based security systems. Furthermore, there are many ASK readers commercially available for very low prices on online retailers. These devices are capable of reading and duplicating ASK modulated RFID cards, which is a threat to many basic security systems.

The cards upon which we conducted our research were FSK modulated, a term we will define in depth in subsequent sections, meaning they could not be read or manipulated using a cheap ASK reader. While we knew this to be true from theoretical presumptions, previous experimentally research we had conducted had indicated that Yale ID cards were definitely not ASK compatible, which prompted us to search for alternative methods of interfacing with the Yale RFID cards, leading to this project on RFID and its potential applications.

3. Design and approach

3.1. Card reader

The first part of our project consisted of designing and building a device that could read the data from an FSK encoded RFID card. For convenience, we tested our reader using a standard issue Yale ID card, which we knew to operate at a frequency of 125 kHz based on documentation from the card manufacturer. Additionally, from the international standard for contactless integrated circuit cards (ISO/IEC 14443-4), we deduced that Yale ID cards used FSK modulation and were encoded by the processes of Manchester encoding and decimation, which will be discussed later in the results section.

The primary analog section of our reader was built following documentation of a similar project found online, titled “DIY FSK RFID Reader.” (cite) The goal of that project was to “present a simple solution for reading FSK tags which addresses the following shortcomings: make it robust and reliable for real-world environments, base it on the Arduino, and build the RFID reader ourselves using a few simple low-cost parts,” which aligned quite nicely with the goals of our own project. On the following page is schematic of the circuit that we built.

<u>Overview of the circuit</u>

The Arduino produces a 125kHz square-wave, which drives the LC tank (tuned to resonate at 125kHz) consisting of L1 and C1. When a RFID card comes close to the tank, they inductively couple and the card draws a small amount of power from the reader, causing a drop in amplitude of the 125kHz sine wave present at the junction of L1 and C1. The amplitude of this sine wave is peak-detected by the diode, C2, and R1. C3 then decouples any DC bias from the AC signal (which is still following the peak of the 125kHz sine wave). R2 and C4 serve as a low-pass filter to remove any noise. The signal is then amplified by OP1 and placed on top of a 2.5V bias from OP2, which in turn feeds into a comparator (OP4) which converts the signal to a square-wave.

