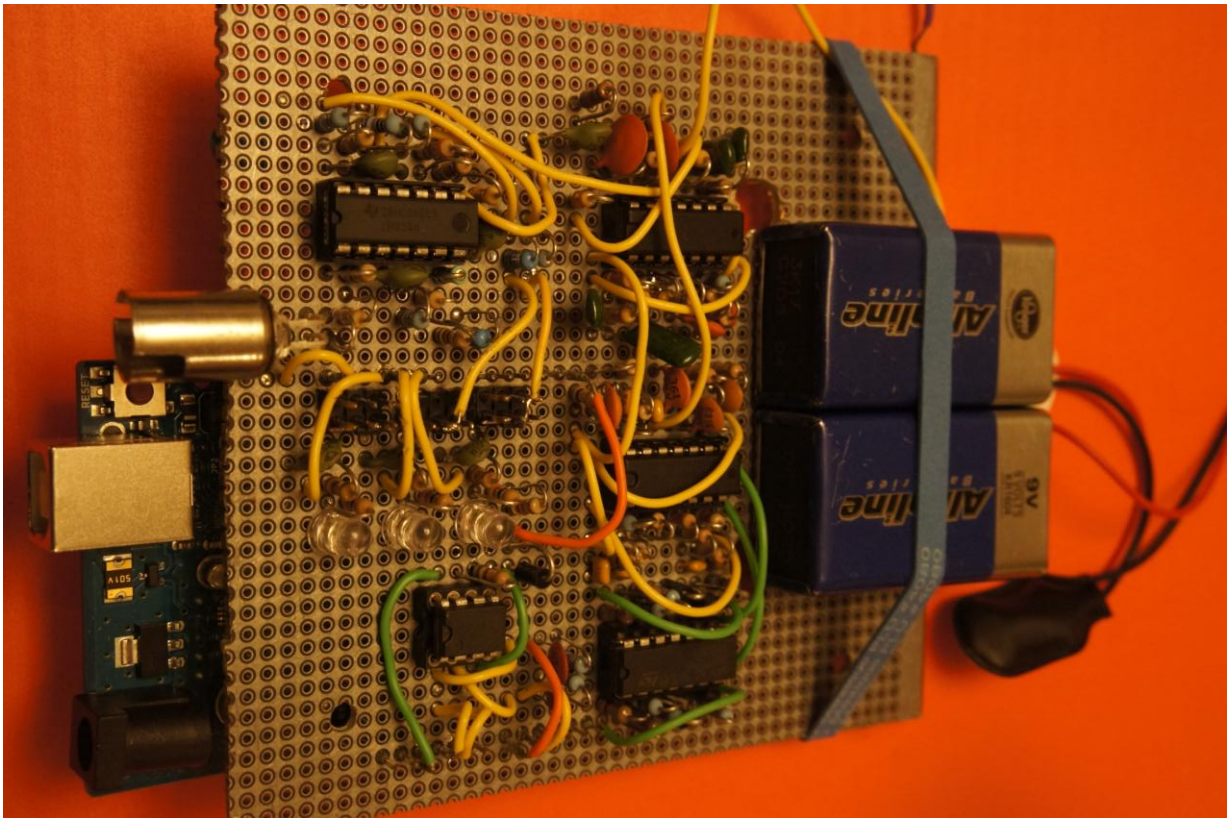
Optical FSK Receiver

March 2014

Alex Vassallo



1 Abstract

This project was created as a final project for EE 205 Signal Conditioning class at the University of Washington. The assignment was to design an optical receiver using a photodiode to collect transmitted data from light that is being transmitted from a central base. The light source pulsed at either 3kHz to indicate 0, or 5kHz to indicate 1, and at various bitrates. The embedded data contained ASCII characters which could be displayed as text on a console via USB connection. The microcontroller we were required to use was an Arduino Uno, which has an analog sampling frequency just below the Nyquist frequency for a 5kHz signal. The biggest challenge of the assignment was to detect that 5kHz signal. In response, I chose to develop a much larger front end circuit to perform the frequency detection, which used filters, peak detectors, and automatic gain control to convert the analog signal back into digital. Once I had a digital signal output to process, I was able to receive data at speeds well above expectations. By monitoring voltage on the peak detectors the software controlled MOSFET gain control circuits. The sensitivity could be automatically adjusted to reach very long distances, while still functioning at close range.

