Resistor Sorter

David Gordon Gentry and Charles Timblin Sussman

E90 Final Report

5/5/05

Introduction

The idea for our senior design project stemmed from a list of potential design projects presented at the first E90 meeting. Since we did not have a particular project in mind at the time we chose the resistor sorter. It appealed to us because we felt it would give us a chance to incorporate several different fields of engineering: mechanical, electrical and computer. The mechanical aspect was designing the system and building it in the shop; the electrical component consisted of designing a circuit to determine resistance which we then needed to program with specific outputs for each resistance.

When we began designing the resistor sorter, we hoped its main purpose would be keeping the electronics laboratory cleaner and providing an easy and efficient method of sorting resistors. Looking at the electronics lab, there are a plethora of resistors scattered over the benches, and the only way to determine their values is by their unique color patterns. Our project aims to eliminate this tedious method. As we progressed we realized that our project could be used as a way of targeting younger students and getting them interested in engineering. Consequently, one of the new goals of our project became making it aesthetically pleasing.

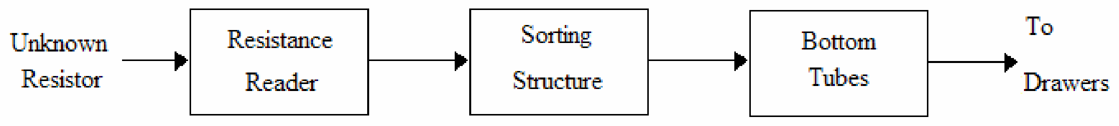


Figure 1 - This figure shows a block diagram of our system.

The general design for our system is shown in Figure 1. We start with an unknown resistor which is then fed into the resistance reader. After the resistance is determined, the resistor is fed into the sorting structure where it falls down specific tubes until it ends up in the bottom level of tubes which each hold a different resistor type. After all the resistors are sorted, they are taken from the tubes and placed into the set of drawers in the electronics lab.

Theory/System Design

Resistance Reader

Before the resistor enters the tubes, its resistance is determined using a voltage divider. As shown in Figure 2, a voltage divider is composed of a known resistor and the unknown resistor linked together in series. The top of the known resistor is connected to 5V, while the other resistor is connected to ground. The voltage between the resistors is

$$V_{out} = 5\frac{R_{unknown}}{R_{upper} + R_{unknown}}.$$

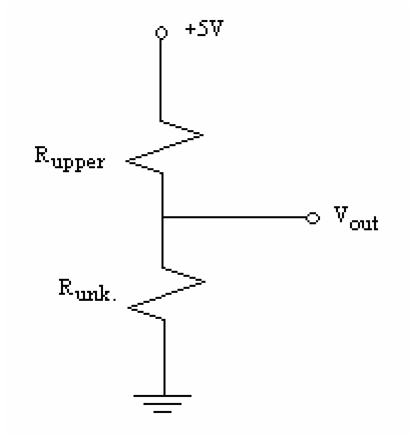


Figure 2 - This shows a standard voltage divider circuit

Depending on how large the unknown resistor is compared to the known resistor, the voltage between the resistors will change. For example, if both resistors are the same, the voltage between them will be 2.5V. V_{out} can be compared to 2.5V as the known resistance varies from 1Ω to 1M Ω. If the unknown resistors are always compared to the same known resistance, the voltage input to the Printed Circuit Board (PCB) will also vary from 0 to 5V, but it will do so in small enough increments that the resistance reader will not be able to detect the variations. Therefore, we developed a system where the resistor to be compared to the unknown resistor would change depending on the unknown resistance.

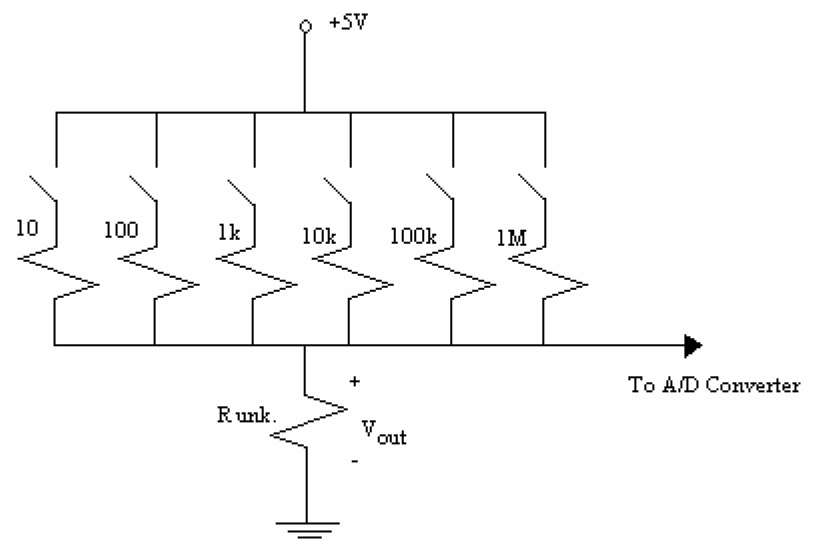


Figure 3 - This shows the schematic for our resistance reader circuit.

As can be seen in Figure 3, the unknown resistor is connected to ground and then connected to six different resistors of values 10, 100, 1,000, 10,000, 100,000, or 1,000,000Ω in parallel. The other side of these resistors is connected to 5V. Only one of the switches is in the closed position, while the other five are all open, acting like open circuits. Whichever resistor is connected to the unknown resistance acts like R_{upper} in the voltage divider. The point in between where the unknown resistor is connected to the known resistors is where the output voltage is taken.

A program loaded onto the PCB decides which one of the switches is closed and which ones will remain open. Initially, the program connects the largest resistor, the 1MΩ resistor to 5V. If the voltage divider reads something below 2.5V (which it will do unless the resistor is equivalent to 1MΩ or greater), then the 1MΩ resistor is disconnected, and the 100kΩ resistor is connected. If the voltage is greater than 2.5V, the compared resistor will remain 1MΩ. If the compared resistor is changed to 100kΩ, the voltage input is once again read. If the output voltage is less than 2.5V, the 100kΩ is disconnected and the 10kΩ resistor is connected. The same process continues, and the compared resistor continues to decrease by a magnitude of 10 each time until the output voltage is greater than 2.5V. For resistors less than 10Ω, a special case had to be made within the program as the output voltage will never fall below 2.5V since the smallest compared resistor is 10Ω. Basically, once the program reaches the 10Ω it stops changing the compared resistor, and the unknown resistance is determined based on the output voltage from the voltage divider even if it is below 2.5V. Overall, this process only takes a few seconds, and it also does well in separating the resistance values into six groups, which helped to debug the resistance reader in the earlier stages.

In order to properly separate the resistors based on the input voltage, the voltage is converted from an analog value of 0 to 5V to a digital value of 0 to 1023, with 512 being the equivalent to 2.5V. This is an automatic function of the PCB, and it did not require any extra programming beyond determining which pin we are using to input the voltage from the voltage divider and entering that pin into the program. Once the correct compared resistor is found, the program itself holds the digital voltage readings that are used to determine what the resistance is.

These voltage readings are based on a range of digital values that were calculated using excel. Basically, we entered in the value of all 96 resistors in an excel spreadsheet. We then determined what voltage would be output from the voltage divider if the voltage is greater than 2.5V. This voltage was then multiplied by 1023 to find its digital value. A range of digital values for each of the resistors was made by averaging the output voltage of a sample resistor with the output voltage of the resistor of the next higher resistance to find the upper value, and averaging the voltages from the sample resistor and the next lower resistance to find the lower value. For the resistors whose values are equivalent to the compared resistors, there are two different resistors that they could be compared to and still fall within the allowable range. The digital ranges for each resistor are represented in the program as a sequence of if-then statements, saying that if the voltage is less than its upper digital value and lower than its lower digital value when being compared to a certain resistor, then it must be that resistor. The range of digital values for each of the resistors in the laboratory is shown in Table 5. One example of how the table works is as follows: if the voltage output by the voltage divider is 3V, and the known resistor that is connected to 5V is the 1kΩ resistor, the program will acknowledge the resistor to be 1500Ω.

Once the resistance is determined, the PCB outputs voltages to the solenoids, forcing the levels to move in the correct directions at the correct times. The tubes at the bottom of the pyramid structure are placed in order where the resistors for the first half (smaller) resistors are in the front and the second half is in the back. Therefore, if the resistor is in the smaller half of resistance values, the solenoid on the side closest to the observer will be powered, and the first level will be pulled inward. The values then go from left to right, where the smaller resistances are on the left. The program determines which resistor to output voltage using the following algorithm:

1. The resistors are numbered from 1 to 96 based on their value, smallest to highest.
2. The resistor number is divided by 48 where any remainder is taken out.
3. If the answer is one, output high to the back solenoid of the eighth level and subtract 48 from the resistor number, if it is zero, output high to the front solenoid.
4. Divide the resistor number by 24, if the answer is one output high to the right solenoid of the seventh level, set k=16 and subtract 24 from resistance number, if zero output high to the left solenoid and set k=8.
5. Divide the resistor number by k (since two-thirds will go one way), if the answer is equal or greater than one output high to the right solenoid, otherwise output high to the left solenoid.

Sorting Structure

The initial design we came up with for the sorting structure of our system was to build a pyramid structure to implement our binary sorting technique. Using this technique, we limited the motion of each level to shifting left or right depending on the value of the resistance. Figure 4 shows a section of our proposed structure and how the resistor would fall down to the next level.

