Implementation of a Wireless SNES Controller
Final Project - ECE-394-A

April 30, 2018

Partners: Brian Frost-LaPlante
Nikola Janjušević
Jack Langner
Karol Wadolowski

Abstract

A wireless Super Nintendo Entertainment System (SNES) controller was designed and implemented using infrared-communication. On-Off Keying (OOK) modulation was used to send frames of controller button presses for short range, asynchronous communication to the console. With controller modifications, a transmitter able to fit within the original frame was built. A receiver bar separate to the processing circuitry was made for signal reception independent from the placement of the console. Error detection at the receiver mitigates frame corruption due to the wireless channel. The result is a compact wireless SNES controller with a receiver that houses the console.

Contents

1 Introduction

Wireless controllers have been a staple to console gaming for over a decade now, with most modern wireless controllers implementing Bluetooth technology. Older consoles, however, were created in a time where such technology was not available, and often did not have first-party wireless controllers. We decided to implement a wireless controller for one of our favorite defunct consoles: the SNES. While this is fairly simple to implement through the use of special-purpose ICs, it is pedagogically uninteresting, and does not allow for the low-level design offered by using only discrete circuit elements and more fundamental integrated circuits. Thus, we decided on the use of infrared light, a historically prolific form of communication.

We attacked design from the top down, first determining the SNES protocol, and then constructing a basic functional block understanding of our project. We laid out key design difficulties, and then went on to design the components in order of the signal path. Our design process, along with some analysis and justification, is followed closely in the sections to follow.

2 Project Overview

A simplistic but important view of this project is to consider the two communicating ends of this circuit; the console and the controller interface; as being mostly independent. The controller is to behave as if the console is asking it for information, it is to send that information to the console interface, and the console is to behave as if it is receiving data from the controller. Of course, this point of view leaves out incredibly important considerations necessary to the functionality of the wireless controller, however it provides the basis for our design procedure.

Below, we outline the protocol implemented by the Super Nintendo, which we must interface with through the console-end circuitry. We then present the outline for the final implementation, inspired by this protocol and a few fundamental design decisions.

2.1 Super Nintendo Protocol

It takes only a cursory amount of research to determine that the Super Nintendo takes serial inputs at a rate of 60 Hz, but the specifics of this input scheme are most easily determined by directly providing stimuli to the controller and observing its outputs. In this section, we provide the experimentally determined data transfer protocol employed by the Super Nintendo in its standard wired setup, as this operation must be emulated, in some sense, by both the controller interface and the console interface.

The controller itself contains fairly minimal circuitry, manifesting on a single PCB. Each button is connected to a v520b IC, which is simply a 12-bit parallel-input serial-output shift register. The console and controller are connected by five wires, although the connector shown below deceptively has seven pins. The wires are color-coded differently on different brands of controllers, so when modifying a new controller, they must be tested individually.
Two of these wires provide 5V and ground to the controller. One wire is used to send a latch signal to the controller, clocking parallel data – representing which buttons on the controller are pressed – into the v520b. A signal on another wire follows the latch pulse with clock pulses, clocking serial

data out of the v520b and into the console through the final wire. Each 60th of a second, the console sends a 12 microsecond latch pulse to the controller, followed by 16 24 microsecond periods of clock (a frequency of about 42 kHz). At each rising edge, the controller outputs the state of the next button for the first twelve pulses in the order shown in the table below. The output of the controller is low if the button is being pressed and high if it is not, and the last four bits of the output are always high. An example of the three active signals during one cycle in which SELECT, LEFT and L are being pressed is shown in Figure 2.

It is important to note here that while data has to be clocked into the console at 42 kHz, data can be clocked out of the controller as slowly as one desires. For example, in our final setup, data is clocked out of the controller at 3 kHz. It is also the case that, physically, the LEFT and RIGHT buttons cannot be pressed at the same time. Similarly the UP and DOWN buttons cannot be pressed at the same time. While this is not part of the protocol per se, this fact influences the output data in a way that is used twice in the design of our wireless communication scheme.

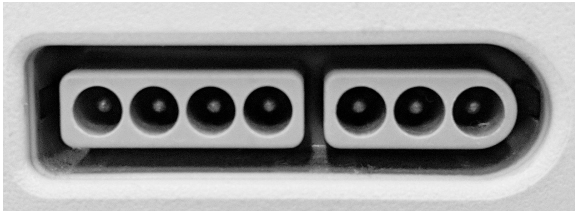

Figure 1: Input to SNES Console

Bit Number	Button
0	B
1	Y
2	SELECT
3	START
4	UP
5	DOWN
6	LEFT
7	RIGHT
8	A
9	X
10	L
11	R

Table 1: Button Sequence

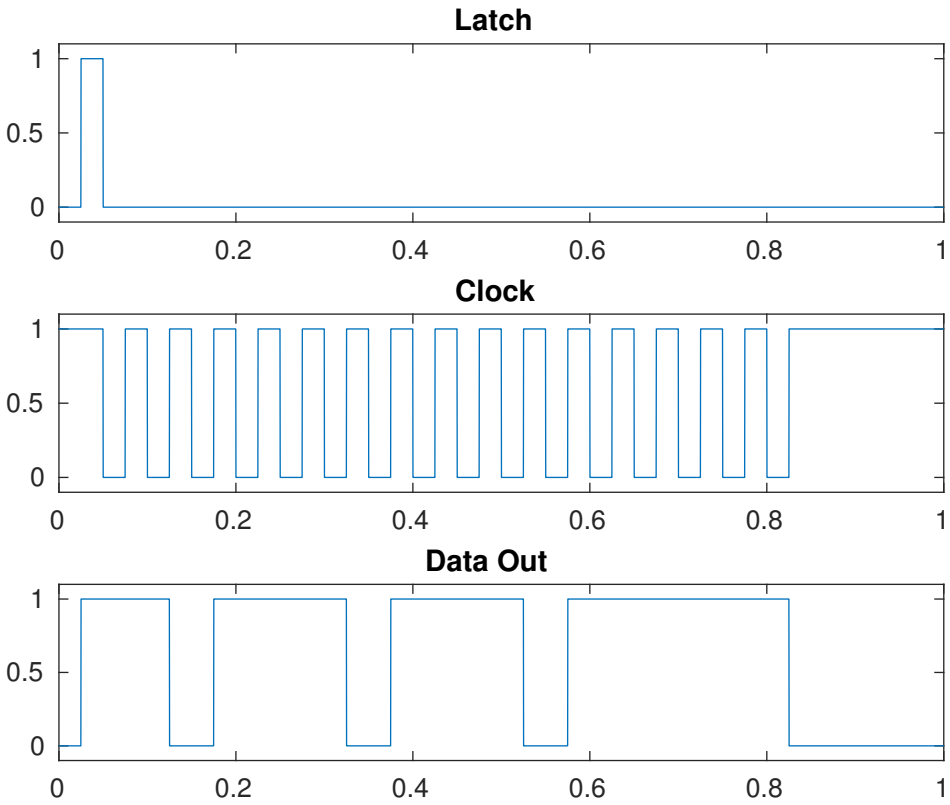

Figure 2: Normalized View of One Cycle of SNES Protocol

2.2 Functional Block Diagram

With this protocol in mind, we designed a very simple functional block diagram, as seen in Figure 3. The controller interface block represents some circuitry which latches and clocks the controller at least once every 60th of a second and applies a simple data modulation scheme, presented in the next section. The transmitter is an IR LED with some additional circuitry to boost the intensity. The receiver is, essentially, a phototransistor whose current output is turned into a voltage and processed so as to generate square pulses to be fed into the receiver interface. The receiver interface is, most basically, a synchronization circuit. Its purpose is to accurately convert the time data from the receiver into digital bits stored in shift registers. The console interface's purpose is to ensure safe and reliable transfer of this data into the console in the same way a wired controller would. The design process, for the most part, followed the signal path from controller end to console end. That is to say that the data modulation scheme and controller interface were designed before the transmitter, and so on. As such, we follow this design procedure in this order in the following sections.

