Remote Control Demodulation

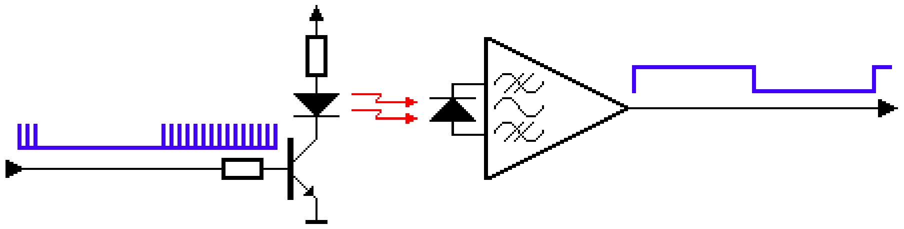
Constructed by Nabil El-Hinnawy

Goal

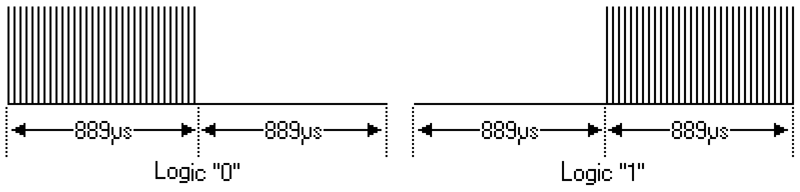
The goal of this project was to create an autonomous circuit that would demodulate an infrared remote control signal into the pure binary code that is sent when you press each button. The original design also included a display circuit that would display each binary code as the button you pressed. This portion was cut out due to time constraints and problems encountered during the building stage of the circuit.

Background Information

All remote control producers have their own method of encoding the information. Almost all remotes use a form of "digital" modulation, in which pulses are characterized by a wave packet of high frequency (as compared to the pulse frequency). A crude model is pictured below:



In certain remote controls, instead of using a positive voltage to represent a logic level "1," they use the transition from voltage 0 to a positive voltage to represent a logic level "1." They use the transition from a positive voltage to zero to represent a logic level "0." This form of coding is known as Bi-phase or Manchester encoding, where each high-low or low-high transition is known as a 'bit.'



Strategy

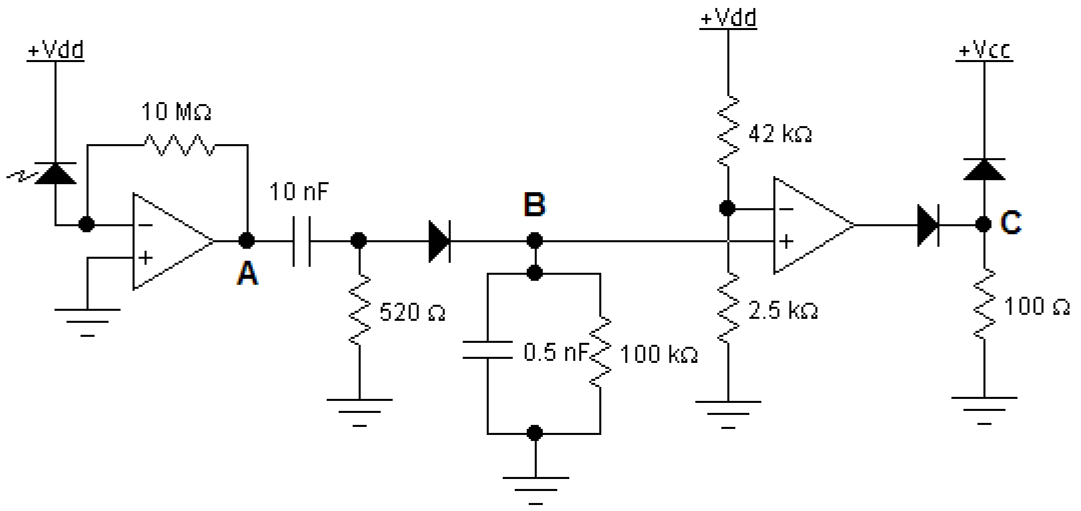
The strategy is to take the modulated infrared wave, break it down into the pure pulse signal, and then digitally process it into binary code. Thus, there are two phases, an analog breakdown, followed by a digital breakdown. But first, a few specifics on the remote control I used.