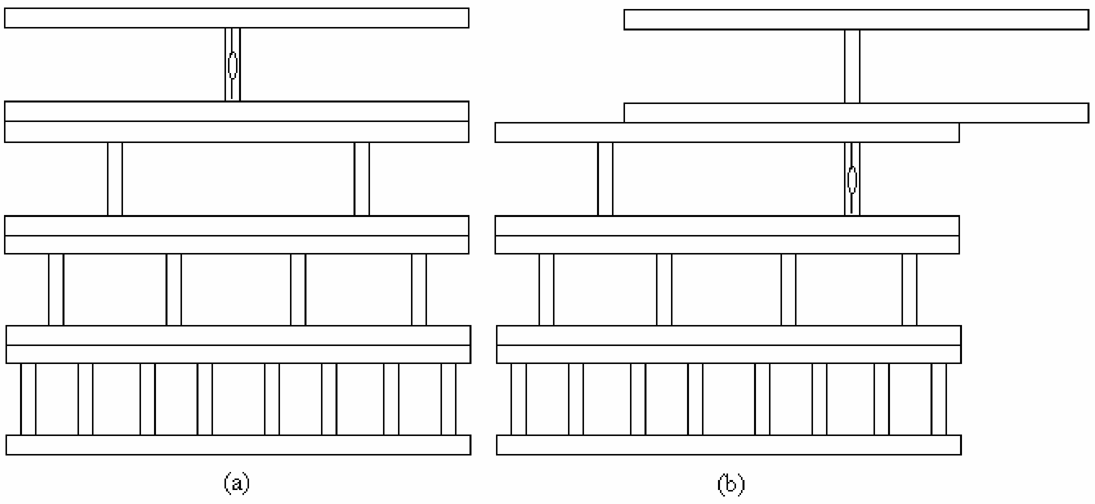


Figure 4 - (a) shows a section of the sorting pyramid in the "hold" state and (b) shows a section with the resistor moving down to the next level.

We knew that there were 96 different resistors that we would have to sort, which meant that we would need to have 128 or 2^7 tubes on the bottom level, and that we would need to have eight levels in the system.

Our next concern was how we would move each level. We came up with two ideas: motors and solenoids. Motors have the advantage of being able to move/rotate an unlimited distance. On the other hand, converting rotary motion to linear shifts would be difficult to implement, and the overall accuracy of motors is not extremely high. The only potential type of motor we could use that had a reasonable accuracy level was a servo motor which moves in finite steps. The major problem with this motor is that it was much more expensive and did not move as fast. Solenoids work much more quickly than motors and are very accurate. Unfortunately, solenoids can only move things very short distances and the force they supply varies inversely with the distance they can move an object. Figure 5 depicts a typical solenoid and how it moves.

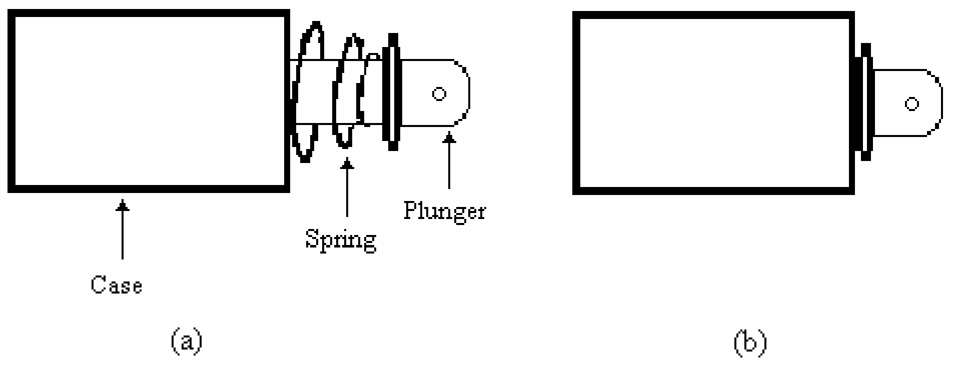


Figure 5 - This shows a typical pulling solenoid and its pieces. The solenoid is shown (a) off and (b) on.

Since the levels needed to move a large distance in a short amount of time in our initial design, neither motors nor solenoids presented an easy solution to moving the levels of the pyramid, and an alternate solution was needed.

In finding a new solution, we did not look for a different device to drive the motion of the system. Instead, we came up with a means of shortening the distance each level needed to move so that solenoids would be a viable option. We constructed a second design in which all of the tubes in the system were angled, thereby minimizing the distance each level moved. This reduction is illustrated in Figure 6.

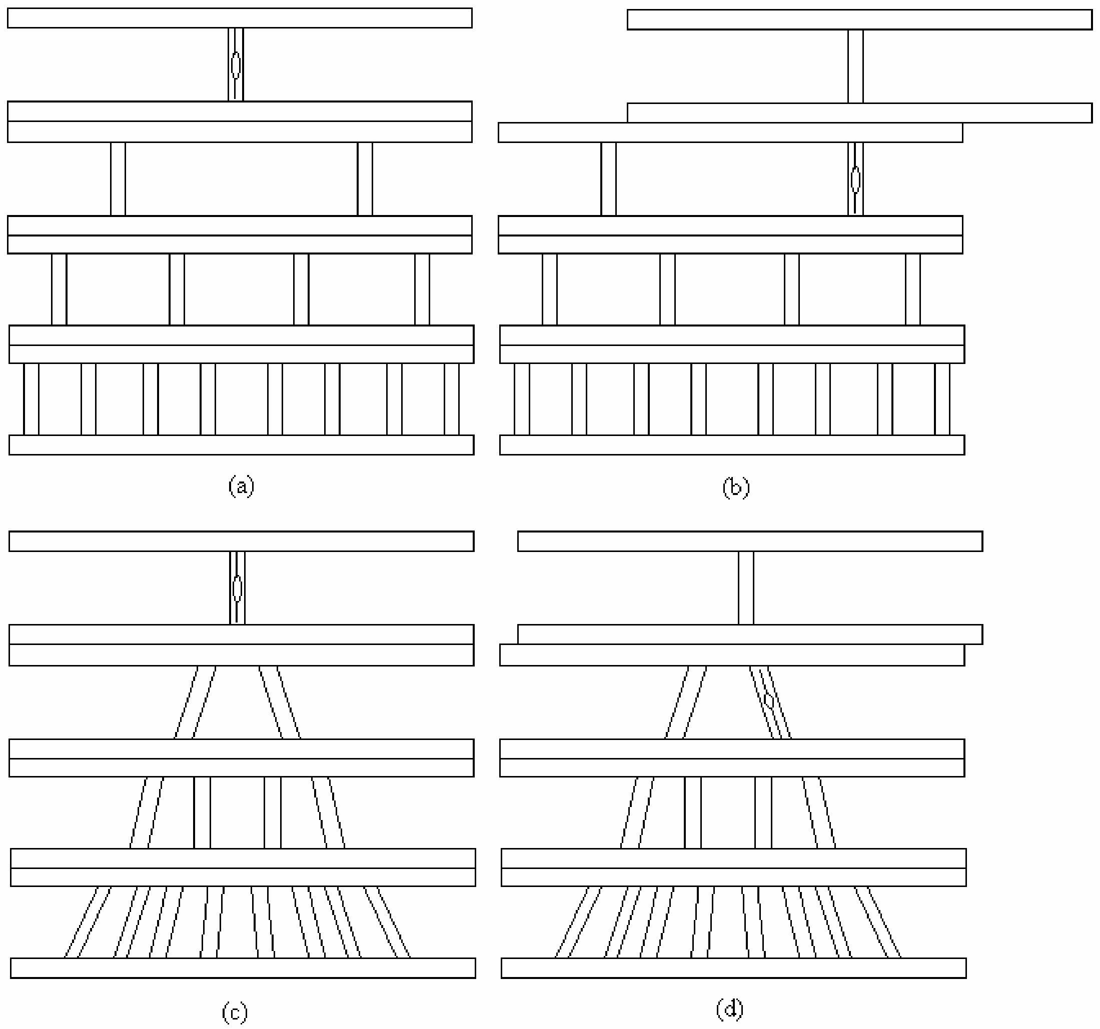


Figure 6 - This shows a comparison of our initial design with straight tubing in (a) and (b) and our design with angled tubes, holding the resistor in place in (c) and dropping it to the next level in (d). The key to the diagram is the reduction in the distance the levels have to move with the angled system.

Once our new design was established, we investigated several possible ways of cutting down on the overall size of the system. We were wary of the fact that if the tubes were angled too severely, the resistors might not be able to pass from one level to the next without getting stuck. Several solutions became apparent. First, we realized that we had 32 excess tubes on the bottom level that were not being used at all since there are only 96 different resistors. We eliminated these tubes and any other tube in the structure that would have eventually fed a resistor into these bottom tubes. Figure 7 shows how a section of our system was modified.

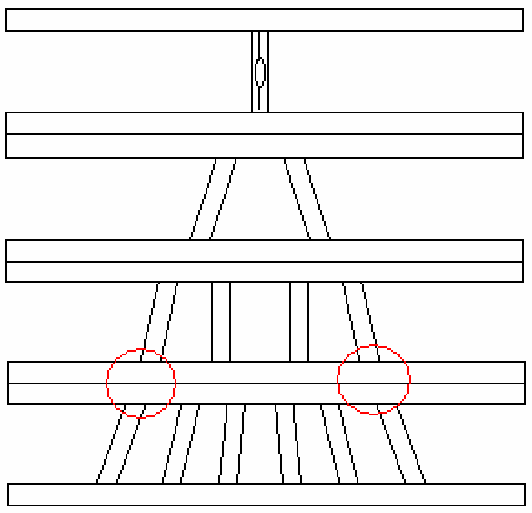


Figure 7 - This shows a reduction in size by eliminating one tube from the level. Now the resistor will fall straight to the next level if it is in either of the outside tubes.

