HARVARD

SCHOOL OF ENGINEERING AND APPLIED SCIENCES

ES52 Final Report

Frank DuBose
Gregory Hewett
Antuan Tran

Professor David Abrams
Preceptor Avinash Uttamchandani

Introduction

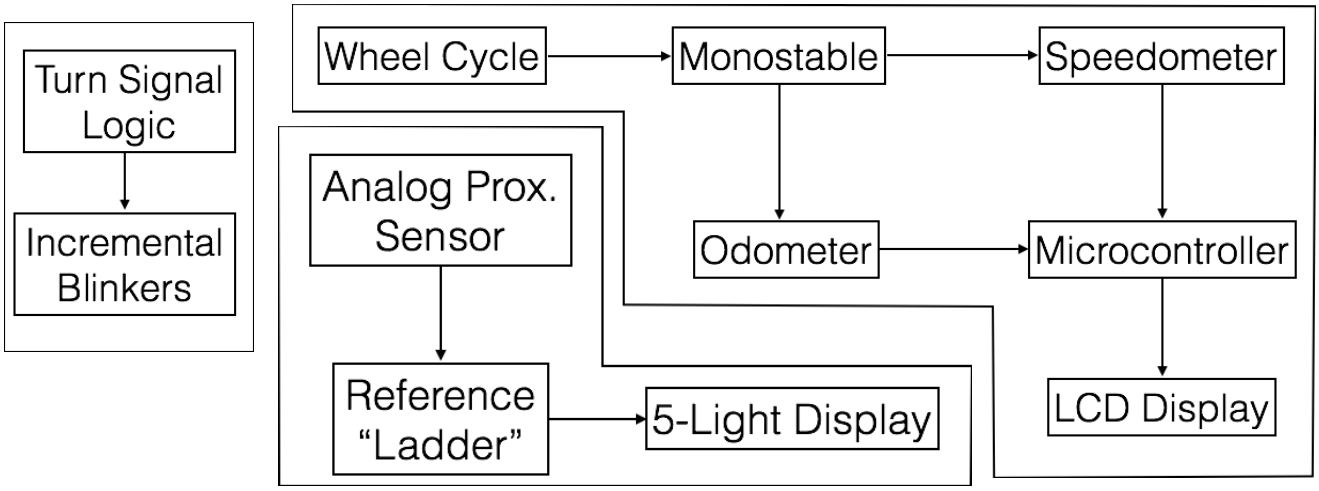
Being self-aware while riding a bicycle is arguably the most important practice to implement in order to ride safely and skillfully. Often times cyclists have little to no idea how fast they are going, and communication between cyclists, pedestrians, and drivers can be extremely poor on today's roads and sidewalks. The goal of our team's ES52 Final Project is to help alleviate these problems and increase biker awareness. We believe that making bikers more aware of how they are riding and giving them better tools to communicate with the people around them will not only lead to safer roads and sidewalks, but will also give bikers the confidence and reassurance that they are riding safely and thoughtfully.

The Design

Systems Breakdown

Our system project consisted of five major units: turn signals, an odometer, a speedometer, a proximity sensor, and a microcontroller and display. The turn signals were built using digital logic combined with analog components to present an interface to the bike rider in which he or she could signal a left turn or right turn via two left and right switches. These turn signals were independent of other components of the project as far as circuit design is concerned, but we felt as if the signals integrated well the other elements of our project. The odometer was built primarily on digital components and allowed us to not only keep track of the number of rotations the wheel had spun, but also convert this number into miles and quarter-miles so that we could send this information to the microcontroller and eventually the display. The odometer received its input from the wheel via a reed switch-monostable configuration and outputted its data to the microcontroller. The next unit of the project, the speedometer, also received its input from the same monostable as the odometer; however, the speedometer used analog components to transform this signal into a voltage value that was proportional to speed that could be read by the microcontroller. Another major module of the project was the proximity sensor. This was a highly analog interface that allowed us to keep track of how close any foreign objects were to the left-rear of the bike. This signal from the proximity sensor interacted with lights to inform the biker of the closeness of any object, but it did not interact with any other major components of the project. As mentioned previously, one other major component of our project was the microcontroller and LCD display. We decided to keep the microcontroller's purpose to be purely taking in data from the odometer and speedometer and sending it to the display. In other words, we sought to perform all calculations and operations via circuitry and simply use the microcontroller and display to relay this information to the biker. Lastly, our project was designed to run off a 9V power supply, so we included a voltage regulator to keep a constant 5V source for the components of our project. The voltage regulator powered all of the components of our project.

Block Diagram:

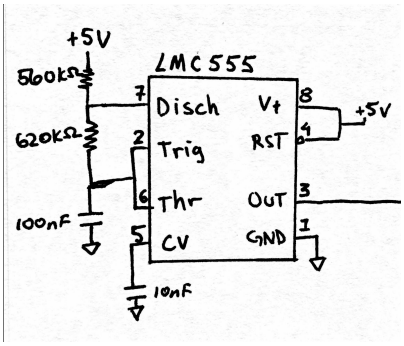


Sub-systems

A. Turn Signals

We wanted to implement useful and visually pleasing turn signals to the bicycle to allow the biker to better communicate with pedestrians, other bikers, and motorists. The usefulness is the innate virtue the turn signal, and to make them visually pleasing we decided on a cascading pattern of three LED's for both left and right signals. The turn signals consisted of an LMC555 clock, a 74HC161 counter (the 74HC163 could also be used), two SPDT switches for left and right along with 74HC10 NAND gates for the digital logic, and red LED's paired with VN2222 MOSFETs as the output turn signal.

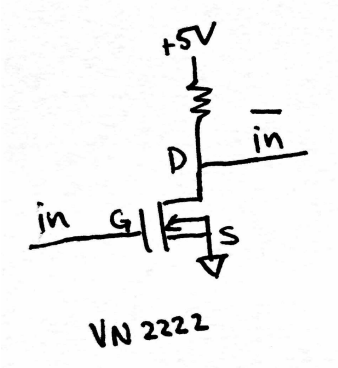
We decided on the clock frequency by considering the duration of the full cycle of the pattern. We considered two seconds for the entire cascading pattern to be the optimal duration that is clear to observers, so since we are using a 16 bit counter to create the pattern, the frequency must be 16 / 2 = 8 Hz. Additionally, since the duty cycle does not matter, we used the following familiar setup from lab to achieve an 8 Hz clock:



*The frequency is given by the following formula: f = 1.44/((R1 + 2R2)*C)*

We used two switches as input for the NAND gates, since we had single pole double throw switches but no single pole triple throw switches. However, using two switches allowed us to have both turn signals on at the same time, acting as hazard lights.

We used MOSFETS as inverters in the following way:



The counter and logic go together since the counter bits are the input to the logic gates. Since we wanted a complete cycle, we allowed the counter to count from 0000 to 1111, and used the output bits in the logic for turning the lights on and off. Originally, the design was to have one light constant, the second to turn on after 25% of the cycle, the third to turn on 25% after that, then the whole pattern to hold for the last 50%, then repeat. However, this did not look too visually pleasing, and since we had another unused NAND gate in the 74HC10 chip, we decided to make the first LED blink as well, but for the shortest amount of time. This way, the first LED turns on ⅛ into the cycle.