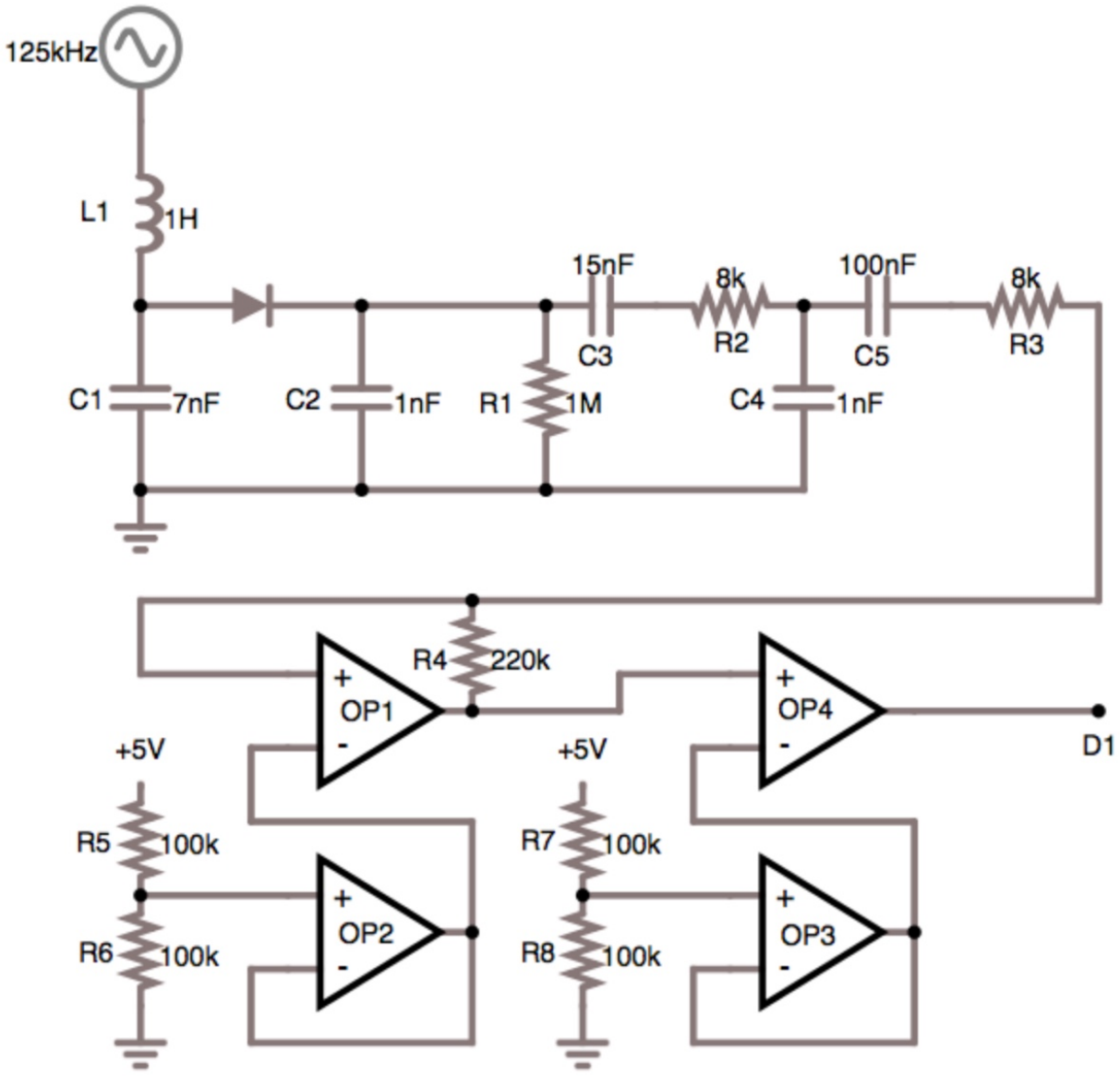
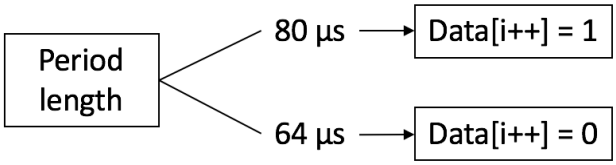


Fig 3.1. Analog RFID FSK reader circuit. This circuit communicates with a given FSK RFID tag and turns its data into a binary signal, which is then decoded into readable data by an Arduino.

Physically, this square-wave represents the amplitude of the 125kHz sine wave that is present between L1 and C1, which can be modulated by a nearby RFID tag. The square-wave present at D1 is then fed into an Arduino, which decodes the FSK modulation, Manchester encoding, and decimation in order to generate numerically-representable binary data.

Overview of the code

FSK modulation works by encoding 0s and 1s as different frequencies, so the decoding process consists of determining the frequency of a given period in the incoming square-wave (from D1 on the analog circuit) and categorizing it as either a 0 or a 1. The first part of the our code for the Arduino established a timer which we used to time the length of an incoming pulse. For each period of the square-wave, we took two measurements: one of the low pulse and one of the high pulse. A form of error detection was inherent in this method of frequency detection, since we could expect the low pulse and the high pulse to have the same length for a given period. A block diagram of this section of code is as follows:



As the program detected the length of the incoming pulses, it stored the associated binary value in an array called Data, as shown above. This process fully decoded the FSK encoding, but it was clear from the length of Data that there were other encoding processes that need decoding. Documentation from the manufacturer of Yale IDs indicated that the length of a given ID was 44 bits, yet the length of our array, Data, exceeded 500 bits. It was quite clear from a visual inspection that the data was encoded using decimation. The program to decode this was simple: check for large blocks of single repeated value, then replace it with a single instance of that value. For example, if a section of Data looked like this:

00000111111000000000000111111

Our algorithm replaced it with:

01001

The algorithm, in this case, determined a large block to be five or six repeated values, which is why the large chunk of 0s in the middle of the above string is replaced not by one, but by two instances of 0.

While the data was now in a more manageable state, it was still twice as long as expected, indicating one final encoding process. We again used ISO/IEC 14443 along with a visual inspection of the data to determine that it was encoded by Manchester encoding. The program to decode this process was also quite simple: examine blocks of two bits at a time; if a given block went low-high (01), we replaced it with a single 1; if the block went high-low (10), we replaced it with a single 0. For example, if a section of Data looked like this:

010101101010

Our algorithm replaced it with:

111000

This process reduced the string down to the appropriate size of 44 bits, thus indicating a valid ID code.