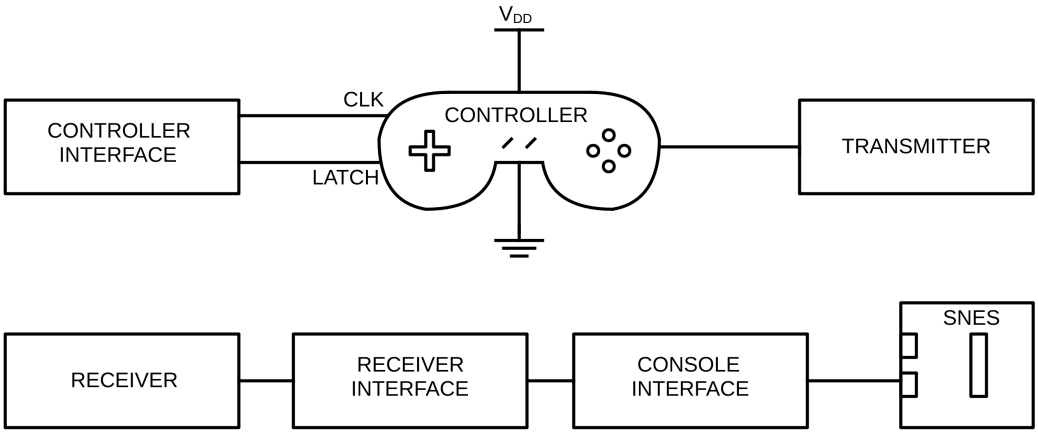

Figure 3: Highly Simplified Functional Block Diagram

2.3 Key Design Challenges

There were several constraints we placed on ourselves before starting the design procedure. These included the following: fitting the controller interface within the controller housing, fitting the receiver interface and console interface within the footprint of the console, and have the console receive new data each console frame. In order to fit the controller interface and transmitter inside the controller, a PLD was used to implement the controller interface. The PLD and 7555 timer, as well as the components for the transmitter, were soldered onto a perf board so that they would take up less space. In order to fit the console and receiver interface within the footprint of the SNES, PLDs were also used to implement large portions of the logic. Lastly, to get new data to the console each frame, data was sent out around twice as fast as the SNES would request it.

Some unexpected challenges that arose when we were implementing our initial designs were: frequency dependence of the emitter and receiver components, synchronizing the controller interface and receiver interface frequencies, synchronizing the incoming data to the receiver interface clock, error detection, and light interference. The frequency dependence problem was solved by choosing a low frequency that still allowed for data to be sent twice a frame. Synchronizing the clock frequencies was done by getting the two clocks close to the same frequency by testing various components and then restarting the receiver clock at the beginning of each new data sequence. To synchronize the clocks, the console end clock was reset each time a new data sequence was observed. This helped as the two clock signals would be relatively synchronized at least while data is being read in. Synchronizing the data was solved by adding a pilot bit which indicated when to start receiver functions. Error detection was needed as when data was lost, the console would receive all low signals which would correspond to all buttons being pressed. This was implemented on two PLDs as to throw out any illegal inputs, for example UP and DOWN at the same time. Lastly, light interference was diminished by adding a housing that blocks any light coming from directly above the sensor bar.

3 Data Modulation

The phototransistors available in the lab begin to act noticeably nonlinearly as frequency is increased, and as such, we decided to choose a data modulation scheme that allowed for the sending of data at a relatively low frequency. As is explained in the section on SNES protocol, the meaningful data transmitted by the controller is only 12 bits, and it is asked for at 60 Hz. We decide that the controller, at the very least, ought to send a stream of lows the size of one data sequence (12 bits) followed by one pilot bit so as to signify the start of each sequence. Similarly we attempt to transmit an entire button sequence at least slightly faster than 60 button sequences per second, so that new data is available to the console at each frame. This gives a lower bound on the data rate of 25 bits per 60^{-1} seconds, or 1500 bps. With each bit representing one clock period, we decide on a frequency of 3 kHz, giving twice this lower bound on the data rate. This is a low enough frequency for our phototransistors to handle, but high enough to allow for a slightly over-compensated modulation scheme, and to assure that new button sequences arrive at the console each in-game frame.

The decided upon modulation scheme is on-off keying, an extremely simple method of communication where a high bit is represented by the IR LED being on for one clock period and a low bit is represented by the IR LED being off for one clock period. The modulator works using a 3 kHz clock, with each entire sequence occupying 32 periods. The first 16 sent bits are low, and the 17th bit is always high. This 17 bit sequence is an extremely safe starting sequence, as it is of longer length than the actual data. This start sequence is followed by the data, and then finally three high bits. These three high bits, as well as the pilot bit, in some sense "hug" the data, and we thus refer to this procedure as *data hugging*. Data hugging allows the starting bit sequence to be distinguished from pressed buttons at the beginning and end of the data sequence (the B and R buttons primarily, followed by combinations of B and Y, L and R, etc.). Beyond this, the pilot bit is used not only to signify the start of the new sequence, but also for error detection and controller receiver synchronization (as explained in detail in later sections of this document).

With this scheme, we send a button sequence once every 32 periods of a 3 kHz clock, yielding a new button sequence at the receiver over 90 times a second. This is safely above the required 60 Hz. It is of note that in the standard SNES protocol, data is transmitted at over 40 kHz. Data must still be fed into the console at this rate, but this can be handled at the console interface so long as a new data sequence is available whenever the console expects one, allowing us to work, for the most part, at a much lower frequency.

4 Sub-System Design

In the below sections, we walk through the design process and operation of the major sub-systems of our project. We follow an ordering based on the signal path, starting at the controller and ending at the input to the console. This ordering is also, for the most part, true to our design process.

4.1 Controller Interface

The controller interface was designed to simulate the console's Latch and Clock signals. The purpose of this was to get the set of button inputs out of the controller, modulate it, and transmit it. In order to do this we implemented a circuit that regularly produces Latch and Clock signals that get

fed into the controller. To do this a finite state machine (FSM) was designed to produce the desired signals (Figure 5). The FSM was implemented on an ATF22v10C PLD chip. To cycle through the FSM a 3 kHz clock a square wave was supplied to pin 1 of the PLD. The clock signal was generated using a 7555 timer as seen in Figure 4 below. The FSM was a 32 state counter. During the first 16 states the controller was left idle so that no data would be output. This section of 16 low bits helped us identify where we were in the data sequence on the receiver interface side. During the next 16 states the latch was set high for half a state. This wrote the button pushes to the internal registers of the controller. The 3 kHz clock then clocked out this data to the PLD. Once it reached the PLD a single high bit was placed in front of the data and three high bits at the end of the data. These high bits were added for our data hugging protocol as explained in the previous section.