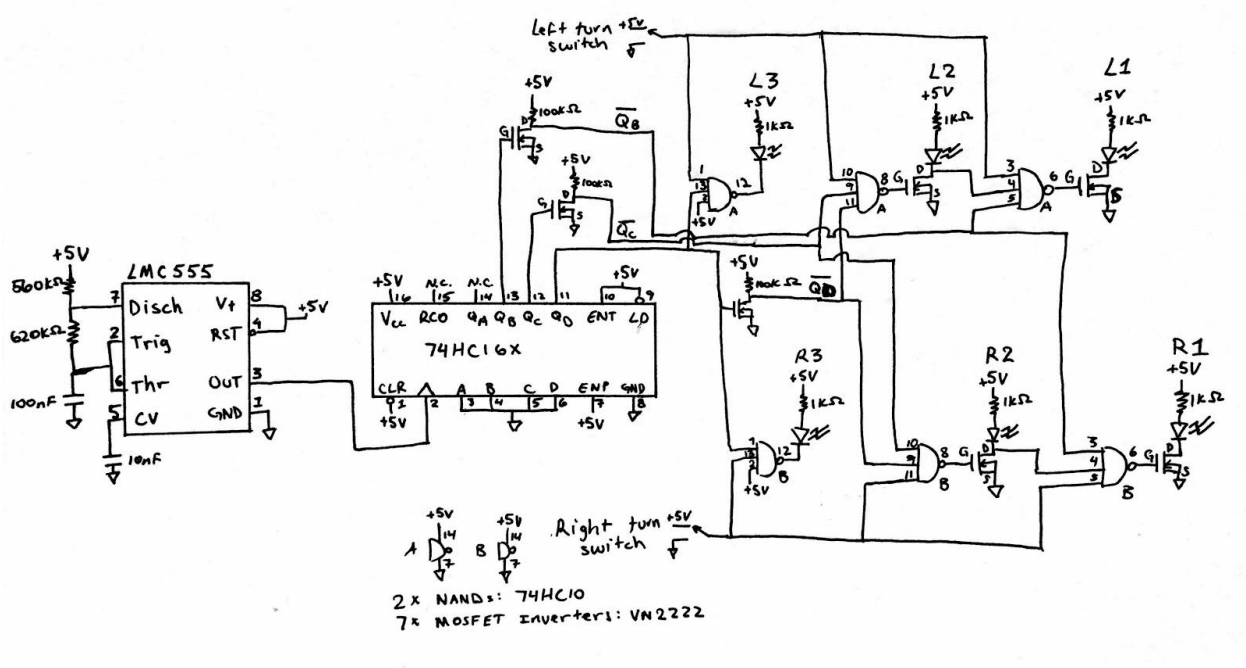
The logic was implemented in the following way: the last LED is on when the appropriate switch is on, and when the counter output Qd, the most significant bit, is 1. The output of the NAND of the switch and Qd is connected to the negative terminal of the LED. The positive terminal of the LED is connected to +5V through a resistor. The second LED is on when the switch is on, and either the Qd OR Qc bits are high. To make an OR gate out of a NAND gate, one must apply De Morgan's theorem and invert the inputs. Since we only want the Qc and Qd outputs to act in the OR gate logic, we did not invert the switch input. The negative terminal is connected to the the inverted output of the NAND gate and the positive is connected to +5V through a resistor. The first LED is on when any of the first 3 most significant bits are 1. From

LED 2 we have the output of Qd OR Qc, so for the input of the first LED we can use that output, the switch, and Qb. The LED is connected as in the middle one. This logic is identical for both left and right turn signals.

A side effect of the inverter to the MOSFET is that the first and second LED's remain on when the turn signal is switched off. This is because we did not want the switch to be included in the OR gate configuration, or else it would not function as expected: the LED would be on for all of the time the switch is on. However, this is not a problem since the on LED's function as brake lights that are always on as in a car.

We could have redesigned the LED's to turn on in an evenly timed pattern by rewiring the logic so that the first LED turns on when Qc is 1, the second when Qd is 1, and the last when both Qc and Qd are 1. However, we liked the visual of the exponential cascade and the lengthened duration of when all lights are on.

- The complete schematic of the turn signals:



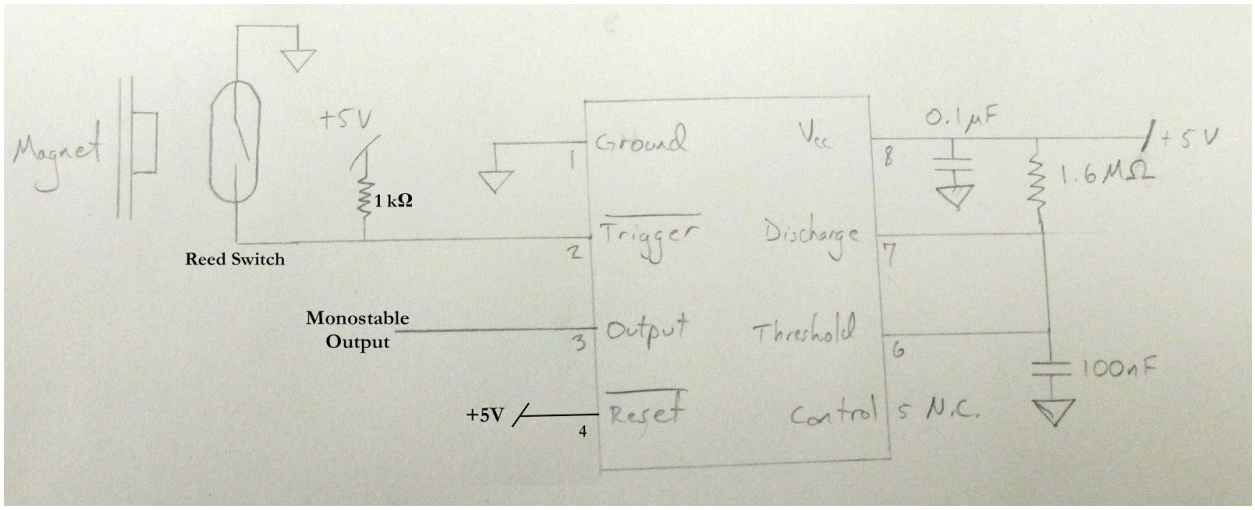
B. Odometer

One of the first ideas we thought to implement was an odometer and a speedometer. We knew that the backbone of these instruments would be counting wheel rotations, so we had to find a mechanism that could do this. Our first thought was to use a switch that opened and closed as the wheel spun, sending a pulse to our circuit for each rotation. We began researching different types of switches and discovered Hall effect sensors and reed switches. While both of these react to magnetic fields, we decided to use a reed switch as it seemed to be a simpler and more suitable solution for us as its only operation is that it closes in the presence of a magnetic field and remains open when no field is present.

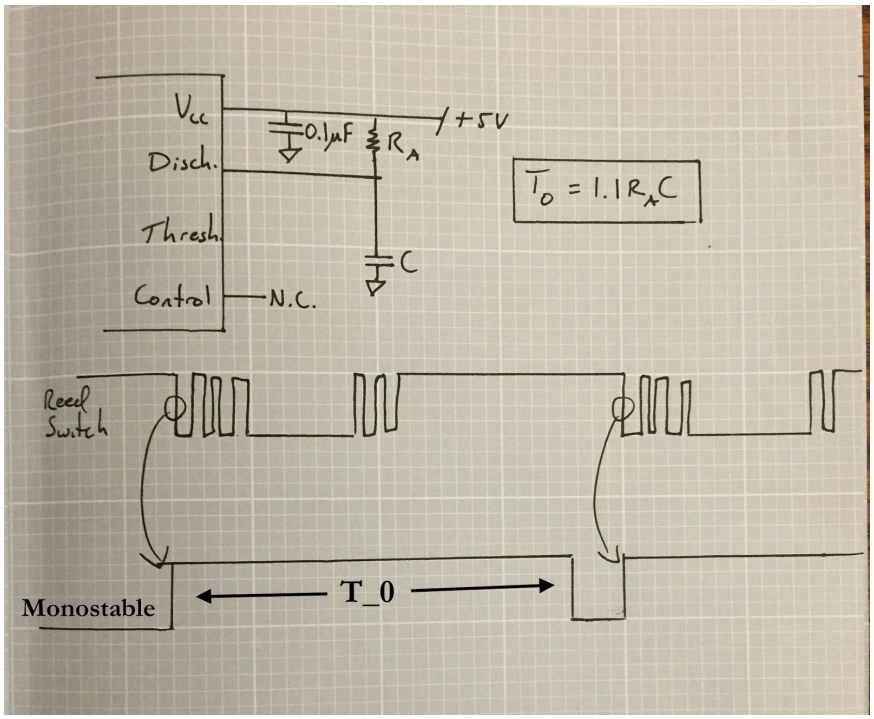
The reed switch system was the only part of our project that physically interacted with our bike. We soldered wires to our switch and positioned it low on the front fork of the bike, being sure to place it on the inside so that it was directly exposed to the wheel. We then secured a magnet to our spokes so that it would pass by our switch as the wheel spun. The wires leading to the switch were held in place with tape, and the magnet held itself to the spokes with its own magnetism.

We decided to place the magnet and reed switch as close to the center of the wheel as possible. Initially, we weren't concerned with placement, but we realized that securing the magnet closer to the center of the wheel would make the magnet move more slowly than if it were toward the edge (physics tells us that Linear Velocity = Angular Velocity * Radius, so by reducing the radius we are reducing the magnet's linear velocity). By making the magnet move more slowly, we were trying to create a cleaner signal by allowing more interaction time between the switch and the magnet. We really only needed a single rising edge, and placing the magnet at any location would likely give us that, but a lower-speed magnet helped ensure our switch would always register the magnet passing.