3.2. Card writer

After building a working card reader, it was only natural to build an RFID card writer, or a device that mimics an RFID tag. As a test of functionality, we chose to use the card reader that we had built: if our RFID writer could "trick" the reader into outputting the same code as a given RFID tag, then the device would be successful. This process proved to be quite trivial in terms of both hardware and software, as we were essentially creating an inverse device to the card reader that we had already made, but it did reveal several interesting problems with the signal generation ability of microcontrollers, which became the focus of our research for the second half of this project.

The hardware and software for this project were designed solely by the members of this project, but it is important to note that very similar devices can be found documented online. The publication that we based our reader circuit on included a small section on an inverse device, which is almost identical to the one that we created. Additionally, hobbyists Asher Glick and Micah Dowty have published similar documentation and schematics for similar projects on their personal websites and Githubs (cite).

Overview of the circuit

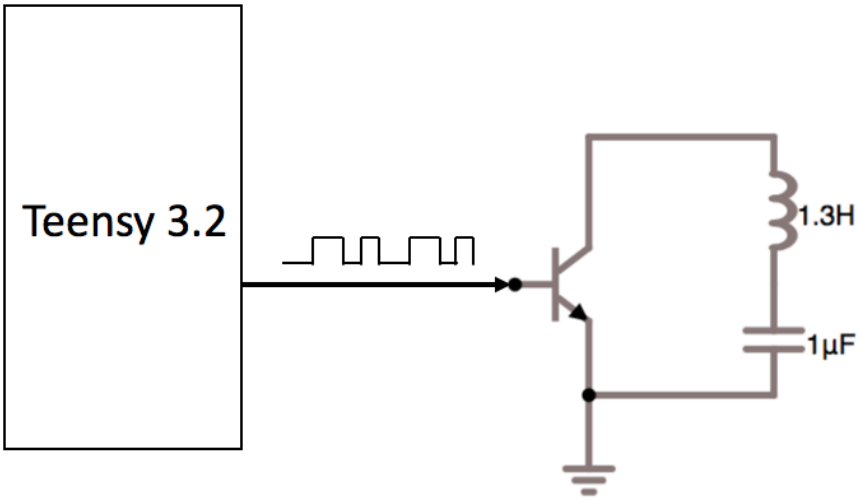


Fig 3.2. Circuit schematic for our FSK RFID writer.

The circuit for the writer is incredibly simple, which is clear from figure 3.2 above. Essentially, the Teensy 3.2 microcontroller produces an FSK digital signal, which controls the current flowing through the transistor. When the signal is high, the transistor is saturated, thus allowing current to flow and closing the LC tank, which is tuned to resonate at 125 kHz. When the circuit is closed and held in proximity of the reader, the LC tank is driven at 125 kHz by the reader, thus drawing power and modulating the amplitude of the driving signal.

Overview of the code

The purpose of the code in the writer was to generate the FSK modulation signal pictured above. The basis for this signal was a 44 bit code (in practice, one that we had discovered by scanning a real RFID card with our reader). Our program took each bit, Manchester encoded it, and decimated it, creating a final string of around 600 bits. It then used this sequence to generate a digital square-wave, where a 0 corresponded to a period of 64 μs and a 1 corresponded to a period of 80 μs. While in theory this code is very simple, the architecture of the Teensy microcontroller made it difficult to ensure that the different periods were in phase; i.e. that the generation of one period began exactly when the previous period ended.

3.3. FSK Signal generation through delayed pulses

Due to the numerous issues with generating a precise FSK pulse based on a given ID code using an Arduino microcontroller (discussed in the results section), we decided to build designated hardware for the job. We intended our first device to operate in cooperation with the Arduino, giving it more time to process the data and hence removing any timing issues.