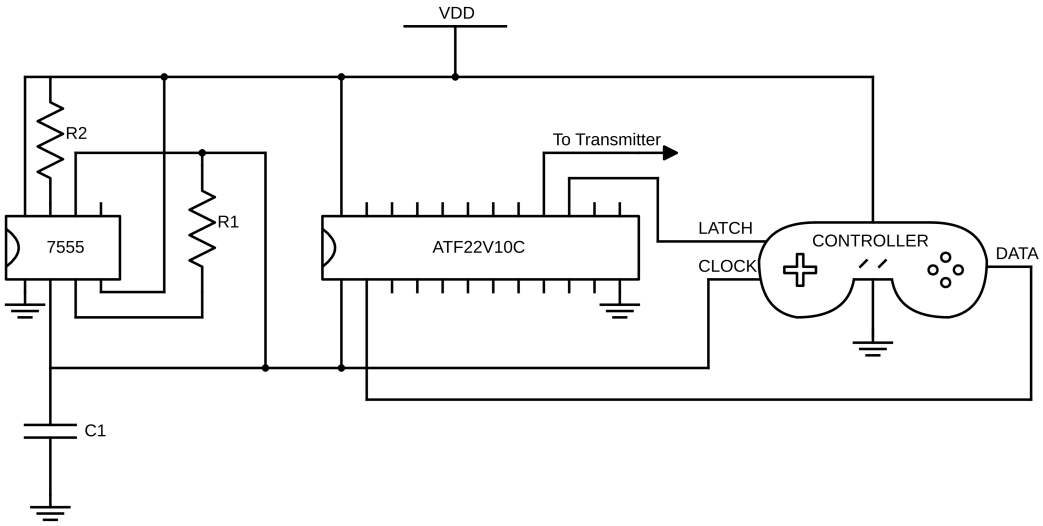

Figure 4: Controller Interface Schematic

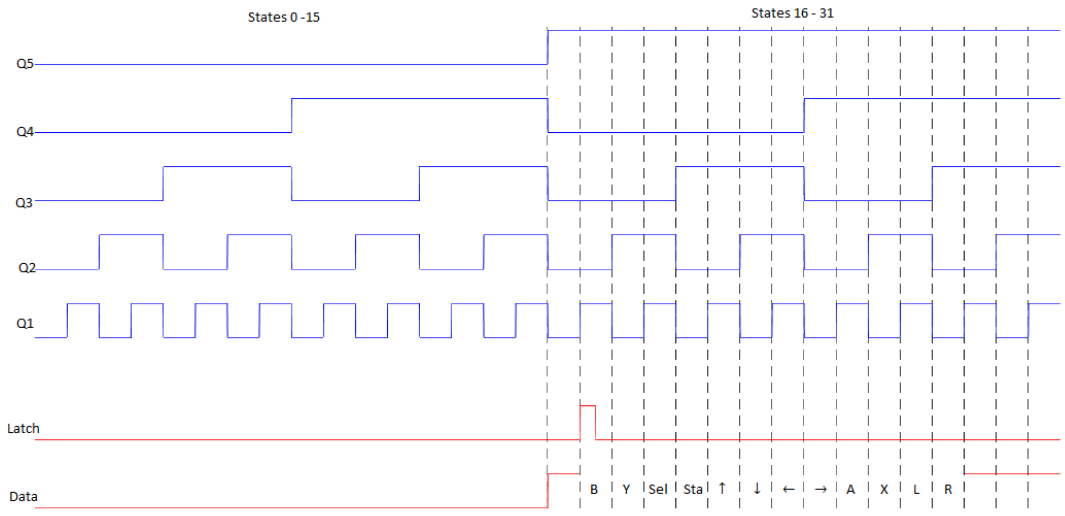

Figure 5: Timing of the Latch and Data Signals

4.2 Transmitter

While the desired signal to be transmitted is output by the controller interface circuit in 5 V amplitude square pulses, it is important to remember that the IR emitter is a current device. The PLD itself can only source a maximum of 3.2 mA according to its datasheet, and with this current through the LED, very low range is observed. To increase the intensity of light emitted by the LED, BJTs in a Darlington pair configuration are used to amplify the current through the LED. This configuration is seen in Figure 6. The LED used is rated for maximum power of 250 mW. Assuming a voltage of 0.7 V across it, and noting that current only flows through it for less than half of the time that the controller is on, we get a maximum forward current of about 700 mA. Assuming a drop of 0.7 across each collector-emitter junction as well, the voltage across resistor R2 is, at a maximum, 6.9 V. Using a 15 Ω resistor, the maximum current through it, and thus the LED, is thereby 460 mA. This is quite high, and yields excellent range without overdriving the LED. The transistors used are rated for a maximum of 600 mA, so they are also safely within their operating range.

This current amplification method was inspired, mostly, by remote controllers for televisions, which use Darlington pairs to achieve large ranges as well. It is of note that remote controllers are incredibly bursty, so the power consumed by them is fairly minimal. On the other hand, our circuit is designed to operate continuously with a median duty cycle of 50%. Our circuit is quite power intensive, so it will drain a 9 V battery in about 4-5 hours of continuous usage. This is in line with the battery lives of modern wireless controllers, however, for cost effectiveness, rechargeable batteries are suggested.

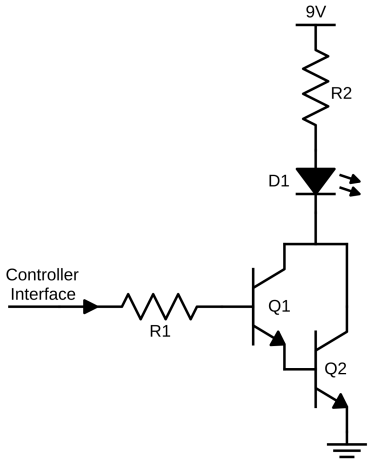

Figure 6: Transmitter Schematic

4.3 Receiver

The purpose of the receiver is to turn incoming IR data into true digital data. The receiver multiplexes the outputs of 8 phototransistors, which are distributed in space across a receiver bar (as described in the section on Physical Implementation). This is to counteract the fact that the IR LED used had only a 10 degree beam angle, so the beam of the LED needs only to excite one of the 8 phototransistors for the data to be received. Phototransistors generate current in approximately linear relation to light intensity, so we first convert the current from each phototransistor to a voltage using a transimpedance op amp configuration. We compare the magnitude of the received voltages from two transistors at a time, and multiplex based on which is larger, as is seen in Figure 7. We repeat this multiplexing until the highest intensity of the 8 received signals is determined.

The phototransistors constantly emit a small current as a result of the lights in the room, however care is taken so that the transimpedance amplifiers will not convert this to a voltage above a diode drop. As a result, when the strongest chosen signal is fed into LED D1, as in Figure 8, the lights in the room alone will not allow the LED to conduct. Of course, any diode would serve this purpose, but the use of an LED is intentional as it provides visual feedback to the user. That is, the LED will only light up if any of the phototransistors are being excited by the controller, so the user can tell how s/he should aim the controller at the receiver bar.

From this point, the strongest signal is amplified by an inverting amplifier configuration so as to drive the signal to/near rail, and then it is inverted with a 4069 chip so as to achieve true digital data. From this point forward, all processing can occur digitally.

The resistance used for the transimpedance amplifier is 4.7 kΩ, so that each milliampere of current is converted to 4.7 V. The inverting amplifier uses resistances R2 = 1 kΩ and R3 = 30 kΩ, giving a gain of about 30. It is desirable to have rail-to-rail swing before being fed into the inverter, which

is operating at 5 V, so a voltage of about 170 mV must be produced by one of the transimpedance amplifiers to produce a valuable signal. This corresponds to a phototransistor current of about 36 μA. Of course, whether or not this can be achieved is dependent on the direction of the LED and its distance from the receiver. We see useful data being transmitted from distances as far as 3.25 meters. An increase in the amplification on the receiver end could lead to reception from longer distances, but also becomes vulnerable to over-amplification of noise and light in the room. As the range we are seeing is a meter larger than the length of an actual super nintendo controller, and the receiver bar does not need to lie directly on the console as a standard controller does, we determine that this range is both safe and respectable.

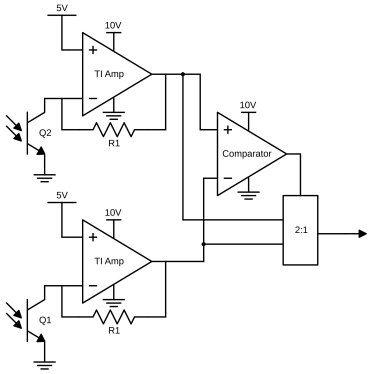

Figure 7: Receiver Pair Functional Block

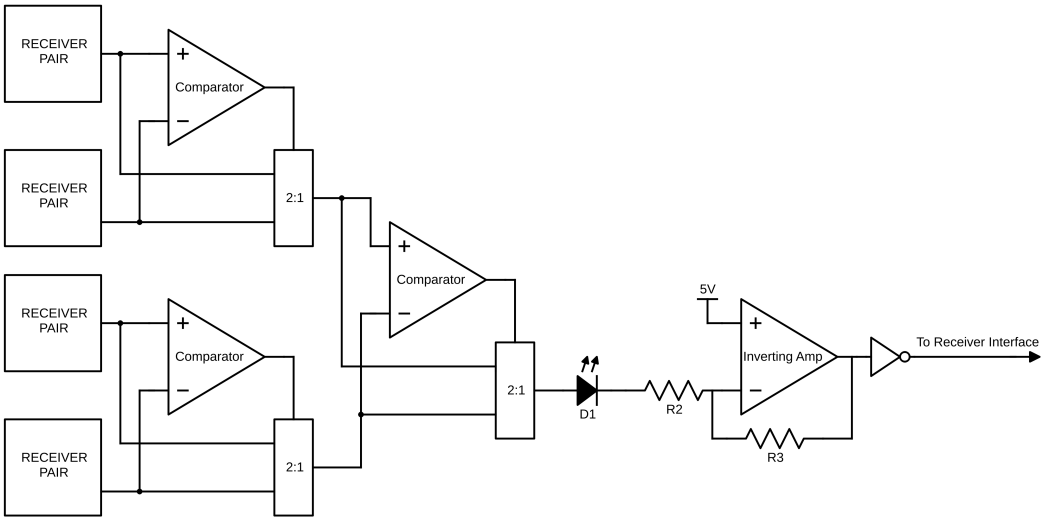

Figure 8: Receiver Schematic

4.4 Receiver Interface

The purpose of the receiver interface is to detect the start of a data sequence based on the signal coming from the receiver portion of the circuit. After detecting the start of a data sequence the receiver interface must also store the data (what buttons were pressed) into shift registers so that the console interface can retrieve the data when it is requested by the SNES.