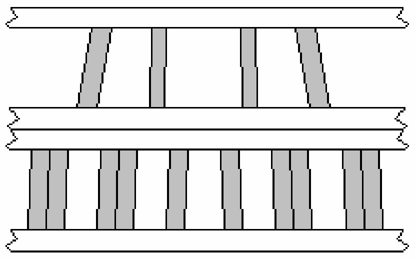


Figure 8 - This figure shows a section of the bottom level and how the spacing between some of the tubes was eliminated.

We were also able to reduce the size of the bottom level, and thus the whole structure, by an additional factor of one-third by positioning the bottom tubes so that they were flush against each other. We were only able to do this in the bottom level because the spacing and angles of the tubes in all of the other levels were directly determined from that of the bottom level. This reduction is illustrated in Figure 8. The final and most significant reduction we made cut the overall size of the system in half. We realized that we could make two symmetric pyramid structures, each with 48 tubes in the bottom level that could then be attached to each other. This concept is illustrated in Figure 9.

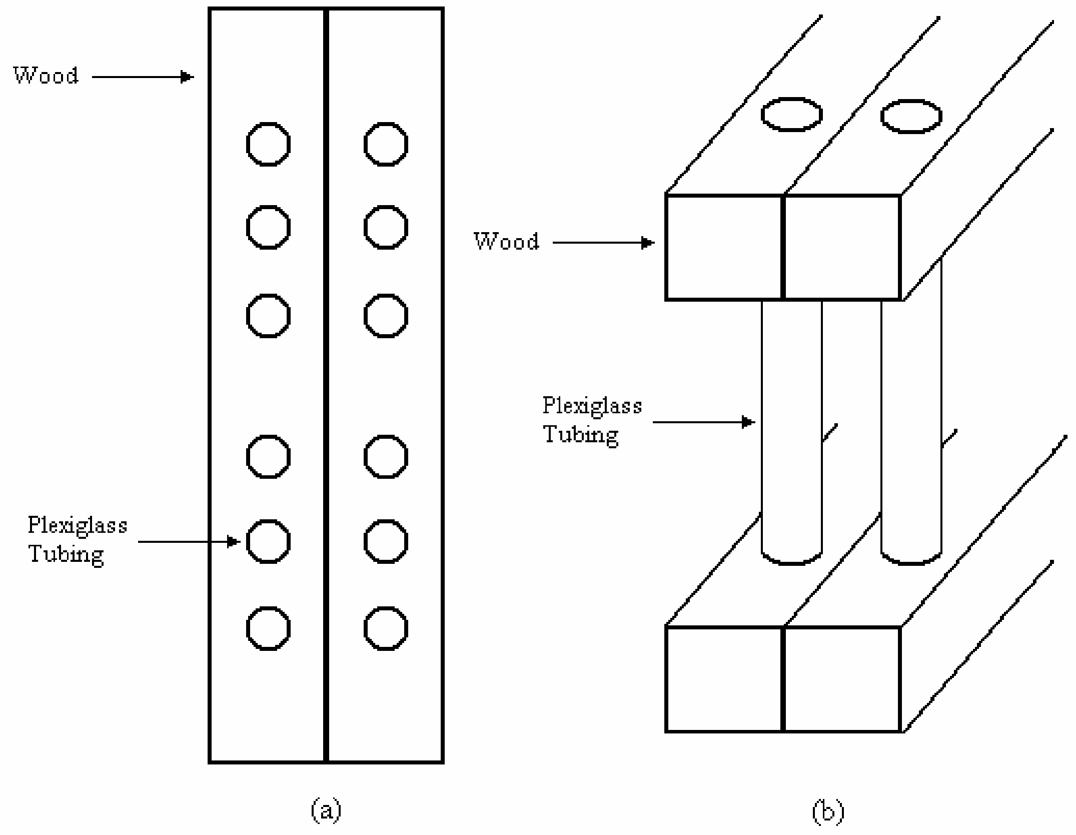


Figure 9 - This figure shows (a) a bird's eye view of the symmetrical pieces of one of the levels and (b) a side/end view of them.

The system would then function in the same way as originally intended, except that the first level would move perpendicularly to the rest of the levels. This perpendicular orientation can be seen clearly in Figure 10.

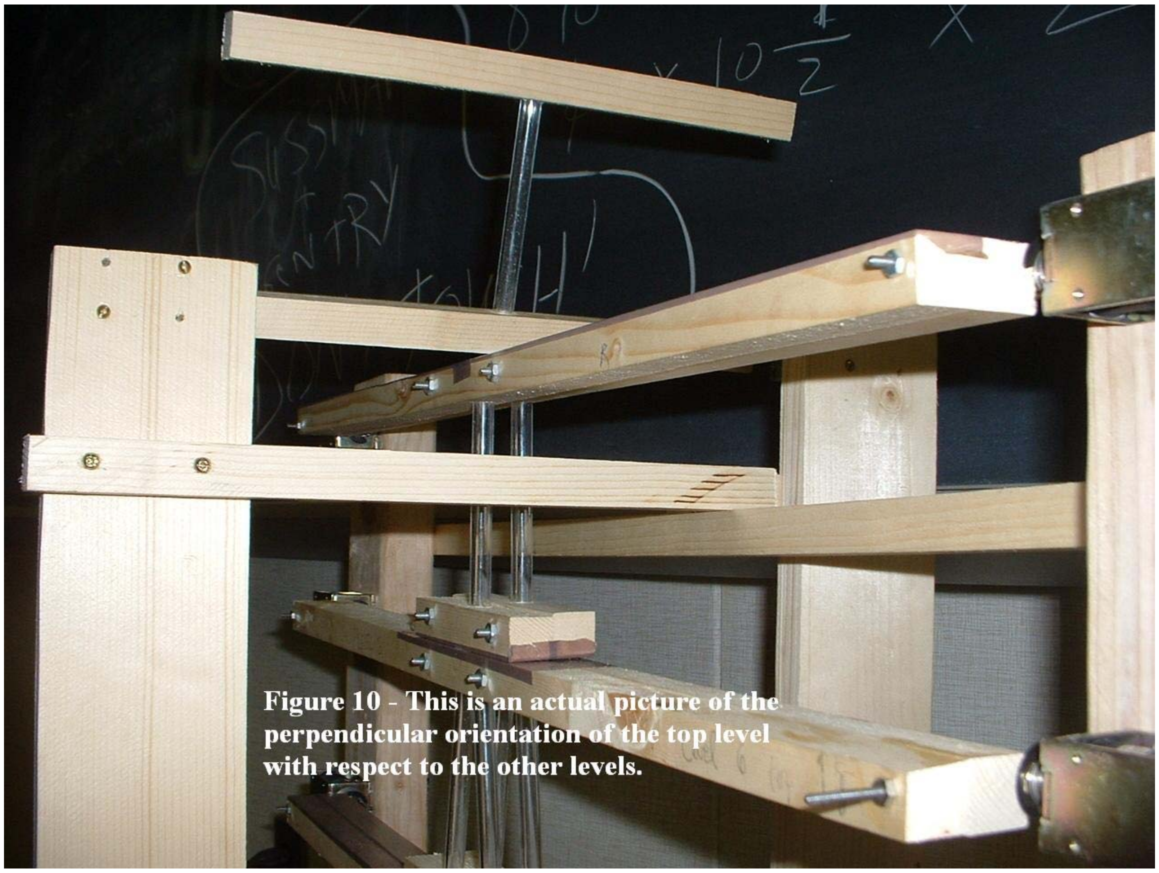


Figure 10 - This is an actual picture of the perpendicular orientation of the top level with respect to the other levels.

Procedure/System Construction

Resistance Reader

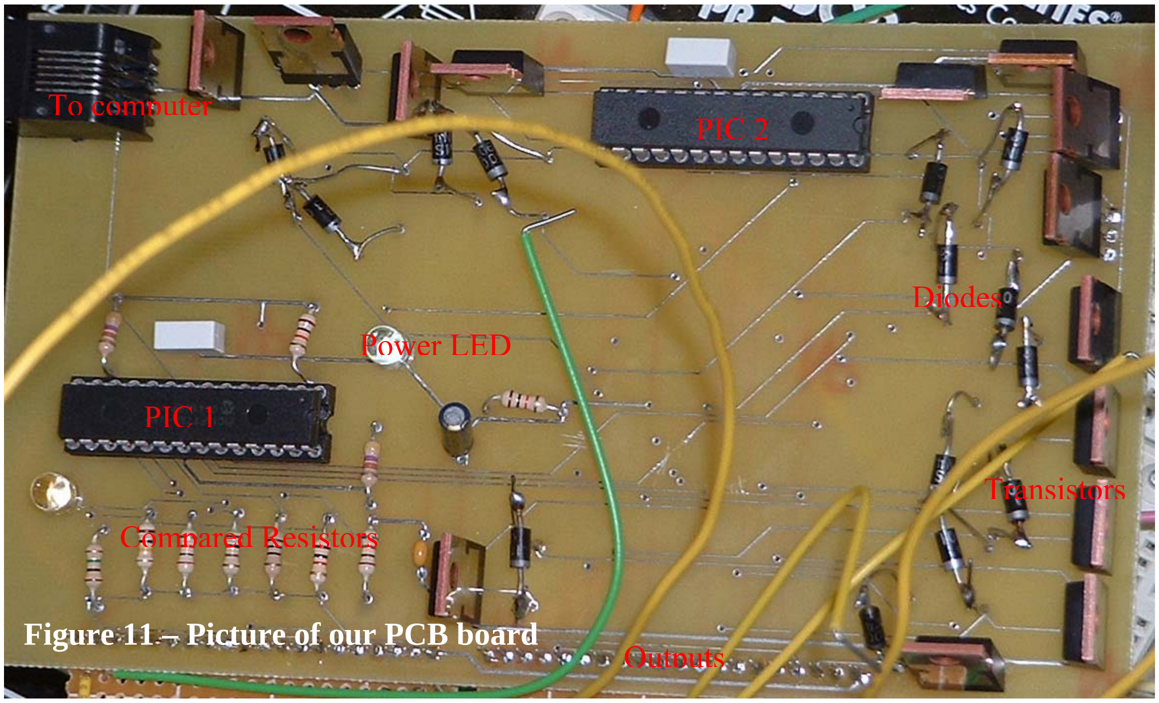


Figure 11 – Picture of our PCB board

To put all the electronic components of our project together in one entity, we decided to put everything on a PCB. In order to do this, we first had to identify all the components that we needed and make the appropriate connections in MultiSim (see Appendix). The circuit was then imported into Ultiboard where the components were placed. Basically we placed the components in a way that the connections were not very long and similar components were grouped together. The finished circuit board layout was then sent to a company by Professor Cheever to be produced. When the PCB arrived, we soldered the components into the appropriate places and tested the circuit.