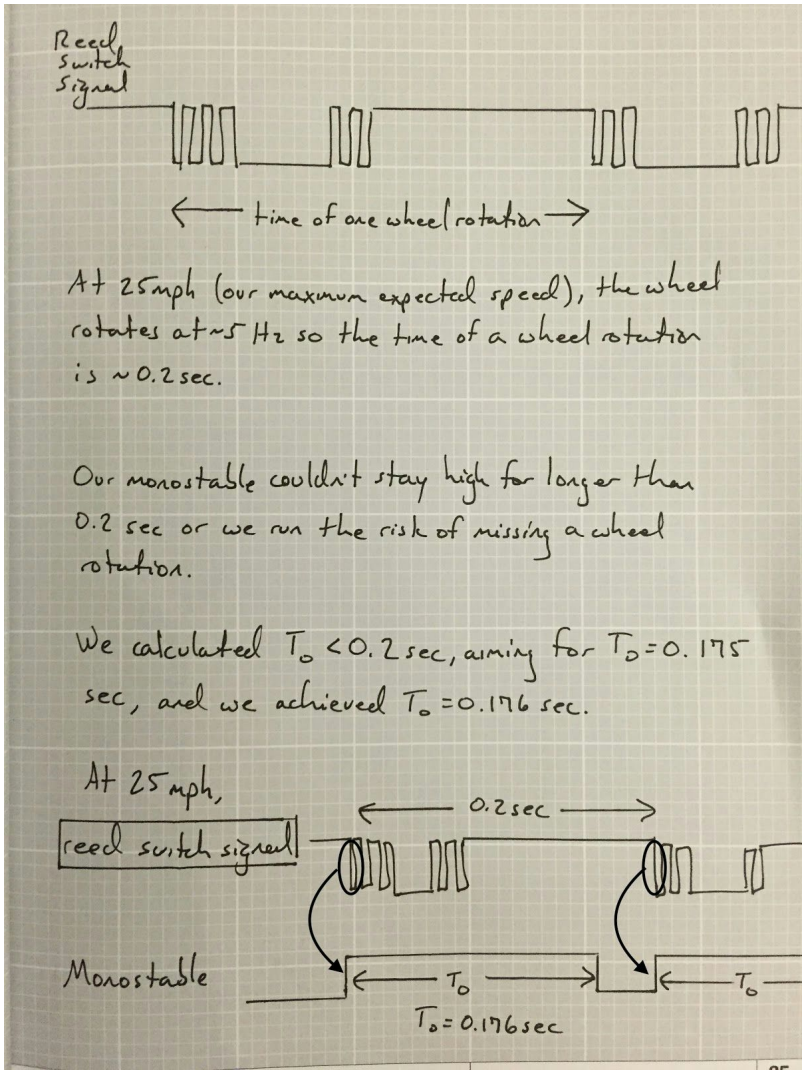
The switch was connected with a pullup resistor to an LMC555 timer on one side and to ground on the other so that each wheel rotation caused our 555 trigger signal to go low. We configured the LMC555 as a monostable so that for each pulse received, its output would go high for a fixed amount of time (this signal will be referred to as the monostable output at various points in the lab). This pulse served as a clock for numerous 74HC161 timers. We determined our monostable pulse length by estimating the minimum length of time between reed switch pulses and setting our monostable output to just under that length of time.



Schematic of our reed switch and magnet signal going to our monostable.



The equation for calculating the length of our monostable output.

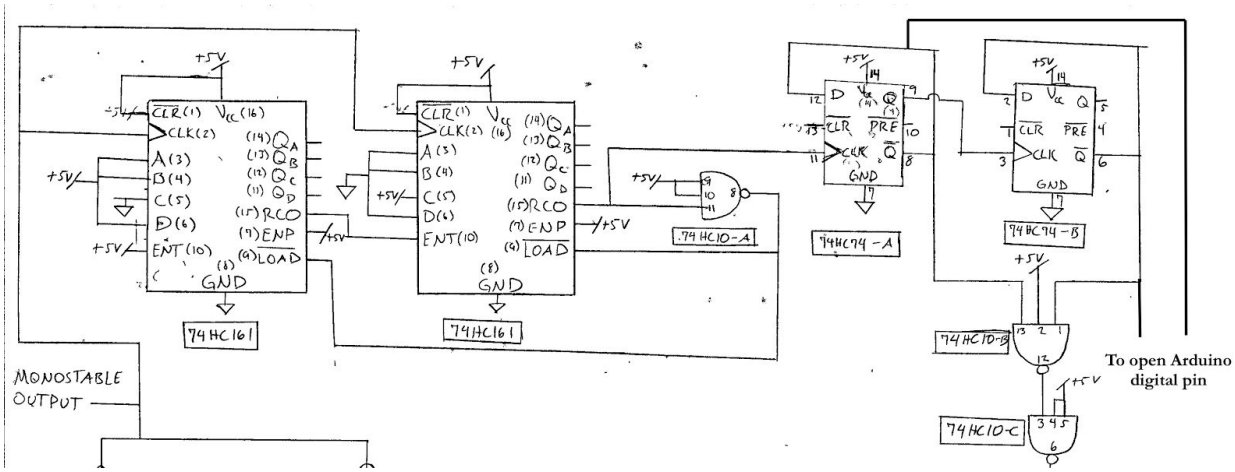


The explanation for determining out monostable output length.

The first two 161s were "¼-mile counters." We calculated that ¼ of a mile would require 180 wheel rotations of our bike, which had 28" diameter wheels, so we loaded these two counters to 255-180 = 75 so that the second counter's Ripple Carry Output would go high each time our wheels rotated 180 times. We then re-loaded the counters with the RCO signal, but we first had to invert it. This required the use of a triple NAND gate (74HC10). We used one of the gates inside, and by tying two of the inputs high and feeding RCO into the third, the output was simply the inverted RCO.

We also fed our RCO into the clock input of flip-flop A of a dual flip-flop chip (74HC74). Q* was connected to D in flip-flop A, and then Q was connected to the clock input of flip-flop B. Again, Q* was connected to D in flip-flop B. Connecting the two flip-flops in this way causes Q of flip-flop B to have a frequency that is one fourth that of the clock of flip-flop A, meaning Q of flip-flop B is sending a pulse for every mile instead of every ¼ mile. However, this configuration also allows us to read the two Q* to D feedback loops as bits that tell us if we are at a ¼, ½, ¾,

or 1 mile (see picture for details). This data gives us ¼ mile resolution that we can concatenate to our mile counter, which is the next part of the odometer.

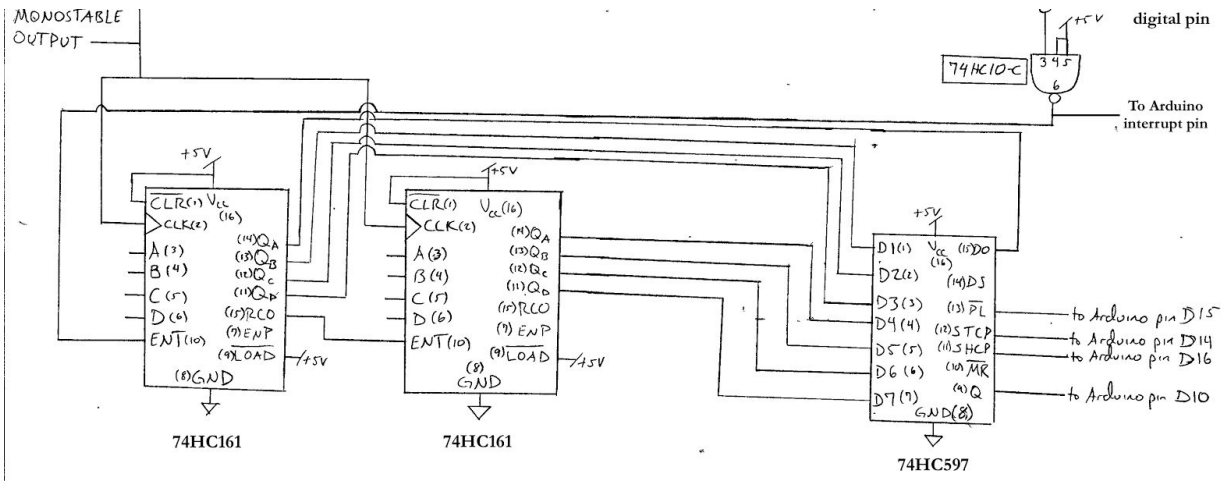


First half of digital odometer. The 161s pulsed every ¼ mile, and the 74HC10-C went high every mile.