The FSM for the receiver interface was modeled as seen in Figure 9 . The FSM works by first detecting a sequence of six lows from the input data. The reason why it detects six lows is that the maximum amount of lows inside the data sequence portion of the Data signal in Figure 5 is five lows. Five lows is the maximum amount of lows present in a row in a valid input. Note that up and down / left and right can't be pressed at the same time. During this portion of the FSM (0 Low to 5 Low) a 3 kHz clock was used to advance the states of the FSM. A 3 kHz clock was used to match the clock frequency at which the controller interface was running at. So after detecting six lows in a row the receiver interface knows that it is not receiving the button data. At this point it waits for the pilot bit which denotes the start of the button data sequence. During this waiting state the clock to the FSM is set to the Data signal since the rising edge from the low portion of the sequence to the pilot bit denotes the start of the data writing stage. After transitioning into the last stage the FSM then enables writing into the register and stores these 12 button presses in a Serial In Parallel Out register (SIPO) where it can be accessed later by the console interface. While the controller is inputting the button data into the SIPO registers a counter goes from 0-11 (one number for each button in the data stream). When the counter reaches 12 the receiver interface knows that the data has been stored and then returns to the 0 Low state where it waits to see it is outside the button data portion of the Data signal. The transition from the writing state to the 0 Low state occurs on the rising edge of the counter going from 11 to 12.

The functional block diagram in Figure 10 shows an overall view of how the Receiver Interface Circuit was implemented physically. The main portion of the Receiver Interface was implemented in the ATF22v10C which contained the FSM described earlier. The FSM on the PLD interfaces with a 4 bit counter chip that counts from 0-11 when in the data writing stage. The FSM also outputs the button to the SIPO register when in the writing stage as well. The PLD takes a 3 kHz clock, the data signal from the receiver, and the counter value as inputs. The outputs of the PLD include a clock for the counter so that the counter only increments when in the writing state, a reset for the counter so that the counter starts from 0 each time the FSM enters the writing state, the button data for the SIPO registers, and the current state of the FSM for use by the console interface.

Note that a buffer was used to avoid a race condition that was occurring.

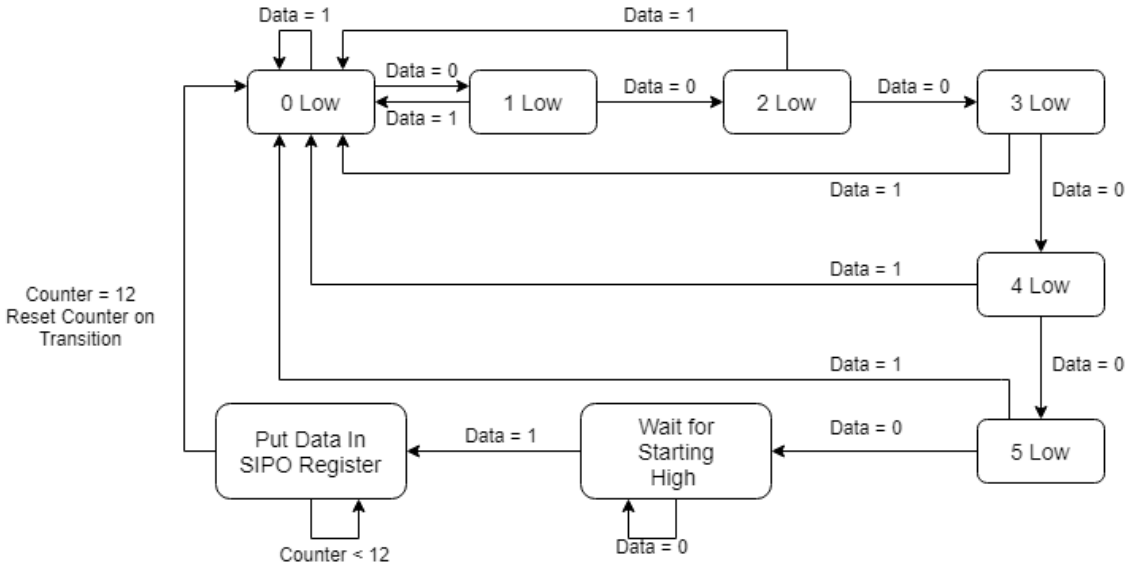

Figure 9: State diagram for receiver interface.

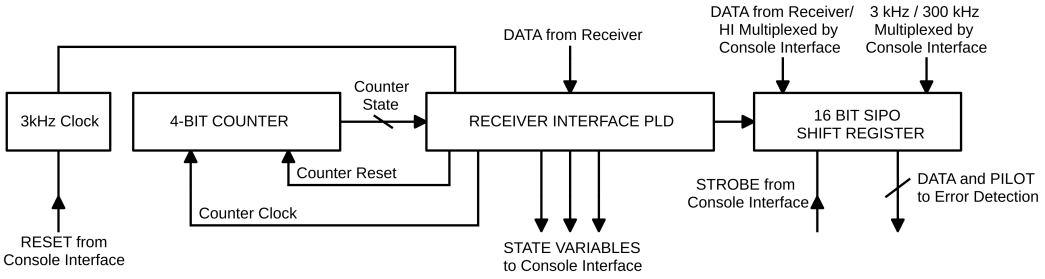

Figure 10: Functional Block Diagram for the Receiver Interface.

4.5 Error Detection

A fairly simple error detection circuit is implemented between the SIPO registers of the receiver interface and the console interface. It can detect three distinct error: a missing pilot bit, UP and DOWN being pressed at the same time, and LEFT and RIGHT being pressed at the same time. These three scenarios cannot occur in an actual SNES controller, and as such they are necessarily due to transmission errors. As can be seen in the logic diagram in Figure 11, this is employed simply by checking these conditions and setting all bits high in the case that any of these three errors occur. This effectively sends a null input to the console, as if the player had not pressed any buttons. This does not detect all possible errors, but it does correct each error which can be distinguished from valid data using this communication scheme. We have found that a majority of errors are of this type, and the player experience is near seamless. Without this error detection, however, about one

transmission in every 10 seconds or so leads to an error of this type, and undesired inputs are fed to the console.

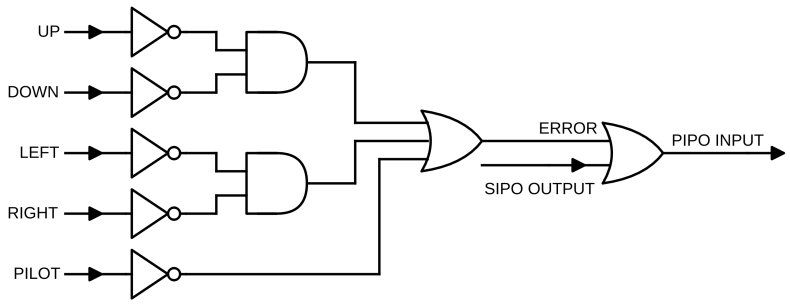

Figure 11: Error Detection for Each Input Bit

4.6 Console Interface

The console interface was designed to take the button inputs that were stored in the SIPO registers by the Receiver Interface and send that information to the SNES when it was requested. The Console Interface was implemented with a four state FSM (Figure 12).

The first state S0 uses the state inputs from the Receiver Interface FSM (T0, T1, T2). When those inputs are all high the Receiver Interface is writing data into the SIPO registers so we can't read the data out of them. So we wait until data stops being written into the SIPO registers. Also right after entering state S0 a counter is reset. After writing to the SIPO is finished the Console Interface can grab the data from the SIPO registers. So the FSM goes into state S1 where it strobes the SIPO chip. The SIPO chip (CD4095) has two sets of registers. When strobed the first set of 8 registers gets clocked into 8 new registers which are on the Parallel Out pins of the chip. After strobing the data to the output of the SIPO chips the data can now be stored in PISO registers (Parallel in Serial Out), so the FSM goes into state S2. In state S2 the outputs of the SIPO chip are read into the error detection PLDs. They are then output of the error detection PLDs and into the PISO chips and then the SIPO chip is filled with High bits. The SIPO is then filled with HIGH bits to avoid reusing the same data twice. The all high bits represent no buttons pressed. After this is done the FSM transitions into state S3 where it waits for the SNES clock. Once the SNES clock signal comes the data is read out of the PISO registers and into the console. The same SNES clock is used for our counter which keeps track of how far in the clock signal we are. Once the counter reaches 16 we know that all the data has been sent out to the SNES and we can move back to state S0.