The main part of the PCB is composed of a PIC chip, labeled PIC 1 in Figure 11, which is what holds the program that runs the resistance reader as previously described. The PIC is basically just a microprocessor that can hold programs downloaded into it using a phone-type jack that hooks into the USB port on a computer. The program was written in C in an application entitled PIC-C Compiler. It was using this software that we were able to debug the program by downloading it onto the chip and checking the chip's inputs and outputs with a voltmeter. Also, we could set the program to stop at a certain line to see the values of the variables and the outputs from the PIC. Because there were only a few input/output pins that were not being used by compared resistors, the unknown resistor, etc and we needed to have 14 outputs, we had to add a second PIC chip. This PIC chip, labeled PIC 2 is used as a demultiplexer. A demultiplexer takes in a small number of inputs and gives out a large number of outputs, which in this case is four inputs and 14 outputs (see Figure 12). It does this by using binary numbers to choose which output is high. For example, if the input number is 0000 where all the inputs are low, no output is turned on. If the input number is 0001, the first output is high and all the rest are low. If the input number is 0010, the second output is high and the rest are low. This procedure worked for our system because we only needed to turn one solenoid on at a time. Therefore, we could choose which solenoid we wanted on and at what time we wanted it on. Once again, the demultiplexer was programmed into the PIC chip using if-then statements. The second PIC did not have a jack to connect the PIC to a computer to download the demultiplexer program. Instead, the PIC was taken out of the PCB and placed on a different, smaller one to be programmed. Once the program was downloaded, the PIC was placed back our PCB. In some ways this produced problems as the outputs and/or inputs of the demultiplexer would be changed, and the only way to change the settings in the program was to move it back to the smaller PCB and reprogram it.

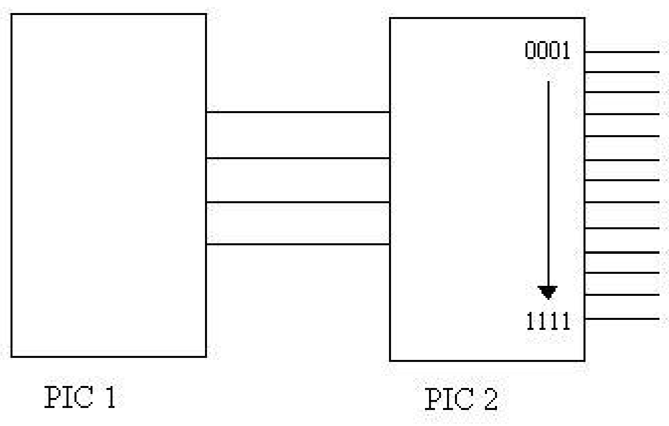


Figure 12 - This shows how the two PIC chips can be implemented to act like a demultiplexer.

Once the outputs came out of the demultiplexer each one was connected to the gate of a logic level transistor as seen in Figure 13. The reason transistors were used is because the PIC chip can only output a few milliamps, and each solenoid requires approximately 1A of current. The transistors work by acting as an open circuit when the gate, or output from PIC 2, is low, and acting as a short circuit when the gate is high. This way current only flows through the solenoid when the gate is high, and no current is flowing when the gate is low. Diodes are also placed in parallel with each of the solenoids so that when the transistor is suddenly shut off, the excess current can be dissipated through the solenoid instead of having nowhere to go.

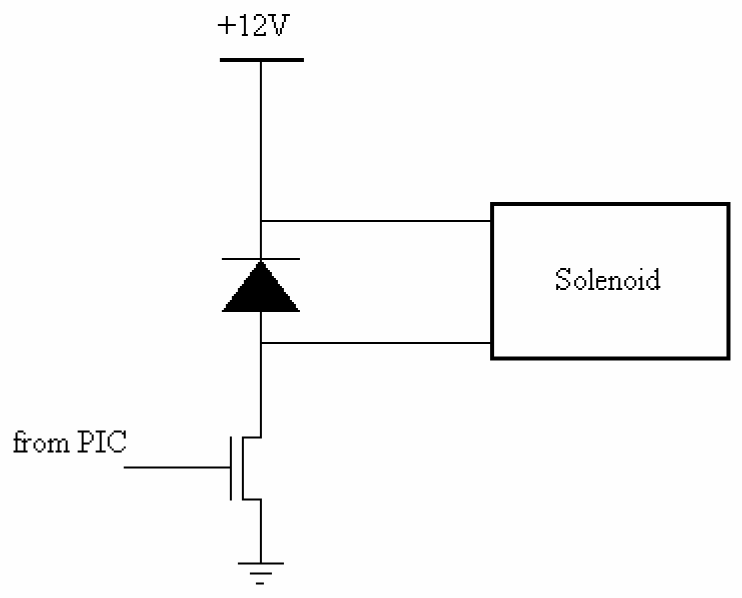


Figure 13 - This figures shows the circuit implemented to control each of the solenoids.

Also on the PCB are two LEDs. The green LED is the power LED and is used to determine if the PCB is in fact receiving power. The red LED is used for debugging problems, as it is easy to program the output from the chip to make the red LED turn on, or flash on and off at certain times. All of the inputs and outputs are placed on one of two 20-pin connectors located at the top of PCB. Since all the diodes are connected to 12V, all of their leads are connected together and used as just one output to the PCB, which is connected to 12V. There are then 14 other outputs (one for each of the solenoids), an input for the unknown resistor, approximately four input/output pins from PIC 1 for debugging purposes, two five volt pins to power the PIC chips, and one ground pin. The 20 pin connectors are soldered onto another, smaller board with three holes connected together. Wires are then soldered to the connections to create an easy way to connect the resistance reader to the rest of the structure. Each of the solenoids is connected to 12V and then back to the PCB to the appropriate pin (see Figure 13 for solenoid pinouts).

Output	PIC pin #	PCB Pinout	Working?
1	RB2	3	Yes
2	RB1	4	Yes
3	RB0	5	Yes
4	RA0	6	No
5	RA1	7	No
6	RA2	8	No
7	RA3	9	No
8	RA4	10	No
9	RA5	11	No
10	RC7	12	No
11	RC6	13	No
12	RC5	14	No
13	RC4	15	Yes
14	RC3	16	Yes

Table 1 – This table shows the outputs of the solenoids from PIC 2.

We had to use a wire to add one connection on the PCB because one end of the connection did not match up with the other end. Also, a few connections were changed because the outputs from the first PIC were not working properly. The pins that connected PIC 1 to PIC 2 were replaced as they did not work properly. These changes are summarized in Table 2, and are represented on the bottom of the PCB by thin, red wire.

Connection on PIC 1	Old PIC 2 Connection	New PIC 2 Connection
A0	B4	Disconnected
B0	B5	C0
B1	B6	C1
B2	B7	C2

Table 2 – This table shows the changes made manually on the PCB board.

A power source that had both a 12V at 1.5A and a 5V wire was purchased for $16 from Jameco. Unfortunately, the power source did not work when we plugged it into a wall socket. The reasons for this could be that the adapter we used to connect the power source to the wall was dysfunctional or not the correct type, or it may just have been a faulty power source. For demonstration purposes, we connected the PCB to a breadboard.

Sorting Structure

Each of the levels was constructed out of $\frac{3}{4}'' \times \frac{3}{4}''$ wood and plexiglass tubing which was $\frac{1}{4}''$ inner diameter and $\frac{3}{8}''$ outer diameter. The size of the plexiglass tubing was chosen after comparing several different diameters and seeing which allowed the resistors to move easily without getting too large. We then chose the wood sizing so that the tubes would fit through the wood with a little bit of extra wood on either side so it would not break when we drilled holes in it.

Since the wood came initially in sheets that were $\frac{3}{4}''$ thick, we used the table saw to cut wood rods to our specified parameters. After all of the wood had been cut, we moved on to drilling all of the holes for the tubes. Drilling holes ended up being one of the most time consuming aspects of the project. For each piece of wood, the following steps were taken:

1. Taking a piece of wood and clamping it to the drill press
2. Finding the center of the wood and marking the exact location for each of the holes we were about to drill using the digital measuring device attached to the drill
3. Setting the wood to a specific angle (which was different for each of the roughly 200 holes we had to drill); Table 3 shows the angles we had to drill

4. Align the drill bit with the marked location on the wood and center drill the hole
5. Remove the center drill bit and insert a bit slightly smaller in diameter than .375” (the OD of the plexiglass) and drill the hole
6. Remove that bit and insert a third bit that was .377” in diameter to bore the hole out to make it fit tightly around the plexiglass tubing[1]
7. Re-insert the center drill bit and repeat steps 3-6

Level	Angle(s)																							
7	0																							
6	9																							
5	9								4															
4	31								19								6							
3	40				35				28				21				13				5			
2	29		27		24		22		19		17		14		11		9		6		3		0	
1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Table 3 - This shows the angles for each level of the pyramid. Note that only half of the angles are shown because the other half of the pyramid is a mirror image.