2 Hardware

2.1 System Schematic

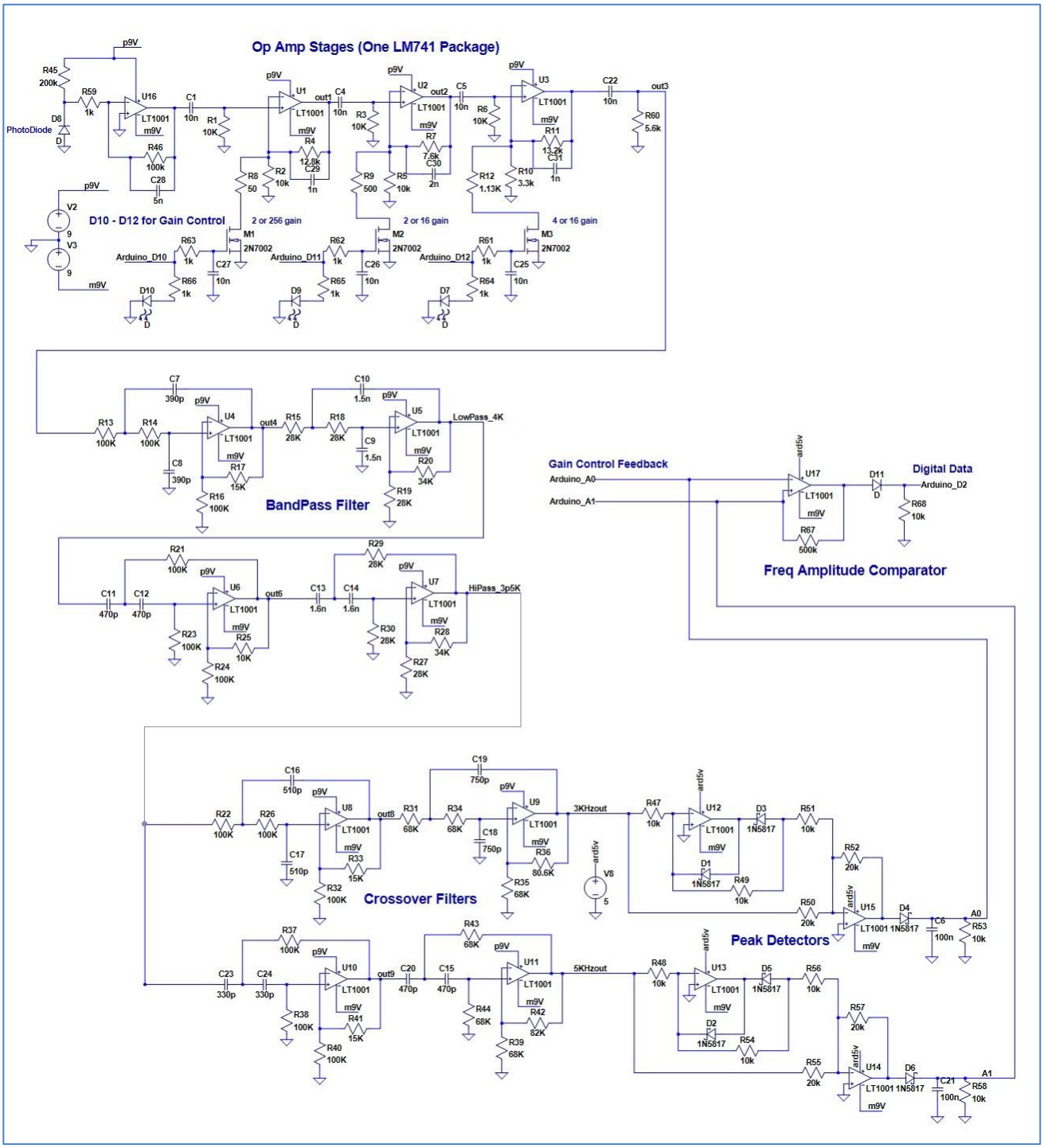


Figure 1: System Schematic

2.2 Variable Gain Stage

The first stage is a trans-impedance amplifier to respond to current changes from the photodiode. This amplifier feeds the next three variable gain stages, which are described in Figure 2. Each variable gain stage is designed to roll-off frequencies above 10kHz using C1. The coupling capacitor C2 between each variable gain stage eliminates DC offset and provides a high pass filter. Each op-amp stage works in conjunction with a MOSFET that changes feedback resistance to raise or lower the gain. Each stage has been configured to allow switching between a gain boost of 4^x. By using gains of 4^1, 4^2, and 4^4, any binary combination can be used to allow various gains up to 4^7. Gain is controlled from three digital outputs from the micro-controller.