The State Diagram (Figure 12) was used to make the functional block diagram of the Console Interface (Figure 13). The main components of the functional block diagram are the Console Interface PLD which contains the FSM in Figure 12, the Error Detection PLDs, and the 12 bit PISO registers. The Console Interface PLD has 5 inputs: a 300 kHz clock thats generated by a 7555 timer, the state variables T0, T1, and T2 from the Receiver Interface PLD, and the most significant bit (MSB) of the 5-bit counter. This PLD also outputs several key signals that control what

is happening to the SIPO and PISO registers, what is being output to the SNES, what clocks are being used be each of the registers, and what data is being sent to the console or SIPO. The first output is called SIPOMUX in the code in the Appendix. SIPOMUX controls what input is going into the SIPO registers and what clock that input should have. In states S0, S1, and S3 the SIPO is being clocked with the 3 kHz and the input in that case is the actual button data coming from the console interface. In state S2 that data has just been strobed so the data in the SIPO register has already been read and is not needed. So using the 300 kHz clock the registers are filled with High bits denoting no buttons pressed on the controller. This prevents the same input from being used multiple times. This was used for the the two 2:1 MUXs that have output SIPO Clock and SIPO input. The next output of the PLD is Strobe which is simply high when in state S1. The CTReset output is high in state S0 in order to reset the 5-bit counter back to 0 so that it is ready for the next time the SNES clock signal comes. The PAR output controls whether the 12 bit PISO register is currently reading in data parallely or outputting it serially. The CLKMUX output is used to control what clock signal is used when the data is being read in parallely into the PISO and when the data is being output serially. When being read parallely the 300 kHz clock is used and when being read out serially the SNES clock is used. This corresponds to the 2:1 MUX with label PISO Clock. This same signal also controls the 2:1 MUX for DATA to SNES. Lastly the RST signal resets the 3 kHz clock used by the Receiver Interface when in the waiting state since we want the data incoming to be synchronized with the clock of the Receiver Interface. The Console Interface PLD does all this to ensure that good inputs are being fed into the console at the proper time.

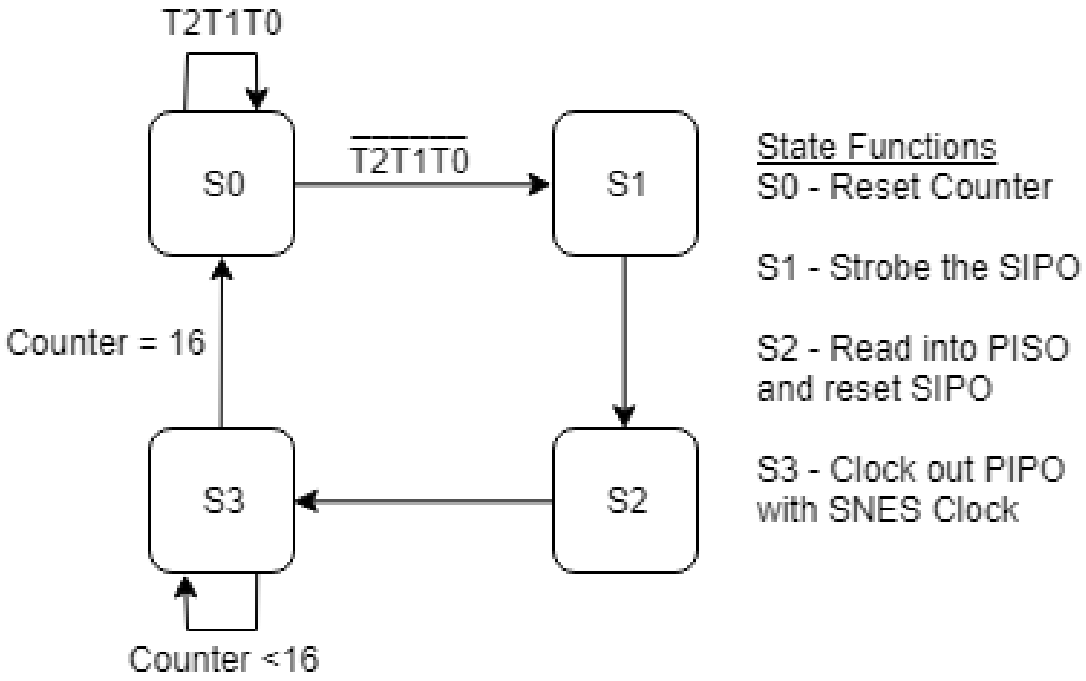

Figure 12: Console Interface State Diagram

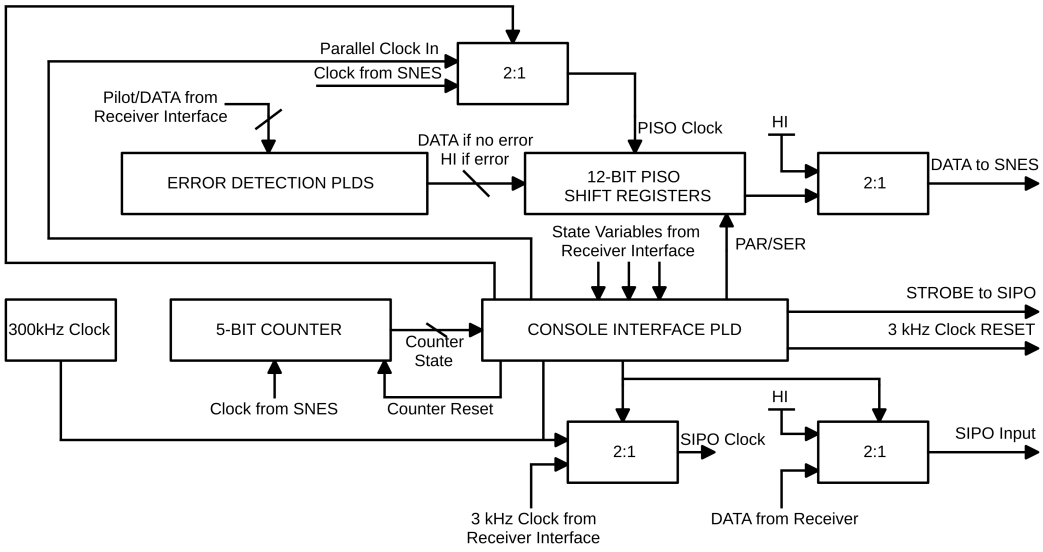

Figure 13: Console Interface Functional Block Diagram

5 Physical Implementation

The above concepts were, of course, implemented in hardware. Below, we detail the physical manifestations of the ideas laid out above, including both physical circuitry and aesthetic housings for the circuitry.

5.1 Controller Modifications

As a result of having physically compact circuitry on the controller end, we were able to fit all but the 9 V battery inside of the controller. This required some basic physical modifications, including the removal of some extraneous plastic within the controller and the creation of a few holes through which to fit certain circuit components. The controller interface and transmitter circuitry were soldered onto cut-up perf boards, which proved quite challenging due to the fairly intense spatial constraints. A switch to turn the controller on and off was implemented, and a circular hole was drilled into the top of the controller so as to expose only the top of the switch when the controller is closed. A yellow LED was similarly implemented to indicate whether or not the controller is on, and a hole was drilled so that this would be visible to the user. The IR LED was fit into the hole through which the wire would normally run, and was radially coated with tin foil so as to reflect all IR beams away from the controller. This minor modification effectively increased the controller's range by quite a bit, and was inspired by a common do-it-yourself TV remote fix.

Figure 14 shows the internals of the controller before it has been screwed closed. The right perf board contains the Darlington pair and a voltage regulator, while the left one contains a 7555 and

the controller-interface PLD. The closed controller is seen from two angles while both off and on in Figures 15 and 16.