The Q output of flip-flop B was actually not used as our mile counter, however. We fed Q* of both flip-flops into the second NAND gate of the triple NAND chip and tied the third input high, so that the NAND would go low when both Q* loops were high (this happens with the same frequency with which Q of flip-flop B goes high so it also happens once per mile). However, we want this signal to be active high so we used the third NAND gate as an inverter, just as we used the first one, and ended up with an active high pulse that happened once per mile.

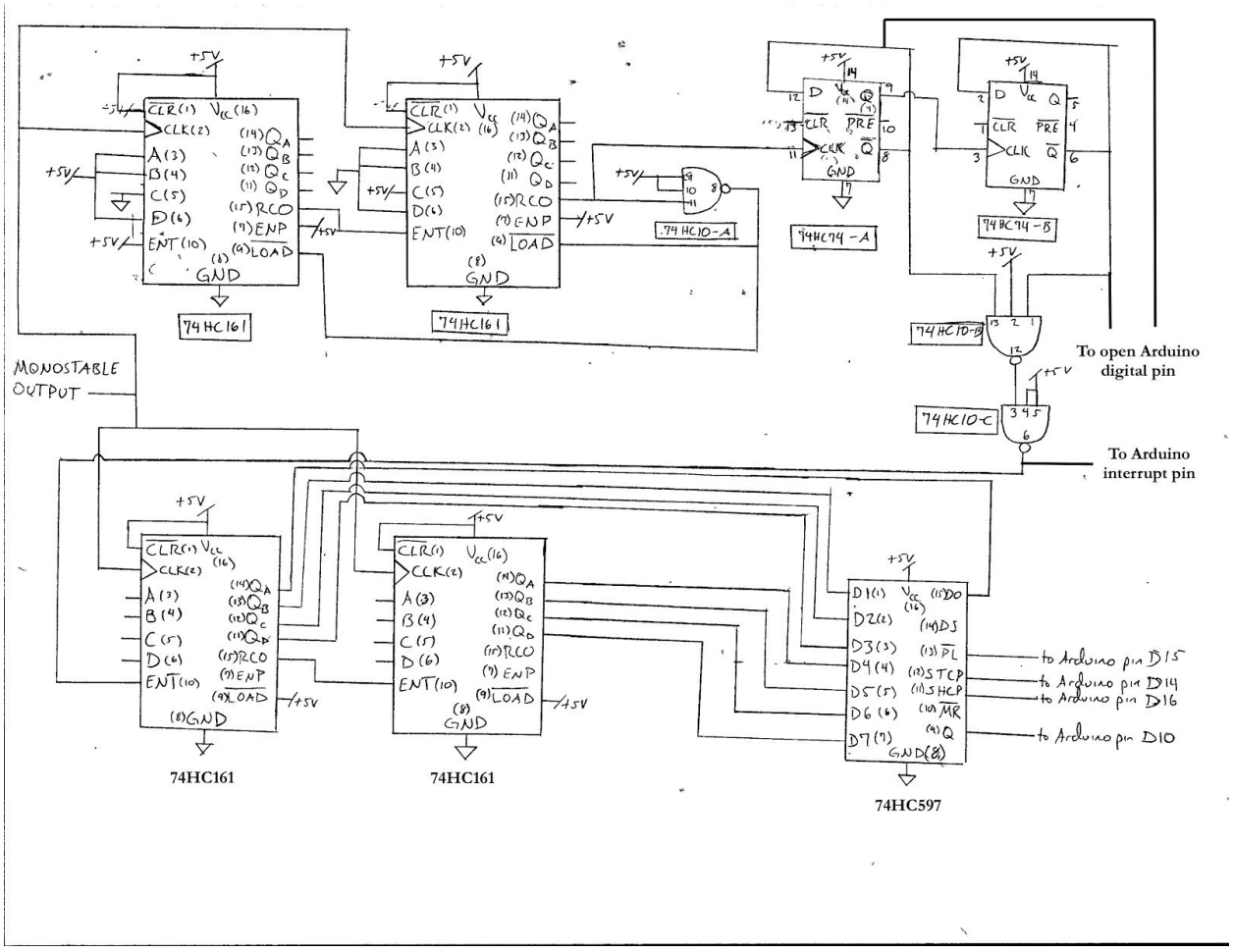
This signal from the flip-flops went to the ENT pin of a third 74HC161. The clock for this counter was the monostable output that served as the clock for the first two counters. The RCO of this third 161 fed into the ENT pin of a fourth 161, whose clock was also the monostable output. These two 161s served as our dedicated mile counters. Their Q outputs were each fed into an 8-bit shift register, a 74HC197. The microcontroller connected to this shift register and was triggered once per mile to use the "shiftIn()" function to read the odometer count. This would load the 8 bits of data from the two counters into the microcontroller. Each Q* feedback loop also fed into a pin of the microcontroller, which would read the bit value of 00, 01, 10, or 11. When displaying the odometer, we would display the 8 bits of data first and then concatenate a value of 0/4 mile, ¼ mile, ½ mile, or ¾ mile that corresponded to the mile information gathered from the Q* pins.[1]

[1] Due to the digital version of the odometer not being fully realized, this Arduino code in the appendix that was used in the fair does not completely reflect the idealized version.



Second half of digital odometer. This kept track of miles and fed into a digital shift register.

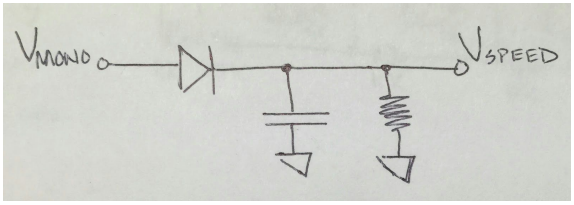
Here is the complete odometer schematic the way we wanted to implement it.



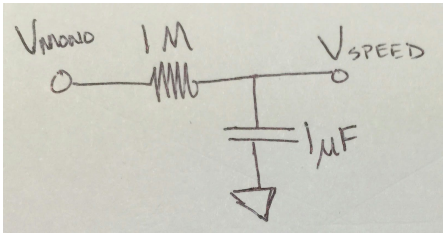
Complete schematic for digital odometer

C. Speedometer

Knowing that we already had more digital than analog components in our circuit, we were determined to make out speedometer out of analog components. To do this, we knew we needed to manipulate the signal from the monostable that pulsed once every time the front wheel made one rotation. As the speed of the bike increases, the frequency of the pulses increase as well, for the speed of the bike is directly proportional to the frequency of the monostable output. Therefore, we decided to put a low-pass filter after the monostable output to give us an output voltage proportional to the frequency of the monostable. The first configuration we tried is below.

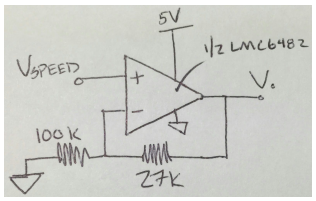


The idea behind this design is that the diode would force the capacitor to discharge through the resistor. We tried this early on with multiple resistor and capacitor values (not shown), and none of the outputs we measured were very accurate, consistent, or quick to adapt to a changing frequency from the monostable. Because we needed an output voltage that changed fairly quickly, we decided to try a more basic low-pass filter:

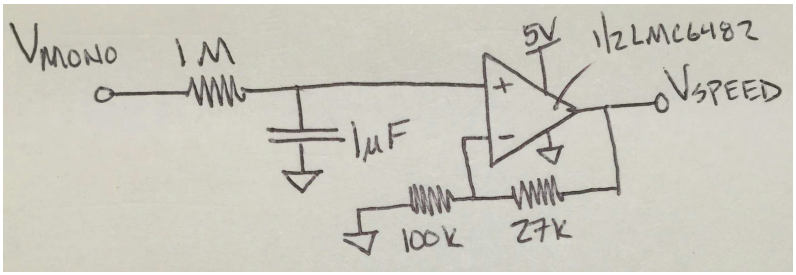


The output of this filter gave us a voltage that was very consistent with the frequency of the monostable (see the *Results* section for oscilloscope readings). The time constant of this circuit is (1 MΩ) * (1 µF), or 1 second. We chose this time constant because this is equal to the maximum period of the signal from the monostable. After trying it out with the scope, we knew this was a signal we could work with.