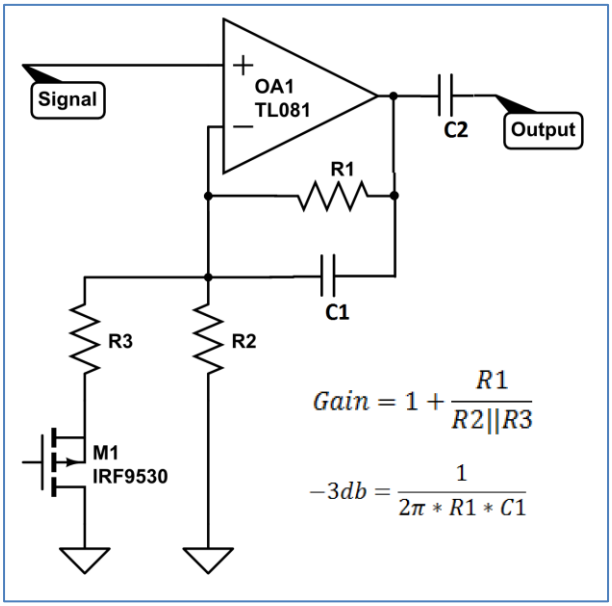


Figure 2: Variable Gain Stage

2.3 Band-pass filter

The next amplifier stage is a band-pass filter, which passes frequencies between 3kHz and 5kHz, and attenuates frequencies outside that range. A 4 pole butterworth filter design as shown in figures 3 and 4 are used for high and low pass filtering of the band-pass. The -3dB points needed to balance the amplitudes of a 5kHz an 3kHz signal for comparison were experimentally discovered using simulators. The calibrated roll-off frequencies were found to be 4kHz for the high pass and 3.5kHz for the low pass. Since they slightly overlap, there were symmetric and steep response slopes on both sides of the 4kHz center frequency, giving balanced band-pass characteristics

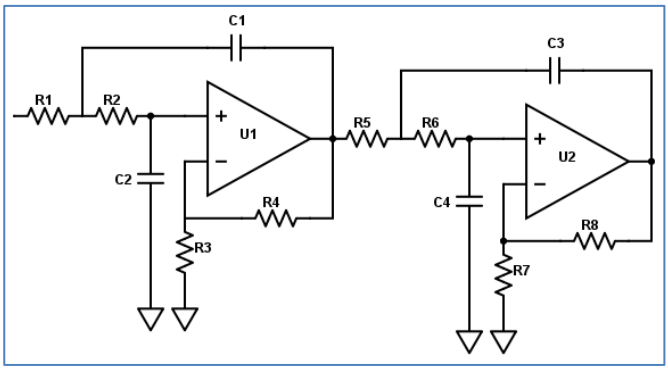


Figure 3: Low Pass Butterworth

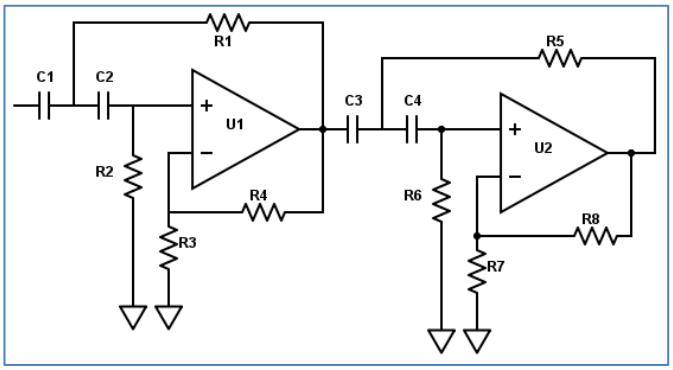


Figure 4: High Pass Butterworth

2.4 Peak Detection

Two more 4-pole butterworth filters, one high and one low pass, separate the 3kHz and 5kHz sinusoidal signals into separate channels. Each filter rolls off right at 4kHz in opposite directions to split our frequency range right down the middle. By connecting each filter output to a full wave precision rectifiers we can measure the levels of each frequency within the signal. Using a full wave rectifier allows double the amount of positive waveform samples to feed into our peak detector, which allows more samples to be used with faster sampling times.