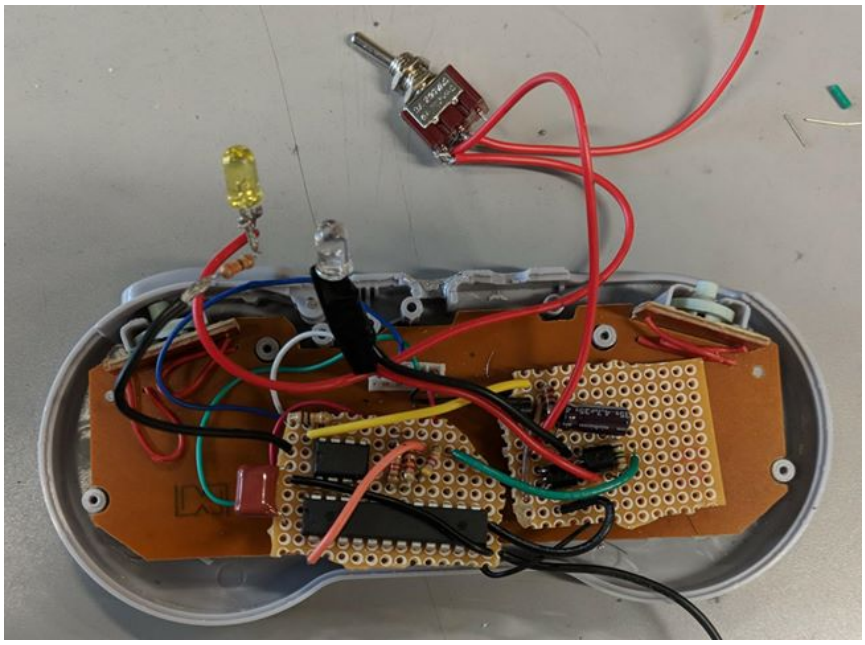

Figure 14: Controller Interface Interior

Figure 15: Front of Closed Controller While Turned Off

Figure 16: Closed Controller While Turned On

5.2 Receiver Bar

As described in the section on the Receiver circuit, 8 separate phototransistors were used, along with some op amp and multiplexer ICs. This circuitry was fit onto two breadboards. Inspired by the Nintendo Wii's sensor bar, we arranged the phototransistors in a rectangle, allowing for more horizontal angular range than vertical. This is physically sound, as the tendency to point the controller forward rather than up of down is usually enforced by the position of the wire with respect to the controller. The final circuit is shown in Figure 17.

The housing for this bar is simply a rectangular prism made of acrylic, to which this circuit is super glued. Figure 18 shows the receiver bar in its casing with the LED on, corresponding to a connection between the receiver and the controller. The walls and back of the casing are made with eighth-inch-thick matte black acrylic. The front is made with eigth-inch-thick smoke grey translucent acrylic. These pieces were each cut with the student-operated laser cutter at The Cooper Union. A small hole in one of the smaller walls is used to connect the stereo cable between the console-side circuitry and the receiver.

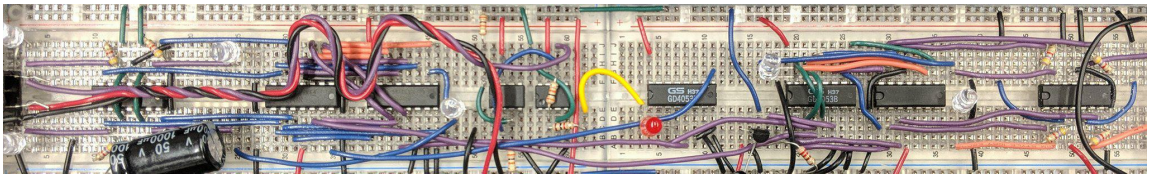

Figure 17: Receiver Circuit

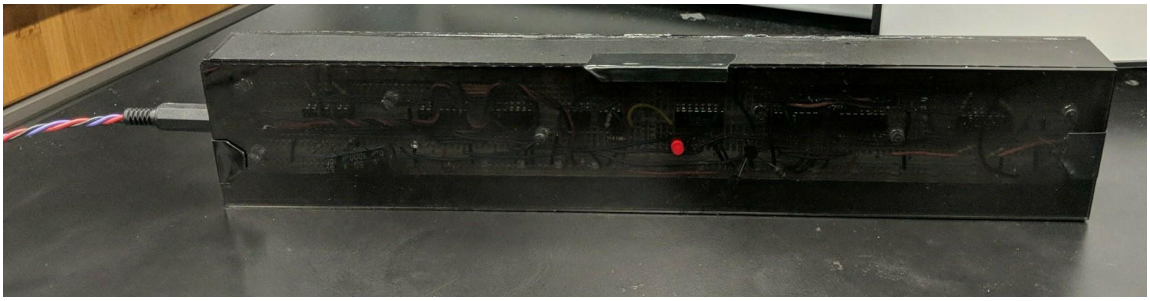

Figure 18: Receiver Bar with LED On

5.3 Console Interface Housing

The console interface housing contains the receiver interface and console interface circuitry. The housing was made from eighth-inch-thick matte black acrylic and clear acrylic. The matte acrylic was used on the externals of the housing for aesthetic purposes, as it would hide the wiring of the two interfaces within. The clear piece of acrylic was used as seen in Figure 20 in order to prevent tampering with the circuit, but also giving a view of the circuit.

The front portion of the housing has two holes: one for a stereo jack entrance that was used to connect to the console, and the second for an LED that would indicate if the circuit was on or off. The place where the holes were made can be seen in Figure 21. The back part of the housing had three holes which can be seen on the left side in Figure 20. The top-left hole was used for power delivery. The middle hole was used to connect to the receiver using the braided wires seen in the image. The final hole was used to implement a switch to power the circuitry. The clear acrylic had a notch in it for access to the circuit in the case that it needs to be fixed.

The SNES fits on top of the housing without falling into it. The sides of the housing also go high enough so that the SNES can be placed into it without sliding out.