After all of the holes had been drilled, we had to cut the plexiglass tubing into pieces so that we could assemble the levels. We wanted the levels to be five inches tall because the length of the resistors was between three and four inches, and because the taller each level was, the less severe its angles would be. The sizes of the tubes that we needed are given in Table 4. We cut all of the tubes slightly longer than we needed them to be, so that there would be extra room for the tubes to extend from the wood.

[1] We wanted to make the holes snug enough so that we wouldn’t have to do any unnecessary gluing to make the tubes stay in the wood.

Level	Length(s)																							
7	5																							
6	5.06																							
5	5.06												5.01											
4	5.83								5.30								5.03							
3	6.52				6.07				5.67				5.36				5.14				5.02			
2	5.71		5.59		5.48		5.39		5.30		5.22		5.15		5.10		5.06		5.03		5.01		5.00	
1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Table 4 - This shows the lengths of the tubes for each level of the pyramid. Note that only half of the lengths are shown because the other half of the pyramid is a mirror image.

Once all of the individual pieces had been prepared, we were ready to begin the construction of the levels. After fitting all of the tubes into the holes and measuring each level to ensure that they were all approximately five inches tall, we cutoff the excess tubing on the tops and bottoms of the levels. Even after the tubes were cut down, the tops and bottoms of the levels were still pretty rough, so we hit them all on the power sander to get rid of any potential imperfections in the surface of the wood that might snag the end of a resistor.

The next step was to combine the symmetric pieces for each level that we had built. Unfortunately, our design had not accounted for the natural effects of climate changes on wood, i.e. many if not all of the pieces of wood had warped and were bowed in different directions. Figure 14 depicts an example of this type of situation.

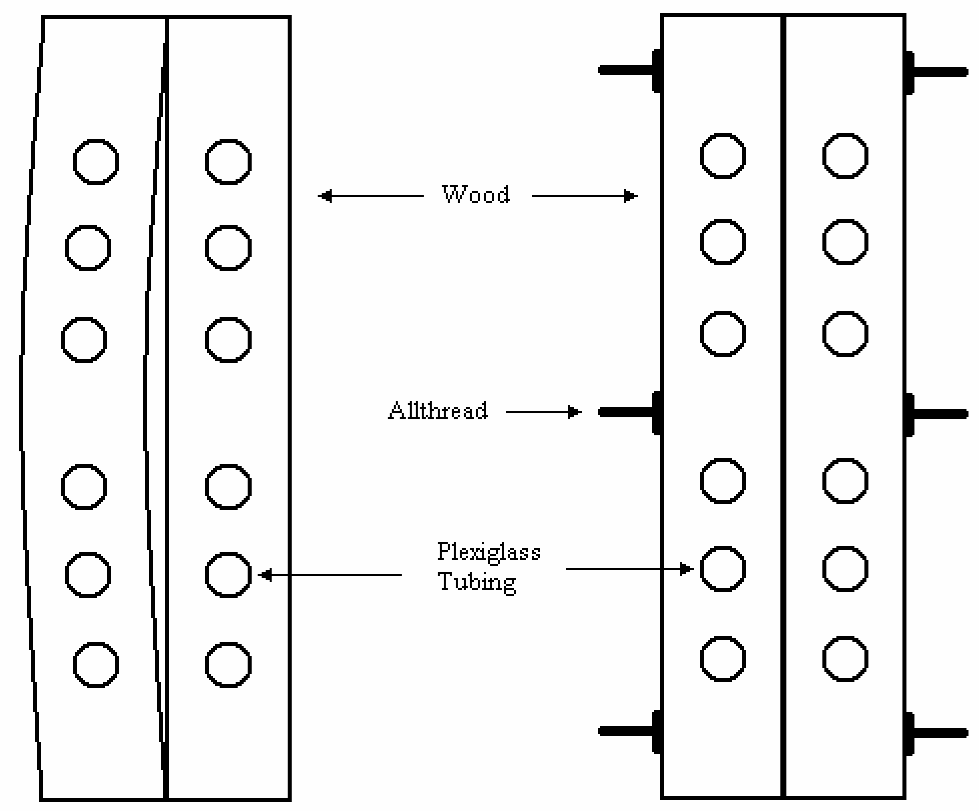


Figure 14- This is an example of the problems of wood warping and how we overcame it using allthread and bolting the symmetric pieces of each level together.

To fix the imperfections in the wood, we used all-thread which is just long cylindrical material that is threaded so it can act like a screw. We cut two inch pieces of the all-thread and then drilled clearance holes through each of the levels to fit the pieces. We fastened nuts on either end of the all-thread so that the two pieces of wood were pinched against each other so the warping was no longer an issue. After each of the levels had been put together, they were still uneven on the tops and bottoms, so they all had to be sanded again until they were level and could stand alone.

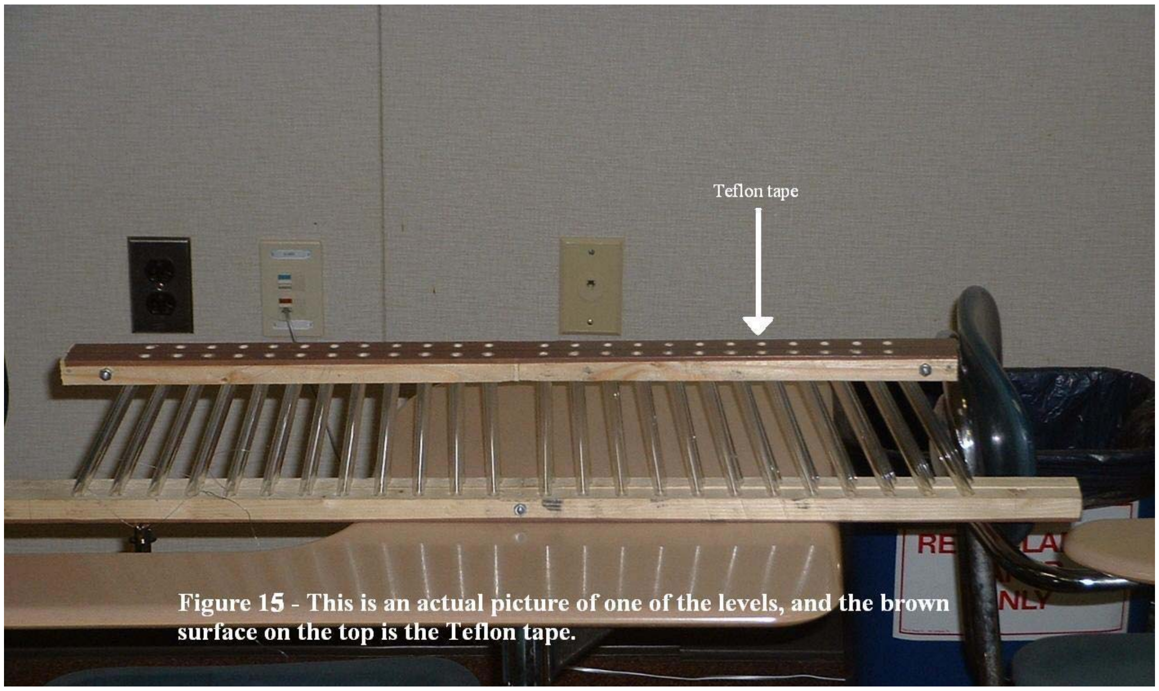
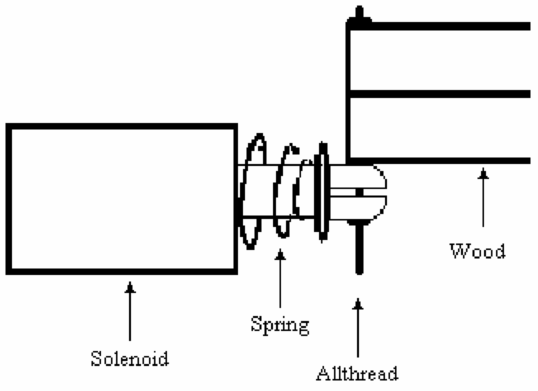


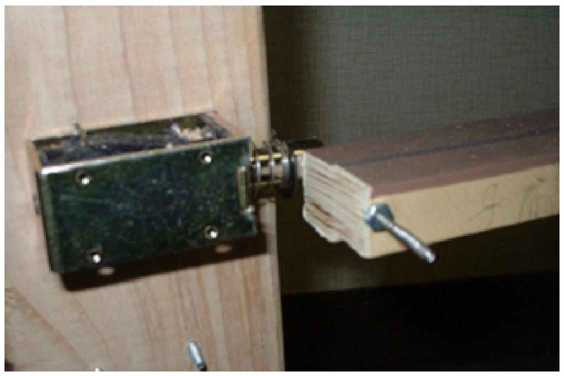
Figure 15 - This is an actual picture of one of the levels, and the brown surface on the top is the Teflon tape.

The last thing we had to do was reduce the overall amount of friction in the system so that the levels would not get stuck against each other when they were sliding. To do this we used Teflon tape, which is a material commonly used in making cooking utensils because it does not stick to anything and it is nearly frictionless. By putting Teflon tape with a thickness of only .003" across the top and bottom of each level (as seen in Figure 15), we were able to minimize the amount of friction in our system. After the tape was put down, we had to go through and carve out holes for each of the tubes using an exacto knife.