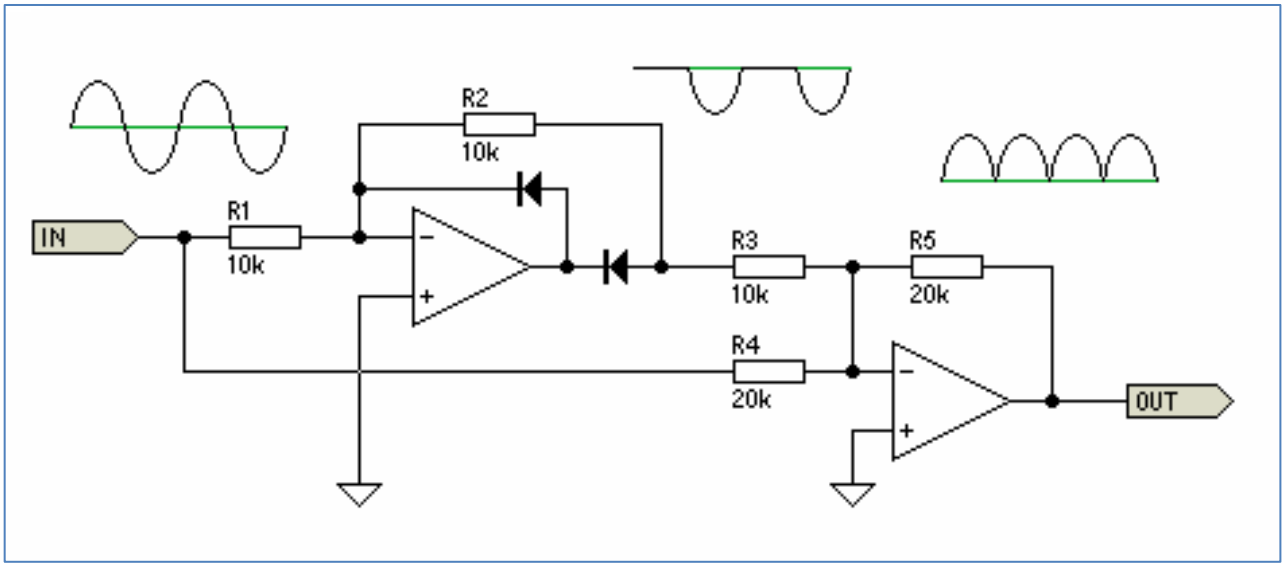


Figure 5: Wave Rectifier

The peak detectors operate by using the wave rectifier output to charge up C1 and C2, which are continuously discharging through R1 and R2 to ground. The charging and draining of C1 and C2 must support bitrates up to 1kHz, which means our sample period lasts 1ms and capacitors should be able to discharge 60% within half of the bit period. A comparator measures the voltage of C1 and C2 and outputs high or low accordingly.

Formula to calculate values of R1, C1, R2, and C2:

$$V = V_o e^{-(\frac{t}{RC})}$$

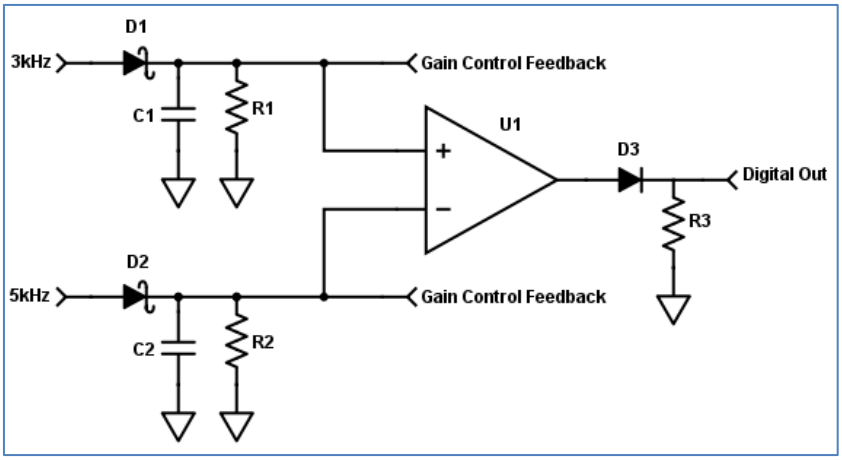


Figure 6: Peak Detector

Automatic gain control is accomplished by monitoring the voltage levels of C1 and C2. Whichever voltage is higher is used to determines the gain adjustments required. If the highest voltage drops below 0.5V, the gain is increased by a factor of 4, bringing the voltage up to 2V. Similarly, if the voltage is above 3.5V, the gain is reduced by a factor of 4 to bring it down to 1.2V. The scaling factor of 4 was chosen exactly for this behavior.