Specifics

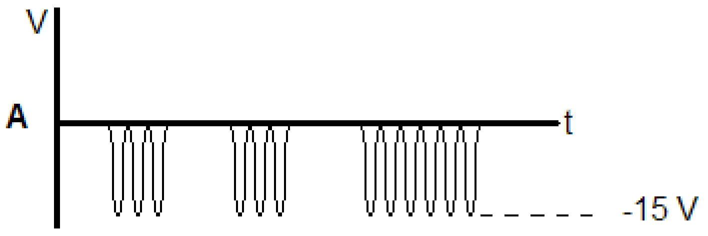
I used a Phillips Magnavox VCR remote control, which specifically had a carrier wave frequency of 36 kHz, and a pulse wave frequency of 1 kHz. The bit length was almost 2ms exactly, making each high or low length 1ms. All Phillips remote controls come with two logic "1" starting bits at the beginning of each signal, and contain 14 bits of data.

Analog Breakdown

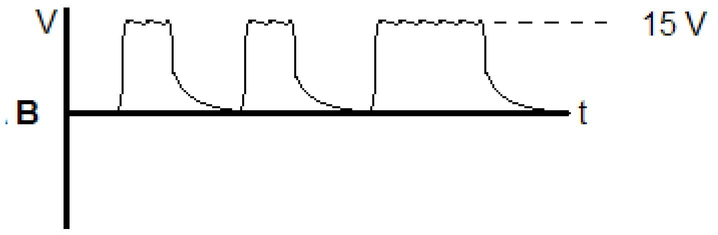
I designed this circuit to breakdown our signal into the pure pulse code:



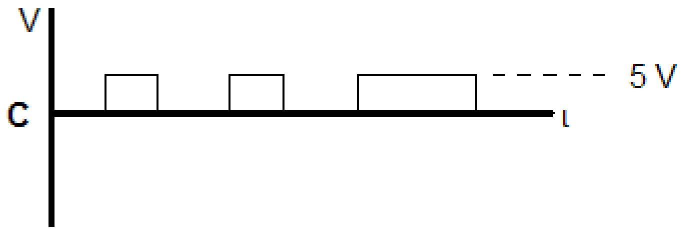
The circuit flows from left to right. The photodiode picks up the infrared signal, and sends it to an inverting amplifier with a huge gain, which we need to improve the efficiency of the rest of the circuit. The signal at point A is shown below:



From there, we send the signal into a high pass filter, with a cutoff frequency of about 30kHz, to cut out any lower frequency noise or dc current that would come from constant light sources (overhead lights). This also has the added effect that it makes this negative voltage signal into a positive voltage, so it can be fed into the demodulator. The demodulator consists of a diode which rectifies the signal and an RC component which has a time constant slow enough that it hides the transitions from the 36kHz wave, but fast enough that it allows the 1kHz pulse thru with no demodulation. The time constant here is 50 µs, which corresponds to a frequency of 20kHz. The output at point B (after the demodulation) is:

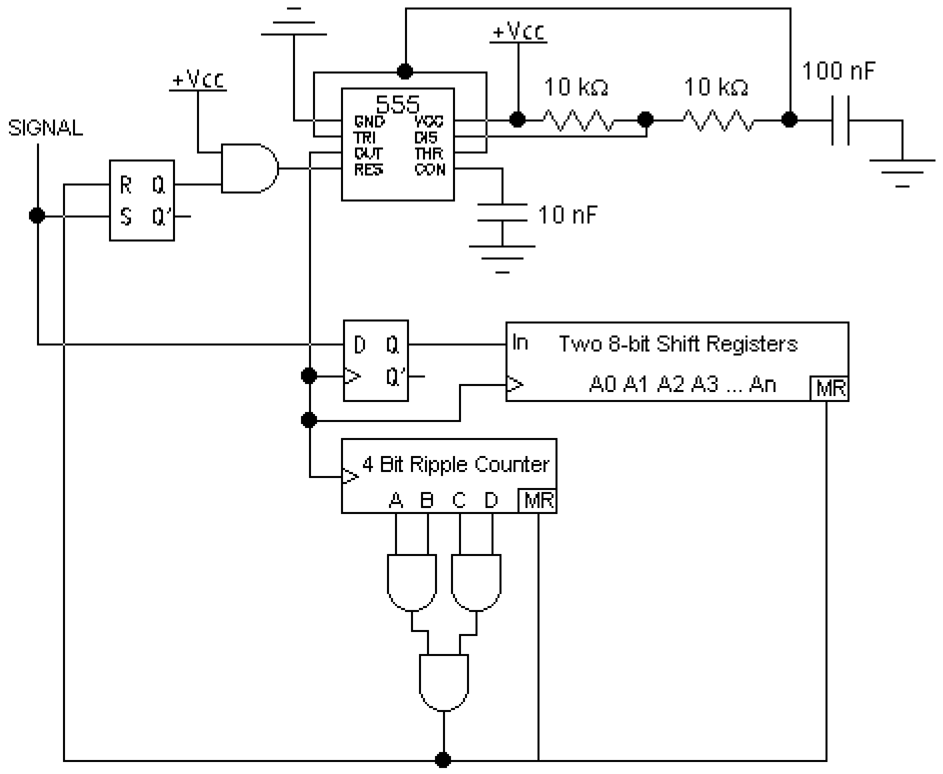


This signal is now in the basic state where one can look and see the code trying to come in. The only thing left to do is clean up the signal, and convert it from 15 volts to 5 volts so it will be TTL compatible. This is why the signal now goes into a comparator set to a threshold voltage of about .5 volts. A more elegant solution would be to use hysteresis on this op amp, but for this circuit, this simple solution works. We run the clean 15V signal through a forward biased diode to rectify the negative swing of the pulse train. The signal is then connected to a forward biased diode that leads to a 5V supply. The reason for this is that any voltage above 5.6V will now be flooded to the power supply. Now our wave has a range of approximately 0V to 5V. I connect a resistor of 100Ω in order to make the output of this circuit TTL compatible with the digital chips in the next section. The output of the analog breakdown is now the pure, clean, pulse code containing the command:



Digital Breakdown

I designed this circuit to process the pure pulse code of the IR remote control:

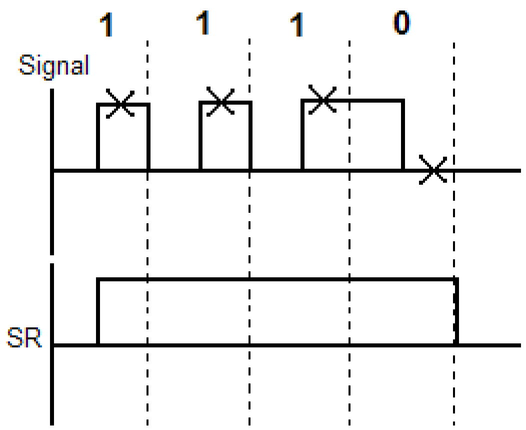


We now have the pure code coming into this step of the circuit. Since each command will be lead with two logic “1” bits, it reduces to 2 pulses started every command. The basic idea is to design a circuit which starts when it sees a pulse, and resets after the last bit has been processed. I accomplished this in the following manner. Instead of reading the transition of each but, I can just read the second half of each bit, and pass that voltage level into a shift register, using a clock that has the same frequency of the bit length.

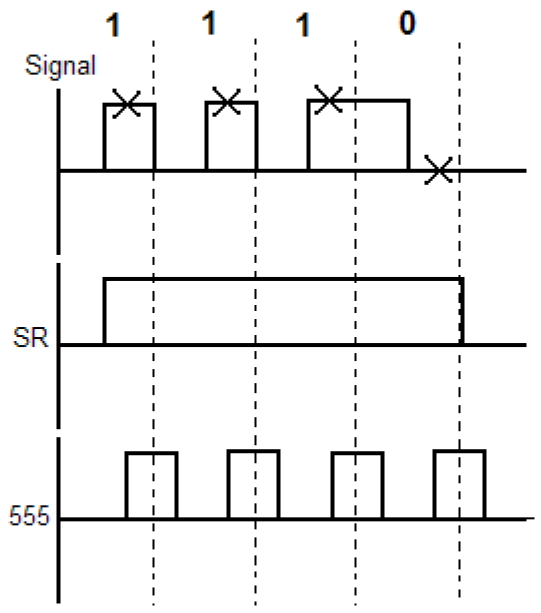
The first step was to send the signal into a an SR flip flop that would stay high after the first pulse of the starting bits reached the chip, and reset to low after the reset was momentarily held high. To do this, I built my SR out of NOR gates, so that the 00 element would be the memory element, and not force Q and ~~Q~~ to 11 when SR is 00. A truth table of the NOR SR flip flop is below:

S	R	Q	~~Q~~
0	0	mem	
0	1	0	1
1	0	1	0
1	1	1	1

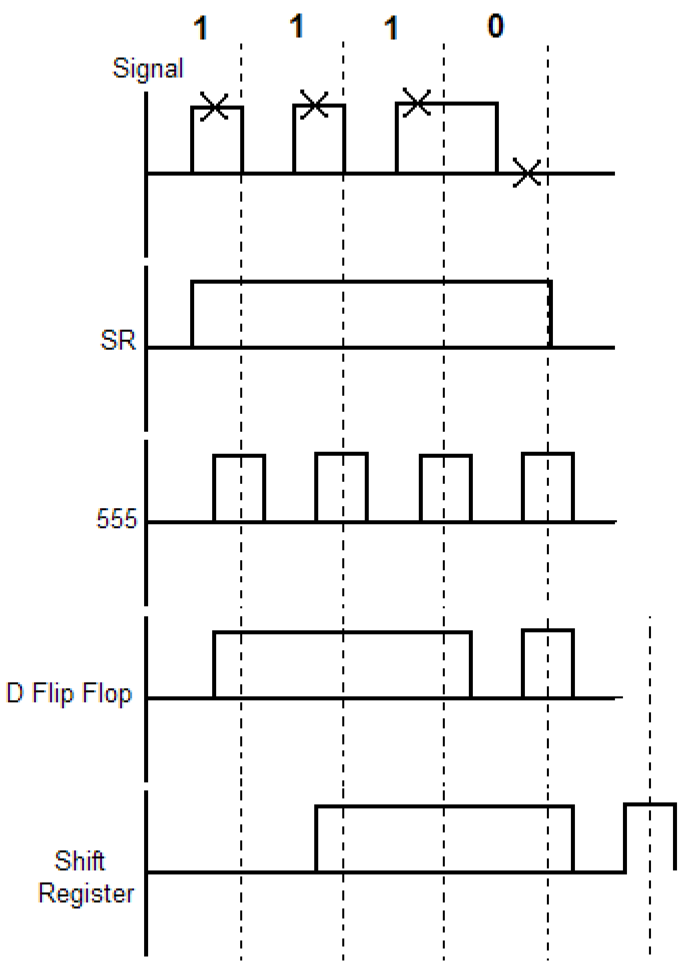
The signal and SR output for a fictitious 4 bit signal is below:



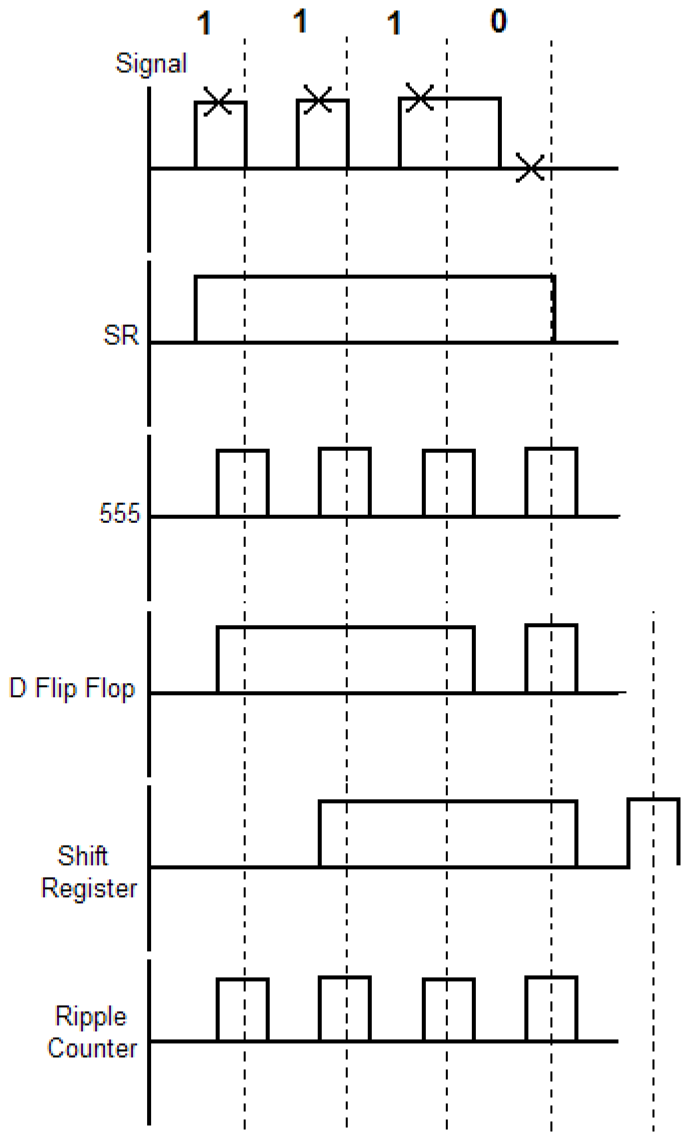
The reason the SR shuts off will be apparent later. The next step is to have the Q output fed into an AND gate with one input held high, so whenever the SR is turned on, the AND gate will be 1. The AND gate is fed into the RESET pin of a 555 timer. When the AND gate is low, the 555 is grounded, and has no output. When the AND gate is high, the 555 is active. The 555 chip is set to asynchronous pulse of period 2ms, exactly the bit length of our data signal. So when the SR is turned on, it now turns on the 555 clock which has a slight delay on its first pulse, which works in our favor. The timing of the signal, SR, and 555 is shown below:



The 555 is then used as the clock for the last 3 portions of the circuit, the ripple counter, the D flip flop, and the shift register. The idea for this last 3 portions is that the 555 will read the exact portion of the signal that we need to read (as indicated by the X marks on the graphs) so we feed the signal into a D flip flop, which clocks over the signal on ever 555 pulse. The clocked over bit is next clocked into a shift register, which hold the data. It occurred to me after I built the circuit that I did not actually need the D flip flop, but I kept it in my circuit anyway because it was yielding the right answer. Theoretically, this is where we could stop, because we have our signal now converted to binary code on the shift register as documented below:



The only problem is to find a way to reset the circuit so one can input multiple commands and have each one processed by the circuit. So the idea is to get a single pulse sent to the Reset of the SR flip flop. To do this, we incorporate a 4 bit ripple counter. Because we have 14 bits coming through, we will have at most 14 pulses being read. Therefore, if we send the outputs of each digit on the ripple counter into AND gates, and feed those AND gates into one more AND gate, when the shift register reads "1 1 1 1" corresponding to the binary number 15, or the 16th state, it will trigger a pulse. We then feed this signal into the reset of the SR flip flop, which will close the 555 timer, and hence the D flip flop. We also feed this signal into the master reset of the ripple counter. Although it is generally a bad idea to have a ripple counter triggering its' own reset, I can get away with it in this case because the propagation delays of the AND gates will cause me to bypass the error. The timing diagram is below:



So to summarize the digital processing portion of the circuit, we have the clean command code coming in, which triggers an SR flip flop. This SR flip flop triggers a 555 timer with the exact frequency of our data signal. We then feed the signal into a D flip flop which is clocked from the 555 timer. The output of the D flip flop then sends the output it read off the signal to a shift register, which is also clocked from the 555. The 555 pulses are also sent to a 4-bit ripple counter, and when the counter reaches its last state, it triggers its own resent as well as the reset on the rest of the circuit, thus preparing the circuit for the next command.

Application

This solution seems quite possible in a theoretical frame, but it is missing key elements in the practical application, which I only discovered with the help of Professor Robinson and our TA Patrick Marks. For one, the propagation delay is not enough to reset itself. It is of the order of nanoseconds, and the period is of the order milliseconds, so it is too quick. I would have to use a 555 timer used as a one shot in order to create the reset pulse. In this application, I must pull out a resistor tying the presets to the counter every time I want to reset the circuit. Also, my circuit pushes the limits of TTL compatibility by scarcely being with the current ranges. Had I built this circuit with CMOS chips, I would have had more immediate success, but I would have had to take greater precaution in the analog to digital conversion, and figured out a different way to display my data. I also found some problems with the fact that certain 555 timers and 7493 chips were falling edge triggered. This caused me great problems in the production phase because it was picking up the wrong bits of the data signal.

Conclusion

The final product worked well considering the elementary design of the circuit. The major flaw in my design step was the assumption that I would be dealing with ideal power supplies, ideal chips, instantaneous propagation times on certain gates, while delayed propagation times through others. I feel if I had another crack at this project, I would make use of such tools as the phase lock loop and CMOS circuitry. But being taught electronics from a physics point of view (as well as the rest of my education) I feel I am inherently trained to figure out theoretically solutions to problems, ignoring the fact that it might be impossible to apply the solution. It is in this regard that I learned the most important step in designing a circuit isn't the theoretical breakdown of the task at hand, but rather to manage the flow of data from step to step without losing any information.