After all of the levels had been completed, we were ready to start attaching them to the solenoids and begin building a support structure for the whole system. To attach the solenoids to the levels, we again used the all-thread. We drilled clearance holes through the levels, cut pieces of the all-thread and fastened them on either end, sandwiching the plunger of the solenoid and the pieces of wood. Figure 16 shows both a theoretical and actual model of the levels attached to the solenoids.



(a)



(b)

Figure 16 - This shows (a) a schematic from a bird's eye view of how the solenoid was attached to the level and (b) an actual picture of how the solenoid was attached.

This method of attaching the solenoids was advantageous for our system because it allowed us to remove each level individually to make any repairs or to remove any resistors which got stuck during the sorting process.

Our design specified that each level only needed to move left or right in the sorting process, but we also wanted to be able to "hold" a resistor in each level. To do this, we needed to have three states for each level: left, right and hold. This proposed a challenge because each solenoid could only pull, so we needed a way to generate a third state by equipping the plunger of each solenoid with a spring so that after the solenoid was deactivated, the springs on either end of the level would push/pull the level back into the middle/hold position. Figure 17 illustrates this spring-reset system.

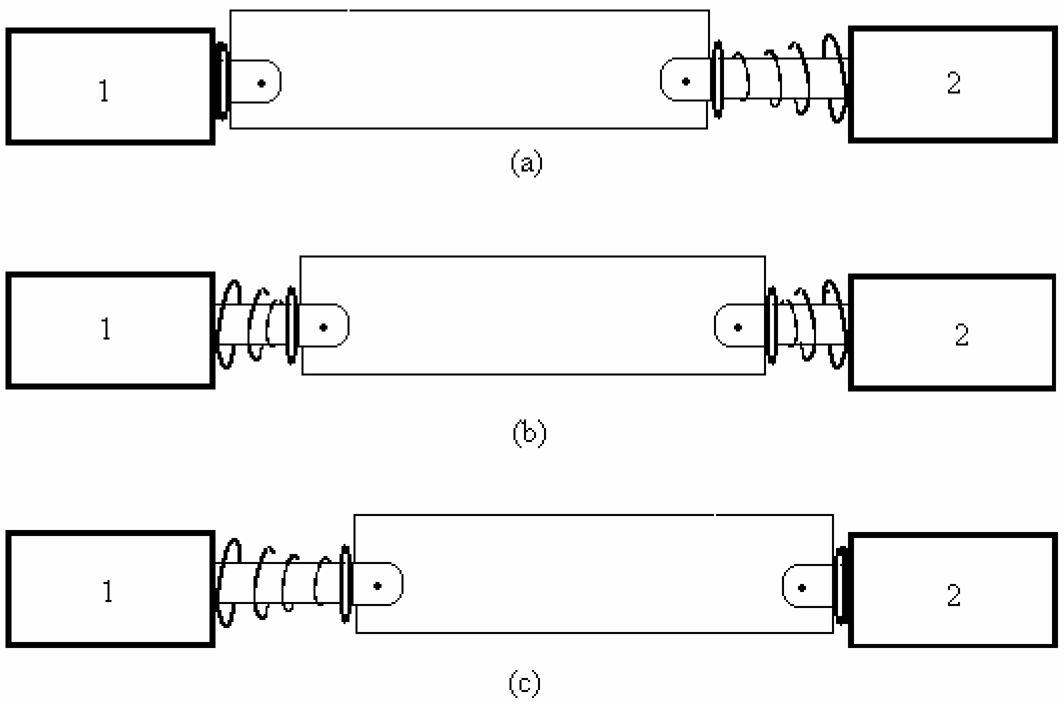


Figure 17 - This figure shows the spring reset system with (a) solenoid 1 on, solenoid 2 off, (b) solenoid 1 off, solenoid 2 off, (c) solenoid 1 off, solenoid 2 on.

Support Structure

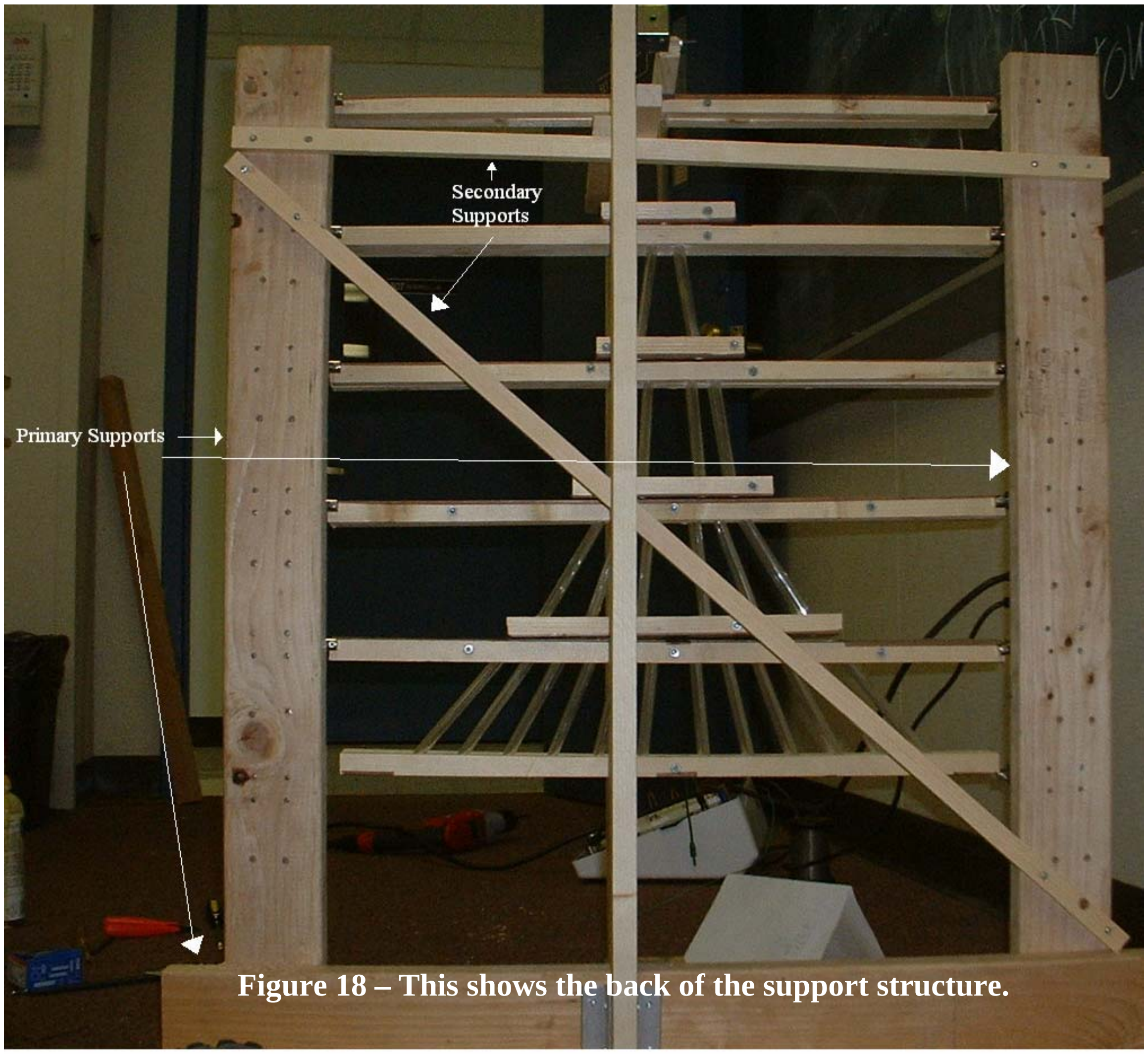


Figure 18 – This shows the back of the support structure.

The main support structure (as seen in Figure 18) was created as a way to hold the solenoids and levels in place. It consists of two 2x4s placed in the vertical direction approximately 30” apart from one another, and another 2x4 placed in the horizontal direction on the bottom attached to the other 2x4s. Also, there is a large piece of plywood attached to the bottom of the structure to provide a solid base. Another, secondary piece of wood that is approximately $\frac{3}{4}'' \times \frac{3}{4}''$ is placed in the diagonal direction and attached to the back of the vertical 2x4s. Finally, a $\frac{3}{4}'' \times \frac{3}{4}''$ piece is placed across the top of the structure connecting the two 2x4s. These secondary supports are used to help prevent the structure from wobbling or becoming misaligned.

The first task was to cut the two vertical 2x4s. We found the 2x4s leftover in Papazian Basement and decided they would provide firm supports. We cut them into pieces approximately 36" tall to accommodate the solenoids placed at 5" apart from each other, as that was the height of each of the levels, and seven solenoids needed to be attached. Once the wood was cut into the appropriate length, we measured the exact position of where we wanted each of the solenoids. We wanted the plunger of the first solenoids to be centered around $4\frac{5}{8}''$ up from the ground. We found this number by dividing the $\frac{3}{4}''$ pieces by 2 to center them, and then subtracting that number from 5". Each of the next solenoids was then placed 5" up from the one below it. We marked four holes for each solenoid, one for each of the screw holes. We then determined the size of the screws that would fit into the solenoids by testing a few and found them to be size 4-40 screws. Taking a drill bit slightly larger than the diameter of the screw, we drilled the four holes for each of the solenoids. Because the screws were not long enough to fit completely through the 2x4s, however, we had to drill another hole, one slightly larger than the head of the screw, part of the way into the 2x4. We cut into each piece by approximately $\frac{1}{2}''$ to $1''$, so the end of the screw could come out the other end of the wood by a small margin. We cut these larger holes on the opposite side of the 2x4s in which the smaller holes were initially drilled. We did this because the drill tends to drift when it is drilling, and we wanted the holes to line up perfectly with the solenoids. Once all the holes were drilled, we screwed in the solenoids to the 2x4s using a screwdriver. We did not add nuts as the solenoids remained tight without them, and we thought it would be unnecessary.