Overview of the circuit

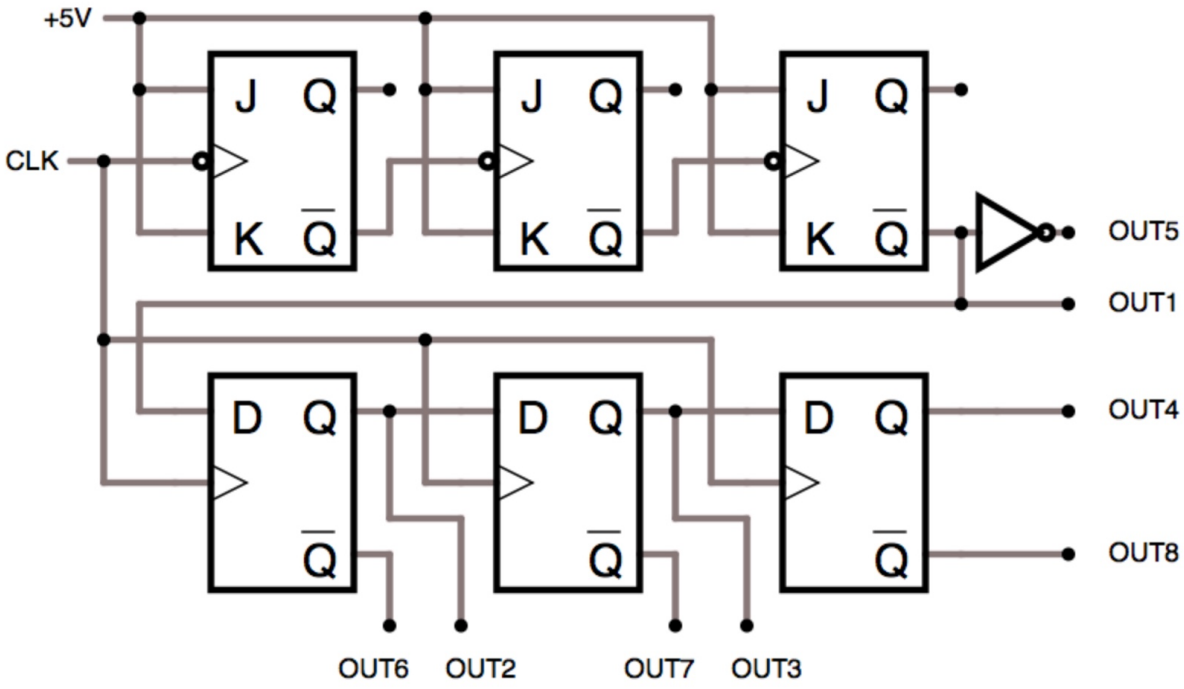


Fig 3.3. Pulse delay circuit consisting of six flip-flops.

In short, this circuit (Fig. 3.3) takes a clock pulse, divides it by eight, and delays it seven times to produce eight separate square-waves, each delayed by an eighth of a period. For example, suppose the clock is oscillating at a frequency of 125 kHz (8 μs period). The top three JK flip-flops take this pulse and divide it in half, three times sequentially, thus producing a 15.6 kHz (64 μs period) square-wave on OUT1. The lower three flip-flops are D-type, meaning whatever data is present on D is shifted to Q at the leading edge of the clock pulse. Since the clock pulses every 8 μs, the signal on OUT2 is just the signal on OUT1 delayed by 8 μs. Similarly, OUT3 is delayed 8 μs from OUT2, and OUT4 another 8 μs from OUT3. To delay another eighth-period (8 μs), we

select OUT5, which is exactly one half-period out of phase with OUT1. Similarly, OUT6, OUT7, and OUT8 are the complement of OUT2, OUT3, and OUT4, filling in the remainder of the digital delay cycle. Visually represented, outputs one through eight look as follows:

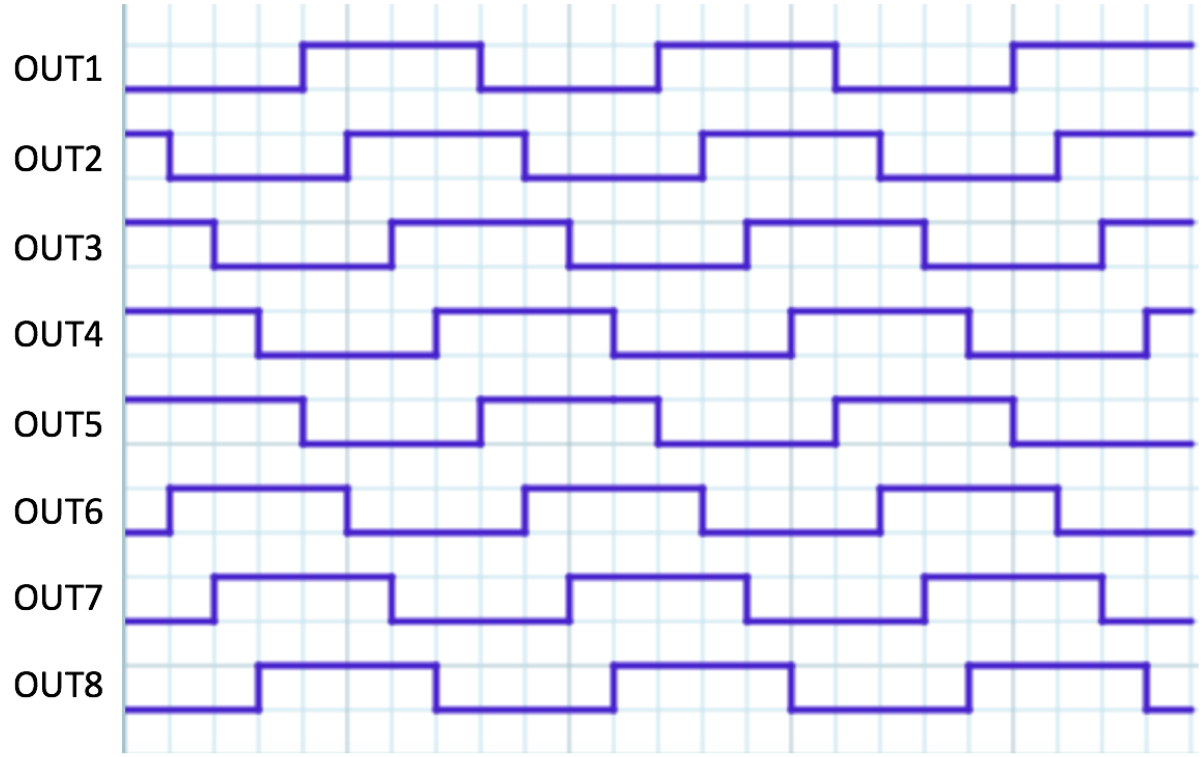


Fig 3.4. Graphical representation of the outputs of our pulse delay circuit vs. time.

We can see from figure 3.4 that if we first select OUT1 as our output, our device will generate a square-wave with a period of 64 μs. However, if at some point during the low part of the cycle we switched from OUT1 to OUT2, the output of our device would see a low period of 40 μs due to the 8 μs delay on OUT2. In the same way, if we then switched from OUT2 to OUT3 during the high portion of the signal, our output would see a high period of 40 μs. Thus by switching twice between three 64 μs signals (15.6 kHz), we have generated a single 80 μs period (12.5 kHz).

This is particularly useful in the case of generating an FSK signal that switches between 15.6 kHz and 12.5 kHz, as was necessary for our RFID writer. With this circuit in place, it was then up to the Arduino to decide when to switch output signals. This was accomplished by running the eight outputs into a 3-bit multiplexer and controlling the addressing pins with the Arduino

digital outputs. This was advantageous to our previous card writer design because it allowed a larger time window for the Arduino to read the data and decide a course of action. If the data bit to encode was a 1, the Arduino had a 24 μs window (from the falling edge of OUT1 to the rising edge of OUT2) to switch signals. This new design was much better than our previous one in many respects, but it was not particularly versatile, as it only allowed for the generation of two distinct signals.

3.4. FSK Signal generation through ASICs

ASICs (application-specific integrated circuits) are used in place of PLAs (programmable logic arrays) and FPGAs (field-programmable gate arrays) for applications that require a single mathematical function to be completed as efficiently as possible. For example, they are not used for computer processors, which are required to perform a variety of functions, but they are often used for cryptocurrency mining rigs, which are required to repeatedly perform the encryption process of hashing.

For the purpose of generating an arbitrary FSK digital signal, we decided that an ASIC, serving as designated hardware for the job, would be more reliable and more efficient than a microcontroller. Additionally, we thought that a versatile arbitrary digital signal generator could have applications in the field of control systems, which further motivated our decision to produce an ASIC. Since this designated hardware was intended to replace the Arduino in our original card writer, we set similar goals for functionality: the device should be able to take a data string (an ID code), a set of timing parameters (rules that convert a data value to a period length), and a clock, and convert them into a digital FSK output signal.