However, the minimum and maximum values outputted from this low pass filter were ~0 V and 3.92 V respectively, and each represented a speed of 0 mph and 25 mph respectively. In order to increase our resolution of speed, we decided to amplify this signal to make better use of our circuit's 0 - 5 V operating range. Thus, we implemented the circuit below amplify our speed value to range from 0 - 5 V.



The gain of this circuit is 1 + (27 kΩ) / (100 kΩ), or 1.27, and (3.92 V) * 1.27 = 4.97 V. This circuit now gives us a speed voltage range of 0V - 5V. The complete schematic is below.



The output, V_{SPEED}, was sent to a microcontroller to be "analogRead" and translated into a number for the LCD display to use. Details of this process can be found under sub-section E. of this section.

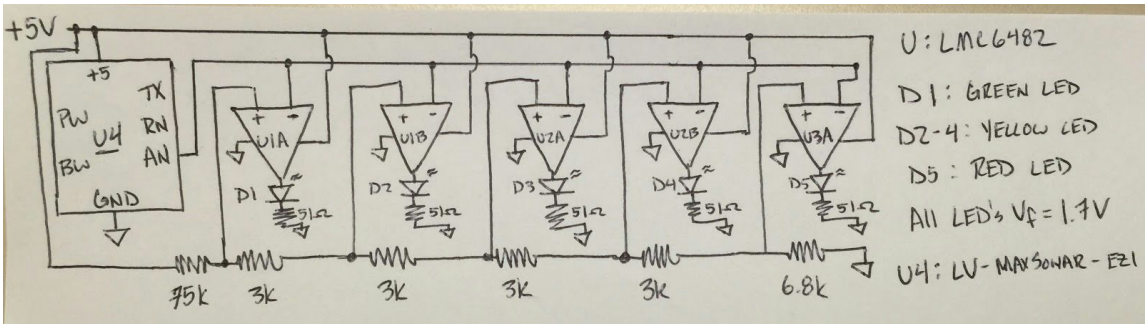
D. Proximity Sensor

The proximity sensor was one of the main modules and goals of the project. We wanted to give the biker a way to know how close an object was to the bike without having to turn their head and create a blind spot in front of the bike. The first thing we did was research different proximity sensor online from Sparkfun and Adafruit. Both offered basically the same thing, and we chose the Sparkfun "Ultrasonic Range Finder - LV - MaxSonar - EZ1" because of its ability to detect object up to 6 meter away, which we thought was crucial to our project and to a potential biker.

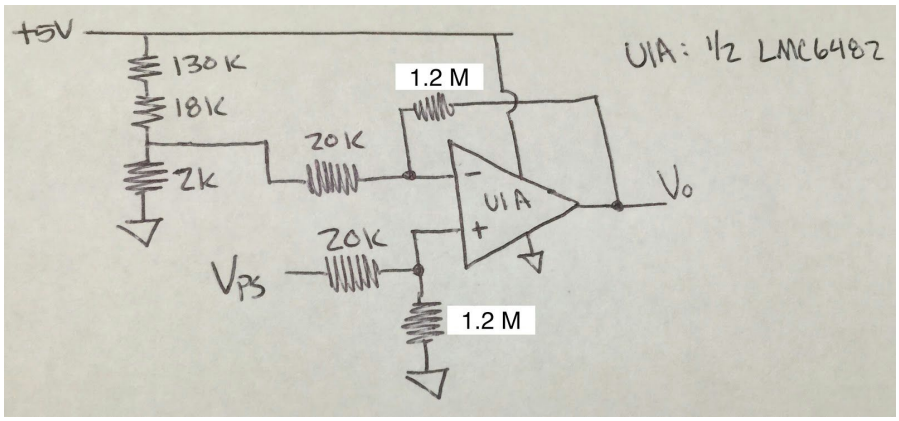
The design of the proximity sensing system was purely analog. The sensor itself has a very useful "AN" pin that outputs a voltage proportional[2] to the distance of the closest object it detects: the further the object, the higher the voltage output. We designed the system to use this output and five LMC6482 op-amps[3] to successively turn on LEDs of varying color as the voltage readout decreased. Initially, each of the five op-amps' inverting input terminal (V_-) was connected to the output of the proximity sensor, and the non-inverting input terminals (V_+) were each connected to a different reference voltage created via a voltage ladder[4], V_{CC} and V_{EE} were connected to 5V and ground respectively, and the output was connected to an LED and resistor in series that was grounded on the other side. To determine the resistor necessary to allow maximum brightness through the LED, we determined the forward voltage of the LED, subtracted this value from the 5V rail value, and used this voltage drop across the resistor to figure out what resistance value would yield the highest current without exceeding the 0.25 W rating of the resistor. The voltage ladder we designed using six resistors to give us five successive reference voltages. The proximity sensor is specified to output a signal of 2.5 - 0 V based on

[2] At our 5V supply, the proximity sensor AN pin outputted 9.8 mV/inch.
[3] We decided to use op-amps instead of a native comparator due the fact that the LMC6482 gave us two op-amps per IC; whereas, if we used the LM311 comparator, we would have had to either use much more board space or cut down on the number of lights we wanted to use to alert the biker of impending doom. We chose to use more lights and go with the LMC6482.
[4] http://en.wikipedia.org/wiki/Voltage_ladder

distance, so we made the voltage references based off this idea to corresponded with distances of object we wanted to alert the biker of. Ideally, the op amps would have compared the AN output of the proximity sensor with the voltage ladder reference and either turned the LED on or off based on the distance of an object to the sensor. However, upon testing, this did not work as intended.

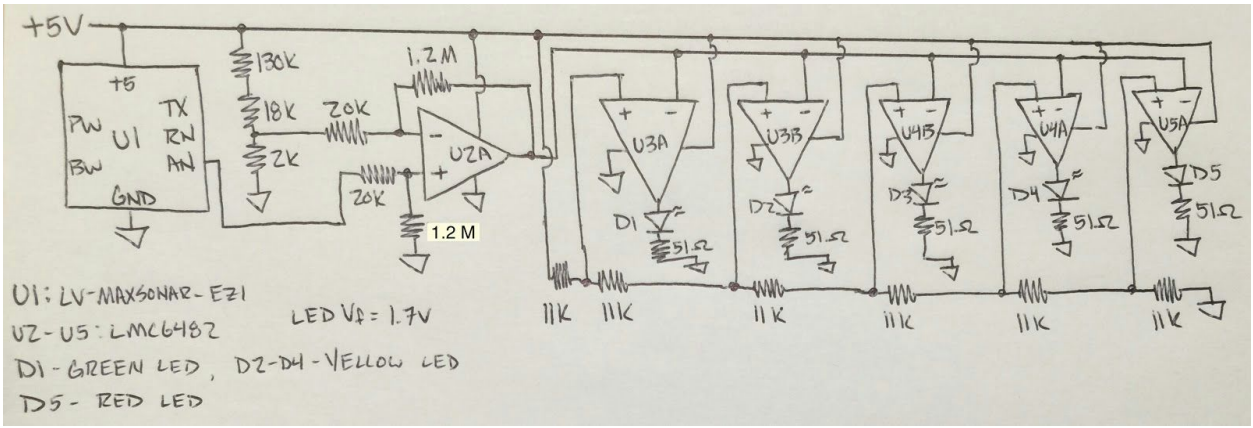


One crucial step that we forgot to take was characterizing our proximity sensor as we had done for so many other sensor in class. Upon examination, it turned out that our sensor did not work for up to 6 meters, but rather it reached its maximum voltage AN output, 150 mV, close to 18 inches (0.45 meters), and minimum voltage output, 66.7 mV, at 6 or fewer inches. Thus, it was time to rethink our design. Since the maximum distance of the sensor was so small, we wanted to use every bit of the 18 inches with maximum resolution. So, we implemented a biasing and amplifying circuit to enhance the signal from the sensor using an LMC6482 op amp.



This configuration yields $V_O = (V_B - V_A)*(R_2 / R_1)$, where V_B is the input from the proximity sensor and V_A is a 66.7 mV reference voltage. The gain is 60, which makes the maximum and minimum outputs 5 V and 0 V respectively. Now that our comparing op-amps are reading a voltage from 0 V - 5 V, we needed to change our voltage ladder. This time, creating the references were easy; indeed, all we needed were six equal resistors connected in series from 5V to ground to give us five equal increments of voltages from 5 - 0 V. Of course, we chose resistors that would limit our quiescent power as much as possible. The final designed consisted of the proximity sensor's AN output being biased and amplified to register values between 0 V

and 5 V, and these values were compared with successive voltages references via op-amp to turn on and off LEDs as the output from the AN pin varied. The closer the object, the lower the AN voltage, and the more LEDs were on. The colors of the LED were green - yellow - yellow - yellow - red, indicating the level of danger rising or falling. This was easy to test, as all we had to do was move closer and further away from the sensor and see if the lights reacted. The final schematic is below.