After attaching the solenoids to the 2x4s, we put the third 2x4 across the bottom. We put the two vertical pieces approximately 30" apart to give room to add the solenoids to the lowest levels. We fastened these pieces together using wood screws and an electric drill. We drilled small clearance holes into the wood, and then used a flathead drill bit to drill in the wood screws. Finally, the piece at the diagonal and the piece across the top were added. They were also drilled in with wood screws, using two for each piece at each end, for eight wood screws total.

Once the outside structure was complete, we attached the bottom level (see Figure 19). Like the other levels, we wanted to attach the bottom level in a way that it could be unattached. First, we determined how far away from the 2x4s each of the levels would be hanging, and we cut a piece of wood to that length. We then cut two pieces to a length approximate to the height of the first level and nailed the two pieces into the 2x4s at either end.

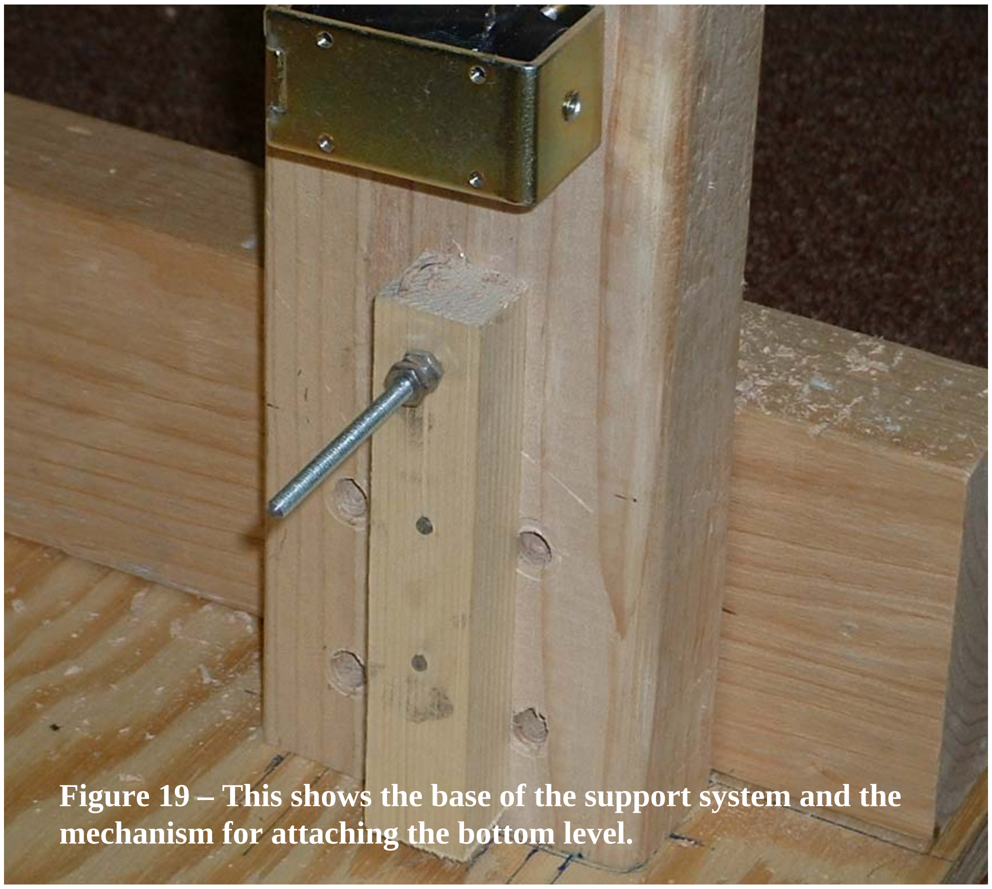


Figure 19 – This shows the base of the support system and the mechanism for attaching the bottom level.

In addition to the previously mentioned supports, we needed separate support for the top level. Figure 20 shows a picture of these supports. We used two $2'' \times \frac{3}{4}''$ pieces of wood to hold the solenoids, and then two aluminum "L" brackets at the bottom of each to attach to the plywood base and provide extra stability. We then added a cross piece of $\frac{3}{4}'' \times \frac{3}{4}''$ wood to stabilize them at the tops. For the top level, the solenoids were mounted and attached to the bottom of the level as opposed to the top as they had been for every other level.

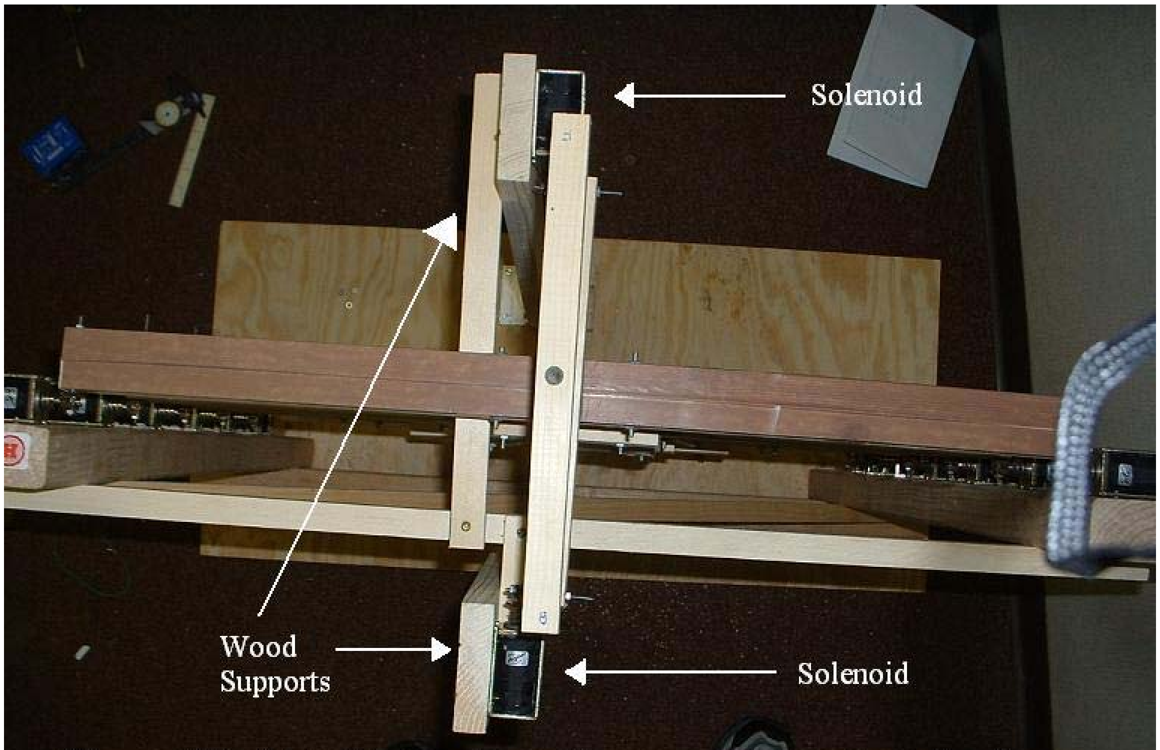


Figure 20 - This figure shows the support structure for the top level in the system.

Conclusions and Future Extensions

There are two main problems with the mechanical part of our resistor sorter: the friction between the levels and the lining up of the levels. First of all, there is a lot of friction created by the levels pressing against each other as well as friction created by the hanging of the levels from the solenoids. Therefore, whenever the space between the levels is small, the levels push against each other, creating friction. This problem is largest for the levels towards the bottom of the structure as they are carrying the weight of all the levels above them. Currently, the solenoids are barely able to move these levels, and even when they do, the springs are unable to shift the levels back to the hold position. One answer to this problem may be to sand down the levels to create more distance between each of the levels. We tried this solution, and it did not work because whenever the space between the levels is large, the levels hang from the solenoids, creating friction between the solenoid plunger and the solenoid box. Also, because the levels are hanging from the solenoids, they tend to shift back and forth so that the tubes no longer match. When resistors are falling down from one level to the next, the resistors often become stuck unless the levels are moved by hand or the levels are removed entirely. Basically, both of these problems are caused by the current structure of the resistor sorter. Because the levels are only held in place by the solenoids and the levels below them, these problems are created. Figuring out some way to keep the levels in place and taking the load off of the other levels is essential. This would require building a better support structure around each of the levels. Perhaps attaching some sort of flat angle-iron to the 2x4s on either side will make the levels slide in the appropriate place while providing more support.

On the electrical side, the main problems came from the second PIC that was acting as a demultiplexer. For some reason, only five of the outputs would work (see Table 1). We know that these are the only ones that work because there are specific times when our program changed certain outputs when certain inputs changed, and some outputs simply did not change. The working outputs were used to do our demonstration and are the only outputs currently hooked up to the solenoids. This may be because we do not know how to program the chip correctly, or it may be because some of the outputs can only be used as inputs, or are already set up for a different purpose. To fix this problem, more research can be done on how PICs work, and how they can be programmed to produce more functioning output.