The final comparator uses the 5V power rail and the -9V power rail from the batteries to output the proper voltage level. A diode is used to block the negative voltage for our square wave and a resistor is used to pull the data pin to ground when the diode is not conducting. The final result of the hardware is a clean digital signal that represents the strongest frequency detected by our photodiode.

3 Software

3.1 Auto-gain

Since the hardware can raise and lower the gain in smooth linearly steps, the software can use an integer value to control the volume of the gain boost. Just like a binary system, the amplification factor of each gain stage increases exponentially to represent its bit position. The 3 gain switches can be arranged into a 3 bit integer, of values 0 to 7, that can be managed by the software.

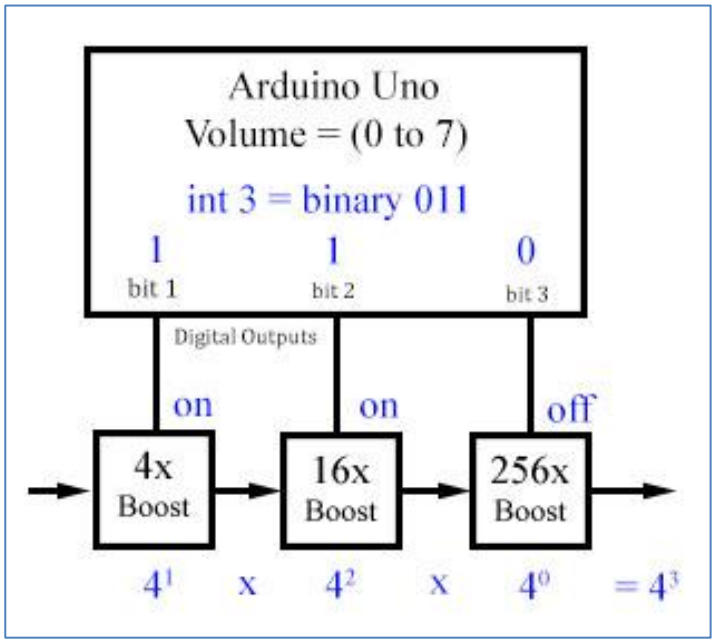


Figure 7: Auto-gain

To determine if the volume, or gain, is out of range, the software must monitor two analog outputs from the peak detectors. The highest value of the two is compared against predefined range limits, (0.5V to 3.5V), and the volume is incremented or decremented accordingly. The response time of the gain control is less than the loop cycle time of the software, so volume can be adjusted by 1 each loop cycle. LEDs connected to each binary output show the currently selected binary combination, thus the current level of gain boost.

3.2 Synchronization

To measure and synchronize with the incoming data stream, an auto-configuration function was built. This took the most time to develop. It begins by attaching interrupt routines to both falling and rising edge detectors. Since both edges cannot be monitored simultaneously by the hardware, each interrupt routine must enable the other in a back-and-forth fashion. This allows those interrupt routines to mark the exact times of each data-line voltage transition.

Figure 8 shows the equation style we use to find the bit time. The unknown variables are both x and the coefficients. Knowing the format of the byte message, we can use a "best fit" algorithm to find the coefficient variables by 1) assigning x to the smallest time found, 2) measuring time between bits and bytes, 3) verifying start and stop bits, and 4) using previously acquired synchronization values. If the coefficient variables are not whole numbers, we can use that variation to simply measure the error or drastically change the timing values.