Overview of the circuit

A full explanation of the inner workings of this circuit would be long and unilluminating for the scope of this paper, but we will present a high-level description of this circuit's operation. This circuit takes four inputs: a register titled DATA, and register titled ACTION, a set of two registers each containing a 5-bit binary number, and a clock. For the production of a single period of a given frequency, the circuit begins by shifting DATA by one. If the bit that is shifted out is a 1,

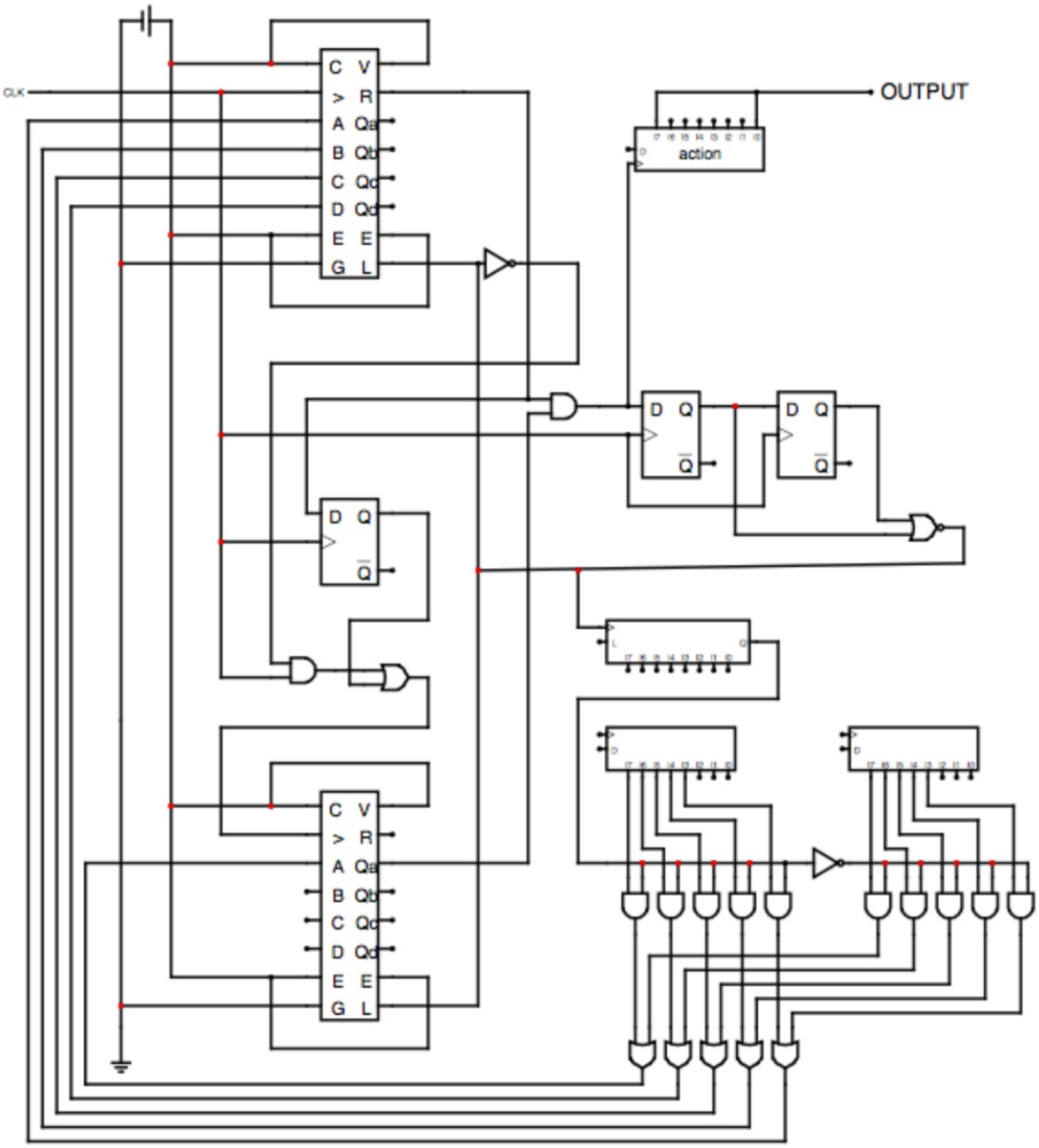


Fig 3.5. Full schematic of our arbitrary digital FSK signal generator.

the left number register (located above the left row of five AND gates) drives the five OR gates in the lower right-hand corner of figure 3.5. Since the right number register is multiplied by the complement of the data output, it drives the OR gate array if the data output is a 0.