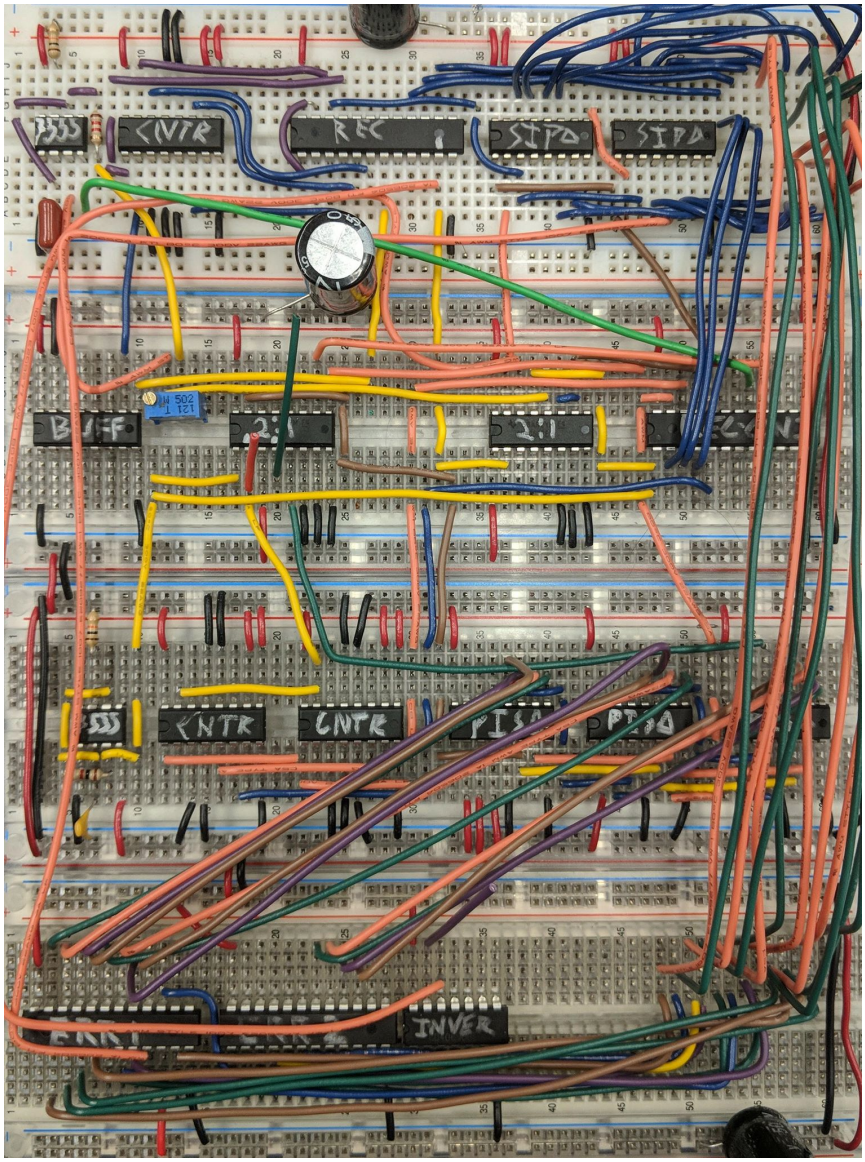

Figure 19: Console Interface Circuit

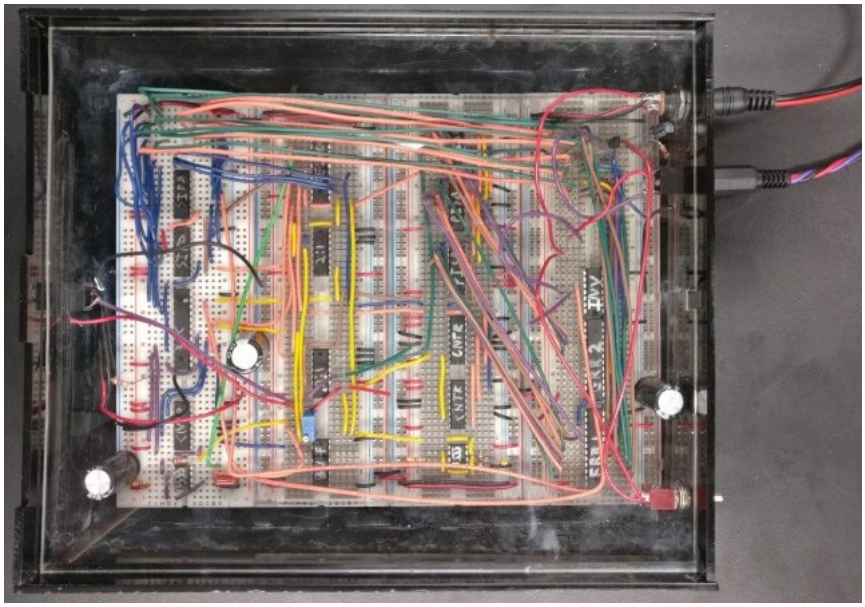

Figure 20: Console Interface Circuit inside an acrylic housing.

Figure 21: SNES placed in place on top of housing.

5.4 Power Supplies and Connectors

Naturally the SNES console outputs 5 V and ground through two wires to power the controller circuitry, which consists of a single IC rated for a maximum input of 7 V. Of course, this must now be provided externally. Furthermore, the op amp circuitry within the receiver bar cannot work at only 5 V, and given that the controller output is only designed to drive a single IC, we decided that it would be safe to not have the console itself power the console-end circuitry. As such, we were to consider powering both ends of the circuit independently.

On the controller end, a 9 V battery was decided on. The circuitry interfacing directly with the controller, namely, the 7555 timer and PLD, were powered at 5 V through use of a voltage regulator, while the transmitter circuit made use of the 9 V supply voltage to boost the current through the IR LED. The regulator used was a 78L05, which suits our purposes quite well, as it comes in a very small package, and requires minimal circuitry (only a 220 Ω resistor and a decoupling capacitor) to operate at 5 V. This allowed us to easily fit the regulator along with the rest of our circuitry inside of the controller. The 9V battery sits in a battery pack that is super glued to the back of the controller. A switch on top of the controller disconnects the battery from the rest of the circuit when the controller is off, so as to conserve some battery life.

On the console end, a connector in the console interface housing connects the circuit to a 10 V AC-DC converter brick from wall. The stereo cable connecting the receiver to the receiver interface carries 10 V and ground to the receiver circuitry, and carries receiver data as well. A regulator (once again a 78L05) on both the receiver and console-side circuitry provides 5 V. For the receiver, this serves as mid-rail for the op amps, whereas for the console-side circuitry, this provides VDD for each IC. The console is meant to be fed data at 5 V, so as long as the console's ground is connected to that of the console-side circuitry, the console need not power anything directly. As the latch and VDD pins of the console are not necessary for the operation of the circuit, only the clock, data and ground pins must be connected. Thus, we use a stereo cable once again to connect these three pins to the circuit, and leave the other two disconnected.

The used stereo cables are shown in Figure 22. The top cable feeds into either of the two controller ports on the SNES on one end and into a 3.5 mm stereo connector on the other end. It was created via cutting the cable from a wired controller, soldering connections to the 3.5 mm plug, and finally applying heat shrink tubing to the otherwise exposed wires. The bottom connector goes between the receiver and the console-side circuitry. Only one end is shown as it is about a meter long, but it has a 3.5 mm stereo plug on either side. The three wires here are simply braided and not heat shrunk, for greater flexibility in the cable.

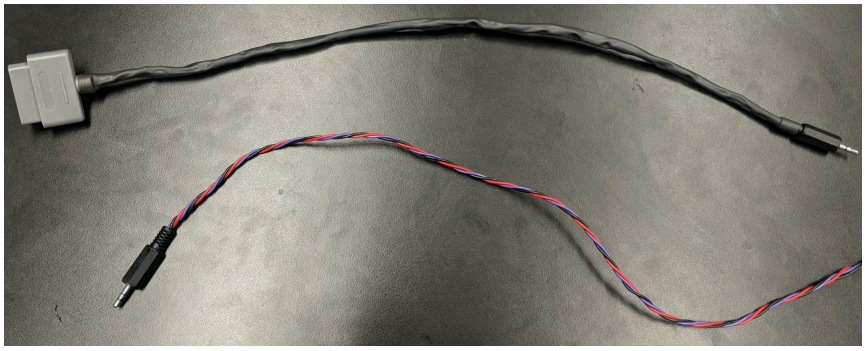

Figure 22: Connector Cables

6 Future Work

The controller, receiver and console interface circuits as of now are not in need of improvement; they do the exact job they are required to do in little physical space. Improvements, if they were to be made, would thereby need to occur at the transmitter or receiver blocks. One observed nuisance is that the low beam angle of the IR LED we are using causes the controller to fail to work when pointed in a large range of directions. Use of a high power LED with a wider beam angle could mitigate this problem without substantially decreasing the range of the controller.

Substantial receiver-side modification would be trickier, but also would provide improved angle and range. For example, one could implement maximal ratio combining at the receiver rather than the simple selection diversity technique implemented currently. More simply, we could implement a higher resolution array, using more phototransistors pointing in optimal directions. The creation of the optimal receiver is a complicated geometric problem, but could likely be solved given ample time.

Error detection could also be improved, as the current implementation uses only basic frame sensing and the checking of a few specific error conditions. A more complicated cyclic redundancy check could be performed, being computed at the controller interface and checked at the console interface. This, however. would require a fairly large change to the hardware, as more bits would need to be processed in each frame. Its benefits would be fairly minimal, especially with improved reception, so this would not be the first modification to be made.

Lastly, the method of IR communication could be replaced with nearly any other method fairly seamlessly by only replacing the transmitter and receiver. Some alternatives include AM or Bluetooth. Should a second controller be constructed using this modulation method, it would not interfere with the IR controller, and thus one could implement a two-player setup.

7 Conclusion

The wireless SNES controller implemented was able to overcome the design challenges of physical implementation, asynchronous communication, and the feat of transmission and reception over

the wireless channel. While IR communication proved to be a challenging medium to work with, the achieved reliability in spite of this speaks as a testament to the robustness of our design. The circuit seamlessly interfaces analog and digital components and cleverly utilizes the Super Nintendo protocol. Our implementation successfully mimics the behavior of a wired controller with the added benefit of range and aesthete.