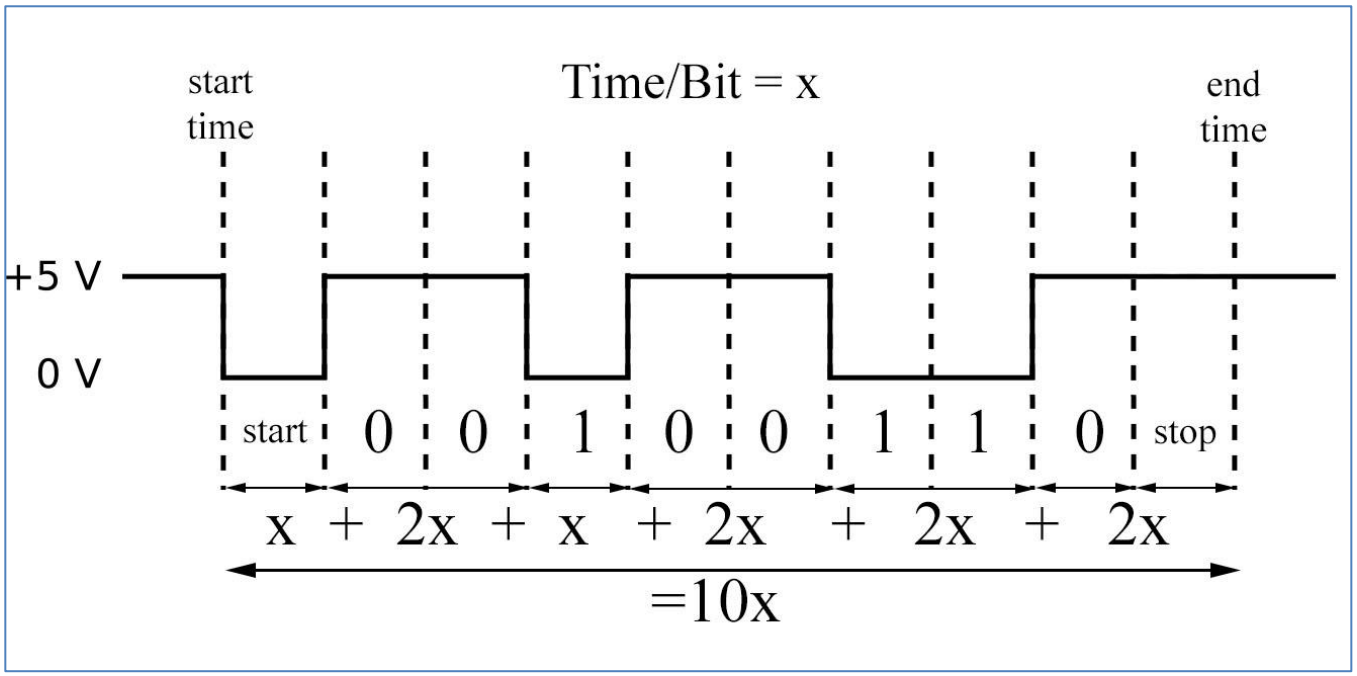


Figure 8: Synchronization

3.3 Performance

Using this hardware and software package, I was able to easily reach the 1kHz bit rate target. The maximum range achieved before auto-configuration could no longer detect a signal was over 10 feet with this design. At that range, physical alignment seemed to be the most difficult problem. The single LED transmitter station was extremely omni-directional. During the final test in the lab, I was able to receive 1000 messages with a few minor errors at 10 feet. The "Miss Rate" was less than 1% and hard to measure.

Auto-configure routines were able to seek bit rates as low as 20Hz, and above 1kHz. The average packet misses while recalibrating was usually only 1, but up to 4 when large timing transitions took place. Overall the auto-configuration was very impressive. It can determine bit rates within an accuracy of 10 micro-seconds, total message size, and any extra or missing bits. It was also allowed to make minor adjustments to current variables to compensate for any drift. When the signal was lost or changed, it immediately began to measure the new bit time, verify the signal, verify the packet, and re-synchronizes with it.

3.4 Software Outline

Main Loop()

- senseGain()
- readData()

senseGain()

- acquire average of highest peak detector voltage
- compare value to bounds, and adjust volume if outside
- Convert volume into 3 bit representation
- Enable or disable gain boost pins from the 3 bit values.

readData()

- Verify it is time to sample a bit
- Sample the bit and set next sample time
- If bit = 1 AND is the first bit after an idle period, reset variables to begin collecting data (start bit)
- if bit = 0 AND we have entered an idle period AND we have collected data...
    - if data is the correct known size and format then print data and sync minor timing variance
    - else call autoSyncData()
- if valid data is detected...
    - align bit into byte value
    - if bit = 1, use timing marks to track smallest bit times detected (autoSync helper)
    - store completed bytes into array

autoSyncData()

Interrupt Routines:

fallDetected()

- measure time since last rise occurred
- set next bit sample time, 1/3 into bit period.
- set data start flags if line was just idle.
- enable interrupt for riseDetected()

riseDetected()

- mark time of rise occurring
- enable interrupt for fallDetected()

4 Photos