On the left side of the circuit are two 4-bit counters that are linked together via the D-type flip-flop, AND gate, and OR gate to form an 8-bit counter. This effective 8-bit counter continuously counts up based on the input clock pulse. When the counter reaches its maximum value (32, in the way we configured the circuit), it returns to some starting value and begins counting up again. This starting value is determined by the five bits on the OR gate array, which are fed into the load pins of the 8-bit counter. In this way, we were able to control the time it took for the counter to reach its maximum value by adjusting the values stored in the two number registers.

When the counter reaches its maximum value, it triggers the ACTION register to shift once. For the sake of our device, ACTION was configured to have the values:

$$\{0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\}$$

Thus, in our case, clocking ACTION twice was equivalent to outputting one period of a square-wave. The length that a given ACTION value remains as the output of the circuit (i.e. the period of one half of the square-wave) depends on the amount of time it takes for the counter to reach its maximum value. This length of time is controlled by whichever number register is driving the OR gate array, which is in turn controlled by the DATA value. The clocking of the ACTION triggers a new data value to be released, and the cycle starts over again.

4. Results

4.1. Card reader

The card reader that we built worked as expected, aside from the occasional glitch due to bad connections. We initially built the circuit on a breadboard, but after encountering connection problems, we decided to solder our circuit onto a proto-board, which was much more reliable. When we probed various parts of the oscilloscope with the oscilloscope, we saw it behaving as expected based on our assumptions during the design phase (section 3.1 of this report). Figure 4.1 (on the following page) shows the peak-detection of the amplitude-modulated sine wave at the junction of the diode, C2, and R1 (in Fig. 3.1).

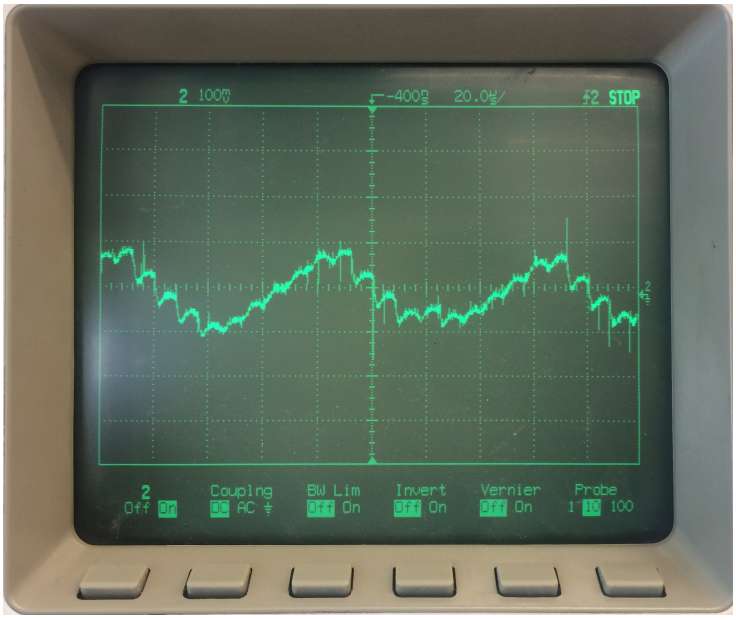


Fig 4.1. Photo of an oscilloscope measuring the voltage at the junction between the diode, C2, and R1 (Fig. 3.1) vs. time.

The fast oscillation shown in figure 4.1. was due to the 125 kHz carrier wave, while the slow oscillation was due to the FSK modulation from the scanned RFID card. If the oscilloscope pictured in figure 4.1 had been set to a longer timescale, we would see that the slow oscillating portion of this waveform was not of constant frequency, but rather it changed between two frequencies corresponding to the FSK modulation from the card.