Appendix I - List of Components

Component	Model/Value	Corresponding Subcircuit	Number of Instances
Timer	7555	Controller Interface	1
		Receiver Interface	1
		Console Interface	1
PLD	ATF22V10C	Controller Interface	1
		Receiver Interface	1
		Error Detection	2
		Console Interface	1
NPN	2N4400	Transmitter	2
Voltage Regulator	78L05	Controller Interface	1
		Receiver	1
		Receiver Interface	1
Quad Op Amp IC	LM324	Receiver	4
Op Amp	LF411	Receiver	1
2:1 Multiplexer	CD4053	Receiver	3
		Console Interface	2
SIPO Shift Registers	CD4094	Receiver Interface	2
Buffer	CD4503	Receiver Interface	1
4-Bit Counter	CD4029	Receiver Interface	1
		Console Interface	2
PISO Shift Registers	CS4035	Console Interface	3
Inverter	CD4069	Receiver Interface	1
IR Emitter	10°, 250 mW	Transmitter	1
Phototransistor	08IRTR	Receiver	8
LED	–	Controller Interface	1
		Receiver	1
Stereo Connector	3.5 mm	Receiver/Receiver Interface	2
		Console Interface	1
SNES Controller	–	–	1
SNES Console	–	–	1
9V Battery/Battery Pack	–	Controller Interface/Transmitter	1
AC-DC Adapter/Connector	10V	–	1

Appendix II - WinCupl Code

7.1 Controller Interface

```
Name     ControllerInterface ;
PartNo   00 ;
Date     3/17/2018 ;
Revision 01 ;
Designer BrianFrost ;
Company  CooperUnion ;
Assembly None ;
Location NYNY ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 1    = Clk;
PIN 2    = CONTROLLER;

/* *************** OUTPUT PINS *********************/
PIN 15    = LATCH;
PIN 16    = OUTPUT;
PIN 17    = Q5;
PIN 18    = Q4;
PIN 19    = Q3;
PIN 21    = Q2;
PIN 22    = Q1;

/* Counter Logic */

Q5.OE = 'b'1;
Q4.OE = 'b'1;
Q3.OE = 'b'1;
Q2.OE = 'b'1;
Q1.OE = 'b'1;

Q5.d = !Q5;

Q4.d =
Q4 & !Q5
# !Q4 & Q5;

Q3.d =
Q5 & Q4 & !Q3
# Q3 & !(Q5 & Q4);

Q2.d =
!Q2 & Q3 & Q4 & Q5
```

```
#Q2 & !(Q3 & Q4 & Q5);

Q1.d =
!Q1 & Q2 & Q3 & Q4 & Q5
#Q1 & !(Q2 & Q3 & Q4 & Q5);

Q1.ar = 'b'0;
Q1.sp = 'b'0;
Q2.ar = 'b'0;
Q2.sp = 'b'0;
Q3.ar = 'b'0;
Q3.sp = 'b'0;
Q4.ar = 'b'0;
Q4.sp = 'b'0;
Q5.ar = 'b'0;
Q5.sp = 'b'0;

LATCH = Q1 & Clk & !Q2 & !Q3 & !Q4 & Q5;

OUTPUT =
Q1 & CONTROLLER
# Q1 & !Q2 & !Q3 & !Q4 & !Q5;
```

7.2 Receiver Interface

```
Name     ReceiverInterface ;
PartNo   00 ;
Date     3/23/2018 ;
Revision 01 ;
Designer BrianFrost ;
Company  CooperUnion ;
Assembly None ;
Location NYNY ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 1    = Clk;      /* From FLIPCLOCK to clock ffs */

PIN 3   = REGCLOCK; /* Fed from 7555 */
PIN 4    = F0;
PIN 5   = F1;
PIN 6    = F2;
PIN 7    = F3;

PIN 11    = DATA;
```

```
/* *************** OUTPUT PINS *********************/

PIN 14    = T0;
PIN 15    = T1;
PIN 16    = T2;

PIN 21    = RESETS;
PIN 22    = CLKOUT;
PIN 23    = FLIPCLOCK;

/* LOGIC */

T0.OE = 'b'1;
T1.OE = 'b'1;
T2.OE = 'b'1;

T0.d = !T2 & !T0 & !DATA
# T2 & !T1 & !T0 & !DATA
# T2 & T1 & !T0 & DATA;

T1.d = !T2 & !T1 & T0 & !DATA
# !T2 & T1 & !T0 & !DATA
# T2 & !T1 & T0 & !DATA
# T2 & T1 & !T0;

T2.d = !T2 & T1 & T0 & !DATA
# T2 & !T1 & !T0 & !DATA
# T2 & !T1 & T0 & !DATA
# T2 & T1 & !T0;

RESETS = F3 & F2 & F1 & F0
# !(T2 & T1 & T0);

FLIPCLOCK = !T2 & REGCLOCK
# T2 & !T1 & REGCLOCK
# T2 & T1 & !T0 & DATA
# T2 & T1 & T0 & RESETS;

CLKOUT = T2 & T1 & T0 & REGCLOCK;

T0.ar = 'b'0;
T0.sp = 'b'0;
T1.ar = 'b'0;
T1.sp = 'b'0;
T2.ar = 'b'0;
T2.sp = 'b'0;
```

7.3 Console Interface

```
Name     ConsoleInterface ;
PartNo   00 ;
Date     4/6/2018 ;
Revision 01 ;
Designer BrianFrost ;
Company  CooperUnion ;
Assembly None ;
Location NYNY ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 1    = Clk;      /* 300kHz */
PIN 2   = T2; /*                  */
PIN 3   = T1; /* From prior 22v10 */
PIN 4    = T0; /*                  */
PIN 5   = M;  /* MSB of Counter */


/* *************** OUTPUT PINS *********************/
PIN 16   = CLKMUX;
PIN 17   = RST;
PIN 18   = Z0;
PIN 19   = Z1;
PIN 20   = STROBE;
PIN 21   = SIPOMUX;
PIN 22   = CTRESET;
PIN 23   = PAR;

/* ************** LOGIC ***************************/
Z0.OE = 'b'1;
Z1.OE = 'b'1;
RST.OE = 'b'1;
Z0.ar = 'b'0;
Z0.sp = 'b'0;
Z1.ar = 'b'0;
Z1.sp = 'b'0;
RST.ar = 'b'0;
RST.sp = 'b'0;

STROBE = !Z1 & Z0;
SIPOMUX = Z1 & !Z0;
CTRESET = !Z1 & !Z0;
PAR = Z1 $ Z0;

Z0.d = (!Z0 & !Z1) & !(T2&T1&T0)
```

```
# (Z1 & !Z0)
# (Z1 & Z0) & !M;

Z1.d = (!Z1 & Z0)
# (Z1 & !Z0)
# (Z1 & Z0) & !M;

RST = !(T2 & T1 & !T0);

CLKMUX = !Z1 & Z0;
```

7.4 Error Detection I

```
Name     ERRORDET1 ;
PartNo   00 ;
Date     4/7/2018 ;
Revision 01 ;
Designer BrianFrost ;
Company  CooperUnion ;
Assembly None ;
Location NYNY ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 2    = Rin;
PIN 3    = Lin;
PIN 4    = Xin;
PIN 5    = Ain;
PIN 6    = RIGHTin;
PIN 7    = LEFTin;
PIN 8    = DOWNin;
PIN 9    = UPin;
PIN 11   = HI;

/* *************** OUTPUT PINS *********************/
PIN 15    = GARBAGE;
PIN 16    = UPout;
PIN 17    = DOWNout;
PIN 18    = LEFTout;
PIN 19    = RIGHTout;
PIN 20    = Aout;
PIN 21    = Xout;
PIN 22    = Lout;
PIN 23    = Rout;

GARBAGE = !UPin & !DOWNin
```

```
# !LEFTin & !RIGHTin
# !HI;

UPout = UPin # GARBAGE;
DOWNout = DOWNin # GARBAGE;
LEFTout = LEFTin # GARBAGE;
RIGHTout = RIGHTin # GARBAGE;
Aout = Ain # GARBAGE;
Xout = Xin # GARBAGE;
Lout = Lin # GARBAGE;
Rout = Rin # GARBAGE;
```

7.5 Error Detection II

```
Name     ERRORDET2 ;
PartNo   00 ;
Date     4/7/2018 ;
Revision 01 ;
Designer BrianFrost ;
Company  CooperUnion ;
Assembly None ;
Location NYNY ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 2    = GARBAGEin;
PIN 3    = STARTin;
PIN 4    = SELECTin;
PIN 5    = Yin;
PIN 6    = Bin;

/* *************** OUTPUT PINS *********************/
PIN 15    = Bout;
PIN 16    = Yout;
PIN 17    = SELECTout;
PIN 18    = STARTout;

Bout = Bin # GARBAGEin;
Yout = Yin # GARBAGEin;
SELECTout = SELECTin # GARBAGEin;
STARTout = STARTin # GARBAGEin;
```