One aspect of the proximity sensor we would have liked to implement was an audible feedback system. We experimented with it quite a bit but were unable to get it working properly before the project was over. Essentially, we wanted to have an AND gate that took in two inputs: 1) the signal from the last comparator from the above circuit that went high when an object was 6” or closer to the bike, and 2) a signal from a comparator that went high every time the speed was above some certain threshold. We wanted to take the output of this AND gate and turn on a speaker with some sound anytime the output was high. This would mean the biker would get some audible feedback anytime he/she was above some certain speed and a foreign object was close to the bike. We almost had it working before the fair, but were unable to get the speaker to work properly. We tried using a class B amplifier, but, unfortunately, we just ran out of time. Given one more lab session, we feel as if we could have easily implemented this.

E. Microcontroller and LCD Display

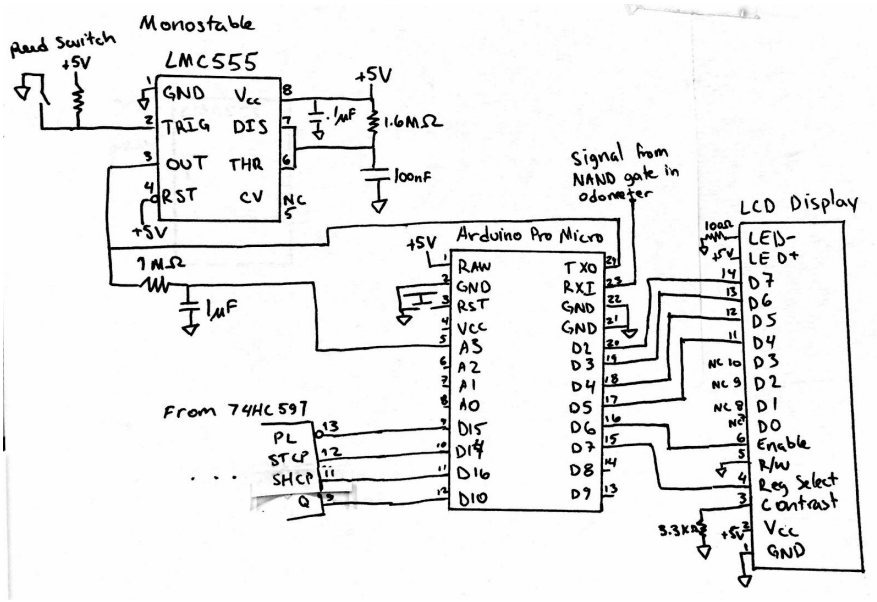
To show the speed and distance to the user effectively, we used a 16x2 Character LCD Display, with the input regulated by a the Arduino Pro Micro Microcontroller. To figure out the speed, the microcontroller took in the voltage reading from the op-amp-low-pass-filter combination that came after the monostable, as the voltage at this point was proportional to ⅛ of the speed in miles per hour due to the gain of the amplifer. Therefore, we could get the speed by reading the voltage then multiplying by 5 and dividing by 1024, which is the maximum of what Arduino converts the voltage into. This value is the speed in mph that is saved to display to the user. The speed is read at the same time with every revolution of the wheel so that it maintains a constant reading if the biker is going a constant speed. This is accomplished by analog-reading the voltage only at the rising edge of the monostable interrupt, which is precisely the voltage we used to figure out the speed-mph relation. The distance was originally calculated digitally and

was to be read serially in, but for the fair, we were unable to complete it. For the fair, we calculated the distance by counting the amount of interrupts, which is sent each time the wheel spins. Then, by dividing the amount of spins by 720 for this particular wheel, we can get the distance in 1/10 miles.

The display is completely connected to the Arduino, power, and ground. After connecting the display's Reg Select, Enable, and 4 data pins to the Arduino, and the appropriate pins to power and ground, we used the the LiquidCrystal Arduino library as follows:

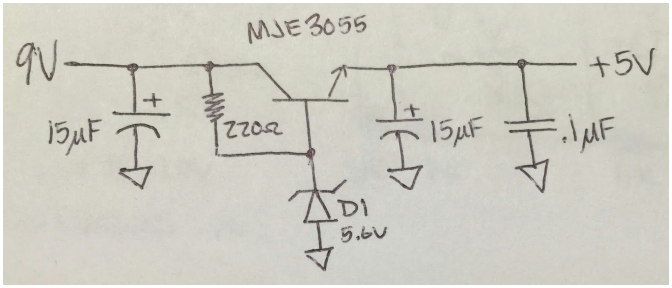
- To initialize, adding the library, selecting pins, and designating proportions
  #include <LiquidCrystal.h>
  LiquidCrystal lcd(7,6, 5, 4, 3, 2);
  lcd.begin(16, 2);
- To place the cursor at a position, and print a string or number:
  lcd.setCursor(0, 1);
  lcd.print("Your Speed:");
- The entire code is located in the appendix.

The schematic and connections are shown below:



F. Voltage Regulator

The voltage regulator system came out of the necessity to be able to run our project off of a 9 V battery[5]. We used the idea from Lab 6 where we designed a regulator to continuously output a voltage of 5 V regardless of the voltage level of the 9 V battery. Essentially, the zener diode holds the voltage at the base of the MJE3055 to 5.6 V, and because the MOSFET is not saturated, there will be a 0.6 V drop from the base to the emitter. This ensures that the emitter voltage is a constant 5 V.



D1: 1N4734 Zener Diode

Results

Turn Signals

The turn signals worked really well functionally. Two of three LED's served as running lights that are on when not turning, and there is a clear cascading pattern after switching on either the left or right switch, or both. The only improvements could have been moving the LED's to a more visible location, which would require a more wire for each LED, brighter LED's to be more effective at warning cars, and switches that were easier to turn on while riding a bike. Otherwise, we were very happy with our turn signals.

Odometer

Our odometer gave us trouble, and it initially seemed as though we may have had faulty parts. We followed our signal through the circuit to see where it began to become compromised and found that initially it worked as expected. Our ¼ mile counter worked fine, counting from 75 to 255 and then resetting to 75 again, and the NAND gates successfully inverted the RCO signal for our load inputs. We found that once our signal reached the flip-flops, our circuit did not behave as expected. The clock input to the flip-flop A (which came from the RCO pin of the second 161) was clean, but the Q* and Q signals were almost entirely noise and didn't even have a distinctive high or low value. Because of this, we were unable to follow the signal further and see if the next two 161s or the shift register were working properly. We unfortunately did not have time to fully debug this circuit or try out a different flip-flop IC, but when we connected the clock of flip-flop A to the waveform generator instead of our second 161s RCO, our flip-flop circuitry functioned as desired. Because the only thing that changed between a non-functional flip-flop circuit and a functional flip-flop circuit was the clock input, it seems like there may have been some physical feature of the RCO signal that made it an unsuitable clock input. However,

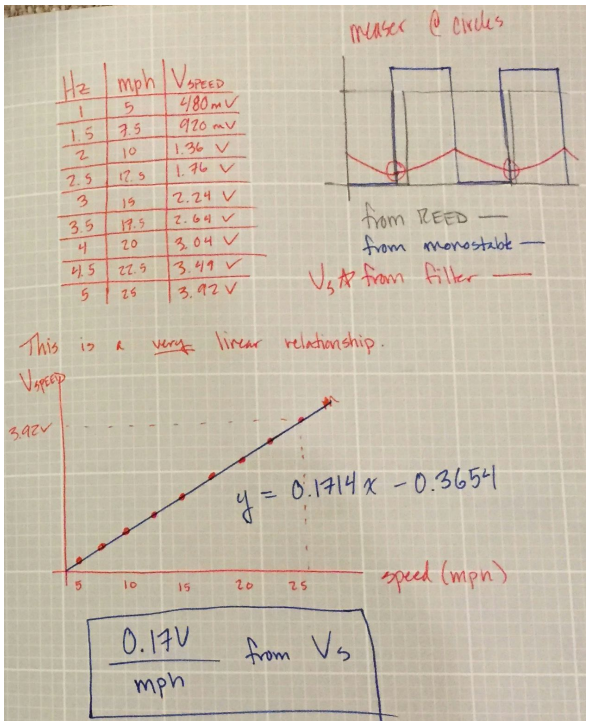
[5] Incidentally, we discovered our circuit drew enough current that we had to use multiple 9V batteries in parallel to power our project independently of a power supply.

we also neglected to tie the set and reset pins of the flipflop high, so that could have been a factor in our problem.