4.2. Card writer

The first card writer we built (based on the Teensy microcontroller), performed as expected but very unreliably. By using the card reader that we had built, we were able to examine the length of the FSK pulses that the Teensy was generating. According to the specifications of the card manufacturer, we needed to generate a period of length 64 μs for a 0, and of length 80 μs for a 1. In practice, however, the pulses from the Teensy for a 0 ranged from 54 μs to 74 μs, and the pulses for a 1 ranged from 70 μs to 90 μs. This uncertainty was far too high given the precision required for our application. While most of the pulses lengths were within ±4 μs of the target value, there were still occasional outliers which corrupted the data being transmitted. This meant that the card writer would sometimes succeed in sending the right ID code and would

sometimes send an invalid ID code. It was this unreliability which prompted us to search for more accurate methods of generating the required digital FSK signal.

4.3. FSK Signal generation through delayed pulses

This circuit succeeded in generating a series of delayed pulses, as shown below in figure 4.2.

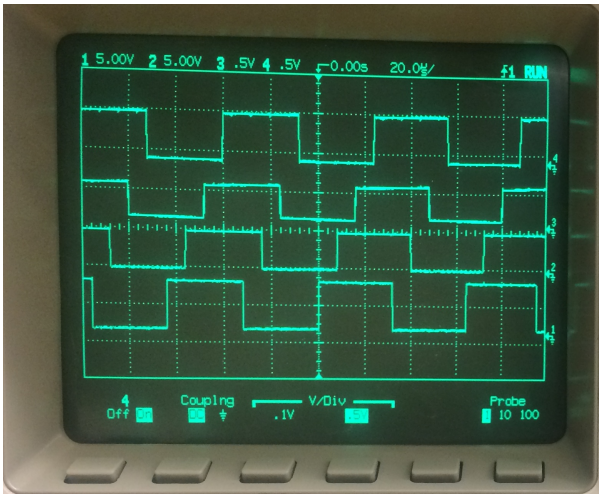


Fig 4.1. Photo of an oscilloscope measuring four outputs of our pulse-delay circuit, as diagrammed in Fig. 3.3.

However, the lack of versatility in this design, as mentioned in section 3.3, led us to stop working on this circuit before it was fully completed. Although we never had a chance to try generating an FSK signal with the addition of a multiplexer and an Arduino, we believed it would be prudent to abandon this project and focus on building a more versatile signal generator.

4.4. FSK Signal generation through ASICs

This circuit worked exactly as expected with no unforeseen errors. We did, however, have to take into account the limitations of the individual parts we used to construct the circuit. Since this circuit depended heavily on a steady clock, it was necessary to calculate the propagation delays inherent in the system. We found that the system behaved well below a clocking frequency of 100 kHz, but above that it started to produce errors. This limitation could in part be remedied by buying better components with lower rise times, but also by redesign of the system as a whole.

5. Next steps

As mentioned in section 3.4, our design of an arbitrary digital signal generator has numerous applications in the field of control systems, which often require digital signals to control physical output devices, including actuators, motors, lights, and speakers. To fully realize the potential of this circuit however, it is necessary to mathematically define a set of general equations describing its functionality. Then, we can examine how to implement those generalized forms into hardware to make a more versatile, and useful, function generator.

While in our case the association between a given data value and its corresponding timing parameter was hardwired in our circuit, this relationship can be generalized as a function of DATA. For example, the function that we chose was:

$$f(DATA[n]) = \begin{cases} 1 = \tau_1 \\ 0 = \tau_0 \end{cases}$$

In other words, if the data bit to encode was a 1, we set a period length of τ_1 for our square-wave output. If the data bit was a 0, we set a period length of τ_0. The simplicity of our function allowed for only two possible output frequencies, but it is possible to allow more frequencies simply by replacing this function with a more complicated one, as long as it can be implemented in hardware.

The other part of this circuit that did not have much relevance for our purpose of generating FSK pulses, but is completely generalizable, is the ACTION register. In the circuit that we built, we let the ACTION simply output a square wave with a 50% duty-cycle. However, if we were to implement a more complicated action sequence, we would be able to get a variety of waveforms. For example, the sequence

$$\{1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\}$$

is a square wave with a 25% duty-cycle. Similarly, if we included multi-bit action sequences, we would be able to adjust the signal generators resolution. The sequence

$$\{00\ 01\ 10\ 11\}$$

represents a 2-bit digital ramp function. This is advantageous over a simple counter however, because we can carefully control the timing of each step of this ramp.