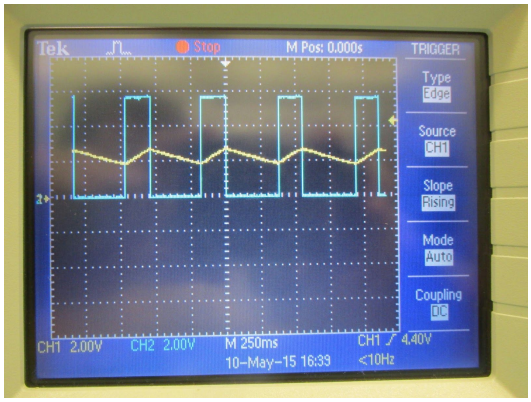
In order to have a functional odometer for the fair on Wednesday we used the monostable output connected to the interrupt to count amount of rotations as mentioned above under the Microcontroller section.

Speedometer

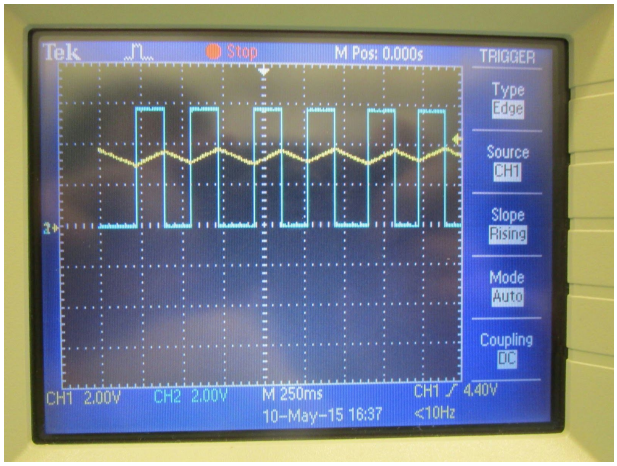
Our speedometer worked quite well at speeds over about 9 mph. We were able to achieve and identify a linear relationship between the out low-pass filter output and the actual speed of the bike. The below chart and graph represents the actual speed of the bike in mph and the V_{SPEED} that was outputted from the low-pass filter (before adding in our amplifying circuit).



As can be seen, there is a linear relationship between the output of our low-pass filter and the speed of the bike. We took all of our measurements on the rising edge of the monostable signal. From this point, we decided the bias was small enough to be ignored and amplified the signal to range from 0 - 5 V. This 0 - 5 V signal was read into the microcontroller on each rising monostable signal, multiplied by 5, and sent to the LCD display. Below are two snapshots of from the oscilloscope. The blue line represents the one-shot signal from the monostable, and the yellow signal is the output from our low-pass filter.



A speed of 10 mph, as indicated by the 2 V signal at the rising edge of the monostable.

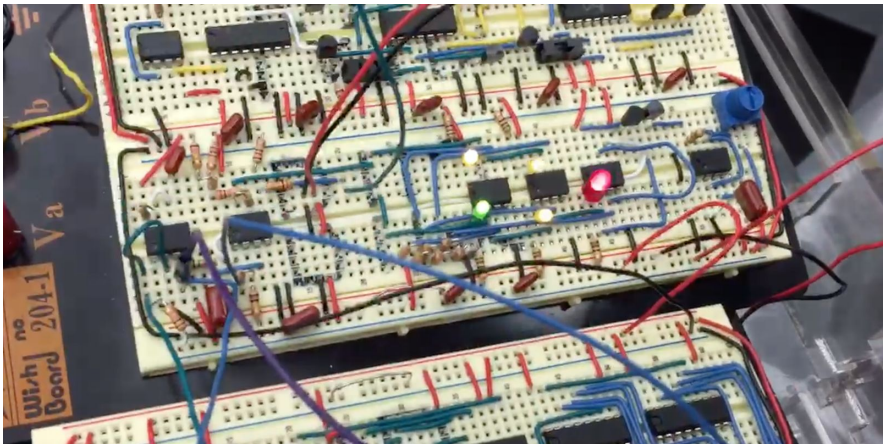


A speed of 15 mph.

We were able to successfully read this voltage from the microcontroller and display it to the biker. Speeds from 9 - 25 mph were successfully calculated.

Proximity Sensor

The proximity sensor worked very well. Even though we were not able to detect objects at ranges greater than 18 inches due to our sensor, the logic and circuitry behind what we did with the sensor worked as we had hoped. Whenever an object got within 18 inches of the sensor, the green light came on, and as the object got successively closer, each yellow light and finally the read came on after that. After the object was within 6 inches of the bike, all of the lights would stay on. Below is a picture of an object within 6 inches of the bike.



Looking back on our design, we decided to use op-amps as comparators strictly due to the fact that each LMC6482 IC gave us two op-amps. Whereas, if we would have used an LM311 for every comparator, we would have either had to use lots of board space or fewer lights. Thus, we decided to use op-amps as comparators even though the comparisons were not as fast and we could not add bias. The lights flickered more than we would have liked, but we felt that it was worth it to put more lights on the board.

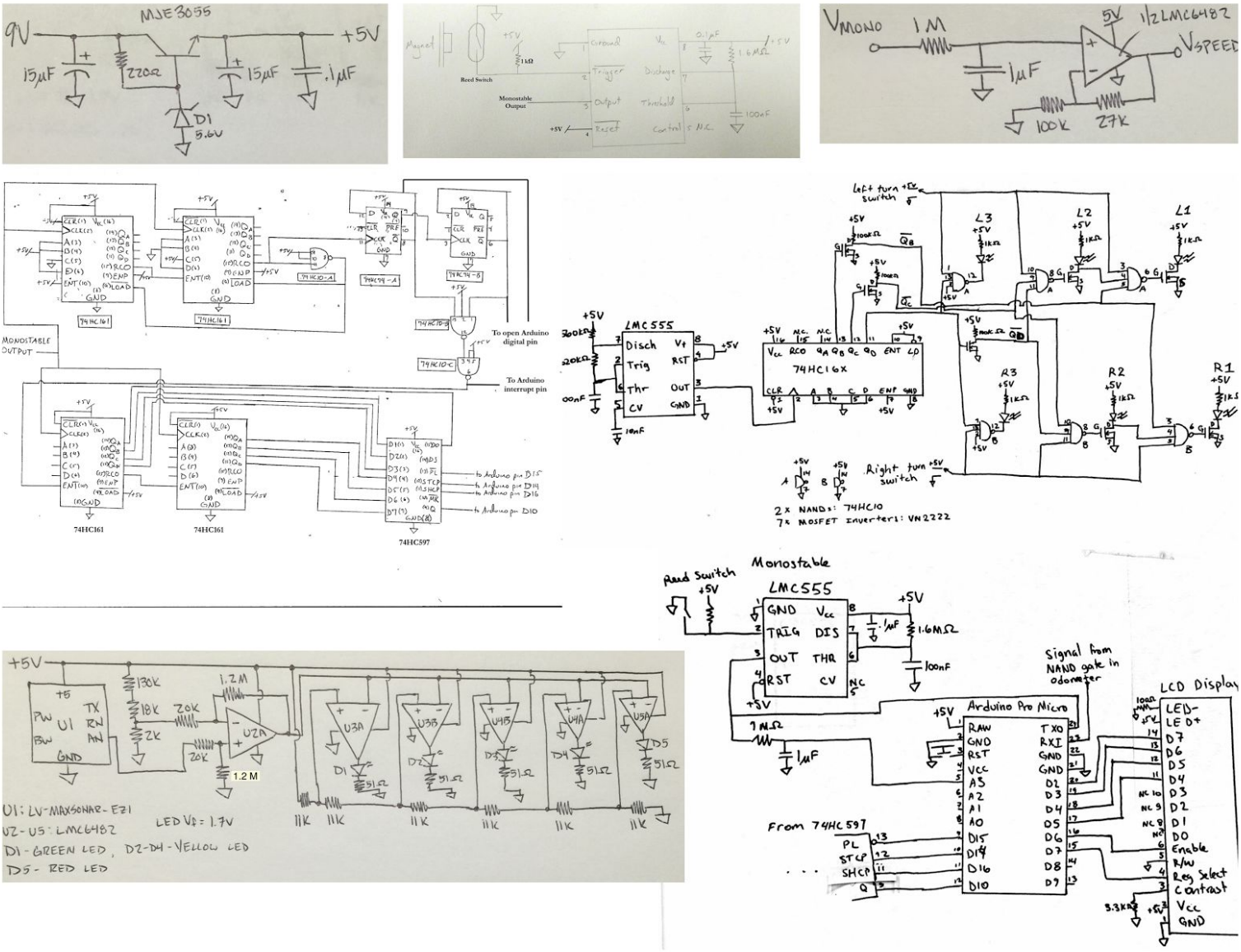
Design Narrative

We feel that we handled our design process well. We began thinking about the project early, meeting over lunch multiple times before our proposals were due to discuss different possibilities. Our general plan was to design each element of our project in its entirety before beginning to build, being sure to seek out what we thought were the best solutions and often double-checking things with TFs or instructors before continuing on. Since our project was a collection of different additions to a bicycle, it was rather easy to split it into parts and design one part at a time. While this approach helped to make sure we gave ample thought to our solutions, it also meant that we had less time to debug if the circuit that we devised did not work since we ended up designing almost the entire circuit before building anything. Aside from the odometer, we did not run into any problems we couldn't debug when actually building the circuit, but we were left with too little time to fix our odometer. If we were to go back to the beginning of the design process, we would begin to build the different parts of our project as soon as we finished the schematic for that element unless component placement depended on other parts of our circuit.

Early in our design stages, we realized the magnitude of the difference between casually writing down what we want and actually creating a schematic for a circuit that does what we want. It was relatively easy to design a project to the level of detail that says "send this signal here, and when this happens then do this," but much more difficult to figure out the details of how to make that happen. For instance, making sure that a signal is active low and not active high, or that an input responds to a rising rather than a falling edge, or that lights blink on the same clock, is much more difficult once it actually has to be implemented. We fairly quickly

arrived at potential solutions for the different goals of ours, but the bulk of our work was definitely in figuring out the details of how we would make our theoretical solutions into schematics.

Schematic



Clockwise, from top left: 5V Voltage Regulator; Reed Switch and Monostable; Speedometer; Turn Signals; Microcontroller; Proximity Sensor; Odometer.

Conclusion

In the end, we believe we created a great project. We made an odometer, a speedometer, turning signals, and a proximity sensor that could be used on any bike. Even though a few of our components did not function as we had originally intended, we implemented everything that we set out to.

If we were to keep improving our project, we would finish implementing a few features we were unable to finish before the project was over, including fixing out odometer mechanism to output miles from a parallel-to-serial converter to the microcontroller, as well as implementing an audio feedback system to audibly alert the biker that a foreign object is close to the bike.

Overall, we are very pleased with how the project went. We felt very prepared to work on each component and felt confident in our planning and are proud with what we created.

Bibliography

Proximity Sensor (datasheet) - https://www.sparkfun.com/products/639
Reed Switch (datasheet) - https://www.sparkfun.com/products/10601
LCD Display (datasheet) - https://www.sparkfun.com/products/709
Lab 6 - voltage regulator
Datasheets for the following:

- **Data Sheet - LMC555 Timer**
- **Data Sheet - 74HC161/163 4-Bit Synchronous Counters**
- **Data Sheet - 74HC10 Triple 3-Input NAND Gate**
- **Data Sheet - 74HC175 Quad D-Type Flip-Flop**
- **Data Sheet - 1N4148 Diode**
- **Data Sheet - N-Channel VN2222 Small Signal MOSFET.pdf**
- **Data Sheet - MJE3055 NPN Power Transistor**
- **Data Sheet - MJE2955 PNP Power Transistor**
- **Data Sheet - LMC6482 Op Amp.pdf**

Course materials from ES52
Help from the professor, preceptor, and TFs

Appendix

Arduino code used for the design fair:

```
/*
  ES52 - Final Project
  Speedometer and Odometer to display interlay
  5/5/2015

  Greg Hewett
  Frank Dubose
  Antuan Tran

  Input:
  For speedometer: read in a voltage and converts to speed via predetermined
coefficient (linear relation), only when interrupt occurs for consistency

  For odometer: Read in a serial binary value when given interrupt (every mile)
```

```
  Output:
To display, as per Liquid Crystal protocol

*/

#include <LiquidCrystal.h>
LiquidCrystal lcd(7,6, 5, 4, 3, 2);

int speedPin = A3;
int odomPin = 10;
int loadCountPin = 15;
int clockPin = 16;
int STCPPin = 14;
int cycTime = 500;
int signalInterrupt = 0;
int speedInterrupt = 1;
int spins = 0;

int dist = 0;
int begTime;
volatile float speedmph = 0;
volatile boolean odomFlag = true; // start by getting info

void getSpeed() {
 speedmph = analogRead(speedPin);
 speedmph = (speedmph*25)/1024;
 spins++;

}

void getOdom() {
 odomFlag = true;
}

void setup() {
  // initialize digital pins as outputs
  pinMode(speedPin, INPUT);
  pinMode(odomPin, INPUT);
  pinMode(signalInterrupt, INPUT);
  pinMode(speedInterrupt, INPUT);
  pinMode(loadCountPin, OUTPUT);
  pinMode(STCPPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  digitalWrite(loadCountPin, HIGH);
  digitalWrite(STCPPin, LOW);
  digitalWrite(clockPin, LOW);
```

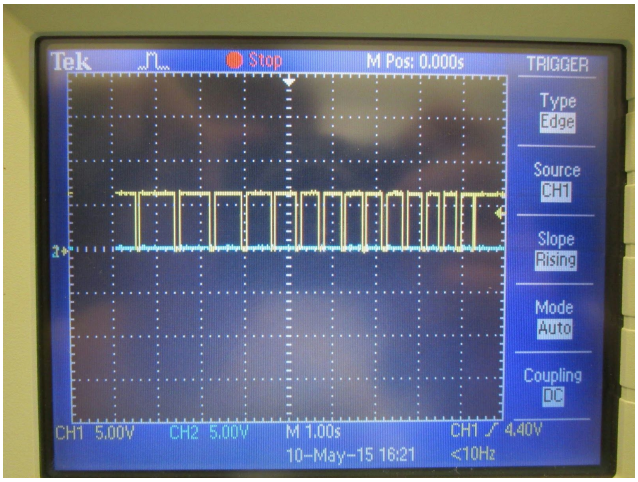
```
  attachInterrupt(3,getSpeed,RISING); // attach interrupt to read the speed at
a certain point every time
  attachInterrupt(2,getOdom,RISING); // attach interrupt to figure out when to
deal with odometer

  lcd.begin(16, 2);
  lcd.print("Your Speed:");
  lcd.setCursor(0, 1);
  lcd.print("Distance:");
}
void loop() {
// following commented out code is for reading distance from digital odometer
 /* if (odomFlag) {
    // initalize clock
    digitalWrite(clockPin, LOW);
    // wait  half a second for STCP to do work
    begTime = millis();// get starting time
    while ((millis() >=begTime)&&((millis() - begTime)< 500));
    digitalWrite(STCPPin, HIGH);
    // wait half a second for the odometer count to update on counters
    begTime = millis();// get starting time
    while ((millis() >=begTime)&&((millis() - begTime)< 500));
    digitalWrite(loadCountPin, LOW);
    // wait half a second for the paralled to serial converter to be ready
    begTime = millis();// get starting time
    while ((millis() >=begTime)&&((millis() - begTime)< 500));
    digitalWrite(loadCountPin, HIGH);
    dist = shiftIn(odomPin, clockPin, MSBFIRST);
    // reset load pin
    digitalWrite(STCPPin, LOW);
    odomFlag = false;
    Serial.print(dist);Serial.print("\n");
  }*/

  dist = spins; // can change the ratio of spins to dist to display in miles,
km, etc
  lcd.setCursor(12, 0);
  lcd.print(speedmph);
  lcd.setCursor(12, 1);
  lcd.print(dist);
  begTime = millis();// get starting time
  while ((millis() >=begTime)&&((millis() - begTime)<cycTime));

}
```

Other oscilloscope readouts:
Reed Switch Signal (yellow)



Monostable (blue) with **Reed Switch Signal** (yellow)