Appendix

resistor #	resistance	state	min	max
1	1	6	0	323
2	10	6	324	537
3	12	6	538	589
4	15	6	590	638
5	18	6	639	671
6	20	6	672	694
7	22	6	695	727
8	27	6	728	758
9	30	6	759	777
10	33	6	778	801
11	39	6	802	830
12	47	6	831	850
13	51	6	851	862
14	56	6	863	881
15	68	6	882	898
16	75	6	899	908
17	82	6	909	917
18	91	6	918	926
19	100	6	927	1023
19	100	5	513	537
20	120	5	538	589
21	150	5	590	638
22	180	5	639	671
23	200	5	672	694
24	220	5	695	727
25	270	5	728	758
26	300	5	759	777
27	330	5	778	801
28	390	5	802	830
29	470	5	831	850
30	510	5	851	862
31	560	5	863	881
32	680	5	882	898
33	750	5	899	908
34	820	5	909	917
35	910	5	918	926
36	1000	5	927	1023
36	1000	4	513	537
37	1200	4	538	589
38	1500	4	590	638
39	1800	4	639	671
40	2000	4	672	694
41	2200	4	695	727
42	2700	4	728	758
43	3000	4	759	777
44	3300	4	778	801
45	3900	4	802	830
46	4700	4	831	850
47	5100	4	851	862
48	5600	4	863	881
49	6800	4	882	898
50	7500	4	899	908
51	8200	4	909	917
52	9100	4	918	926
53	10000	4	927	1023
53	10000	3	513	537
54	12000	3	538	589
55	15000	3	590	638
56	18000	3	639	671
57	20000	3	672	694
58	22000	3	695	727
59	27000	3	728	758
60	30000	3	759	777
61	33000	3	778	801
62	39000	3	802	830
63	47000	3	831	850
64	51000	3	851	862
65	56000	3	863	881
66	68000	3	882	898
67	75000	3	899	908
68	82000	3	909	917
69	91000	3	918	926
70	100000	3	927	1023
70	100000	2	513	537
71	120000	2	538	589
72	150000	2	590	638
73	180000	2	639	671
74	200000	2	672	694
75	220000	2	695	727
76	270000	2	728	758
77	300000	2	759	777
78	330000	2	778	801
79	390000	2	802	830
80	470000	2	831	850
81	510000	2	851	862
82	560000	2	863	881
83	680000	2	882	898
84	750000	2	899	908
85	820000	2	909	917
86	910000	2	918	926
87	1000000	2	927	1023
87	1000000	1	513	568
88	1500000	1	569	651
89	2000000	1	652	694
90	2200000	1	695	738
91	3000000	1	739	793
92	3900000	1	794	830
93	4700000	1	831	850
94	5100000	1	851	862
95	5600000	1	863	875
96	6200000	1	876	887
97	6800000	1	888	898
98	7500000	1	899	908
99	8200000	1	909	922
100	10000000	1	923	1023

Table 5 – Digital Ranges of Input Voltages to the PCB Board

Resistance Reader Programs:

PIC 1

```
#include "PIC_A_D.h"

main() {
  signed int16 m;
            long int charlie;
            int w=0;
            int res;
            int h=0;
            int g=0;
            int k=0;

  //These next 6 lines set certain hardware on the processor.
  //Ignore them for now.
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  //Set up adc port to read from channel 1 (pin AN1)
  setup_adc(ADC_CLOCK_INTERNAL);
  setup_adc_ports(ALL_ANALOG);
  set_adc_channel(1);

  //Blink PIC LED (sanity check to make sure all is working).
            output_low(pin_c4);
            output_low(pin_c5);
  output_low(pin_b4);
  delay_ms(500);
  output_high(pin_b4);
            delay_ms(500);
            output_low(pin_b4);
  delay_ms(500);
  output_high(pin_b4);
            delay_ms(500);
            output_high(pin_c7);
            output_low(pin_a0);
            output_low(pin_b0);
            output_low(pin_b1);
            output_low(pin_b2);

  //Main loop
  while (h<7) {

                        charlie = read_adc();
                        h=h+1;
                        if (charlie > 512)
                                    w=w;
                        else
                                    w=w+1;
```

```
if (w==0) {
        delay_ms(500);
        //output_low(pin_c4);
        //output_low(pin_c5);
        //Outputs "00" or state 0 on LEDs
        output_float(pin_c0);
        output_float(pin_c1);
        output_float(pin_c2);
        output_float(pin_a2);
        output_float(pin_c4);
        output_high(pin_c5);

        delay_ms(500);
        if (charlie >= 923) {          res= 100;  }
        else if (charlie >=909)  {   res=99; }
        else if (charlie >=899)  {   res=98; }
        else if (charlie >=888)  {   res=97; }
        else if (charlie >=876)  {   res=96; }
        else if (charlie >=863)  {   res=95; }
        else if (charlie >=851)  {   res=94; }
        else if (charlie >=831)  {   res=93; }
        else if (charlie >=794)  {   res=92; }
        else if (charlie >=739)  {   res=91; }
        else if (charlie >=695)  {   res=90; }
        else if (charlie >=652)  {   res=89; }
        else if (charlie >=569)  {   res=88; }
        else  {    res=87; }



}
else if (w==1) {
        delay_ms(500);
        //output_low(pin_c4);
        //output_high(pin_c5);
        //Outputs "01" or state 1 on LEDs
        output_float(pin_c0);
        output_float(pin_c1);
        output_float(pin_c2);
        output_float(pin_a2);
        output_high(pin_c4);
        output_float(pin_c5);

        delay_ms(500);
        if (charlie >= 927) {          res= 87;  }
        else if (charlie >=918)  {   res=86; }
        else if (charlie >=909)  {   res=85; }
        else if (charlie >=899)  {   res=84; }
        else if (charlie >=882)  {   res=83; }
        else if (charlie >=863)  {   res=82; }
        else if (charlie >=851)  {   res=81; }
        else if (charlie >=831)  {   res=80; }
        else if (charlie >=802)  {   res=79; }
        else if (charlie >=778)  {   res=78; }
        else if (charlie >=759)  {   res=77;}
```

```
            else if (charlie >=728)  {    res=76;}
            else if (charlie >=695)  {    res=75;}
            else if (charlie >=672)  {    res=74;}
            else if (charlie >=639)  {    res=73; }
            else if (charlie >=590)  {    res=72; }
            else if (charlie >=538)  {    res=71; }
            else  {    res=70; }

    }
    else if (w==2) {
            delay_ms(500);
            //output_high(pin_c4);
            //output_low(pin_c5);
            //Outputs "10" or state 2 on LEDs
            output_float(pin_c0);
            output_float(pin_c1);
            output_float(pin_c2);
            output_high(pin_a2);
            output_float(pin_c4);
            output_float(pin_c5);
            output_high(pin_c6);
            output_low(pin_a5);
            delay_ms(500);
            if (charlie >= 927) {          res= 70;  }
            else if (charlie >=918)  {    res=69; }
            else if (charlie >=909)  {    res=68; }
            else if (charlie >=899)  {    res=67; }
            else if (charlie >=882)  {    res=66; }
            else if (charlie >=863)  {    res=65; }
            else if (charlie >=851)  {    res=64; }
            else if (charlie >=831)  {    res=63; }
            else if (charlie >=802)  {    res=62; }
            else if (charlie >=778)  {    res=61; }
            else if (charlie >=759)  {    res=60; }
            else if (charlie >=728)  {    res=59; }
            else if (charlie >=695)  {    res=58; }
            else if (charlie >=672)  {    res=57; }
            else if (charlie >=639)  {    res=56;}
            else if (charlie >=590)  {    res=55; }
            else if (charlie >=538)  {    res=54; }
            else  {    res=53; }

    }
    else if (w==3) {
            delay_ms(500);
            //output_high(pin_c4);
            //output_high(pin_c5);
            //Outputs "11" or state 3 on LEDs
            output_float(pin_c0);
            output_float(pin_c1);
            output_high(pin_c2);
            output_float(pin_a2);
            output_float(pin_c4);
            output_float(pin_c5);
            output_high(pin_a5);
            delay_ms(500);
```

```
        if (charlie >= 927) {        res= 53;  }
        else if (charlie >=918)  {   res=52; }
        else if (charlie >=909)  {   res=51; }
        else if (charlie >=899)  {   res=50; }
        else if (charlie >=882)  {   res=49; }
        else if (charlie >=863)  {   res=48; }
        else if (charlie >=851)  {   res=47;}
        else if (charlie >=831)  {   res=46;}
        else if (charlie >=802)  {   res=45; }
        else if (charlie >=778)  {   res=44; }
        else if (charlie >=759)  {   res=43; }
        else if (charlie >=728)  {   res=42; }
        else if (charlie >=695)  {   res=41; }
        else if (charlie >=672)  {   res=40; }
        else if (charlie >=639)  {   res=39; }
        else if (charlie >=590)  {   res=38; }
        else if (charlie >=538)  {   res=37; }
        else  {    res=36; }

}
else if (w==4) {
        delay_ms(500);
        //output_high(pin_c4);
        //output_high(pin_c5);
        //Outputs "11" or state 3 on LEDs
        output_float(pin_c0);
        output_high(pin_c1);
        output_float(pin_c2);
        output_float(pin_a2);
        output_float(pin_c4);
        output_float(pin_c5);
        delay_ms(500);
        if (charlie >= 927) {        res= 36;  }
        else if (charlie >=918)  {   res=35; }
        else if (charlie >=909)  {   res=34; }
        else if (charlie >=899)  {   res=33; }
        else if (charlie >=882)  {   res=32; }
        else if (charlie >=863)  {   res=31; }
        else if (charlie >=851)  {   res=30; }
        else if (charlie >=831)  {   res=29; }
        else if (charlie >=802)  {   res=28; }
        else if (charlie >=778)  {   res=27; }
        else if (charlie >=759)  {   res=26; }
        else if (charlie >=728)  {   res=25; }
        else if (charlie >=695)  {   res=24; }
        else if (charlie >=672)  {   res=23; }
        else if (charlie >=639)  {   res=22; }
        else if (charlie >=590)  {   res=21; }
        else if (charlie >=538)  {   res=20; }
        else  {    res=19; }

}
else {
        delay_ms(500);
        //output_high(pin_c4);
        //output_high(pin_c5);
```

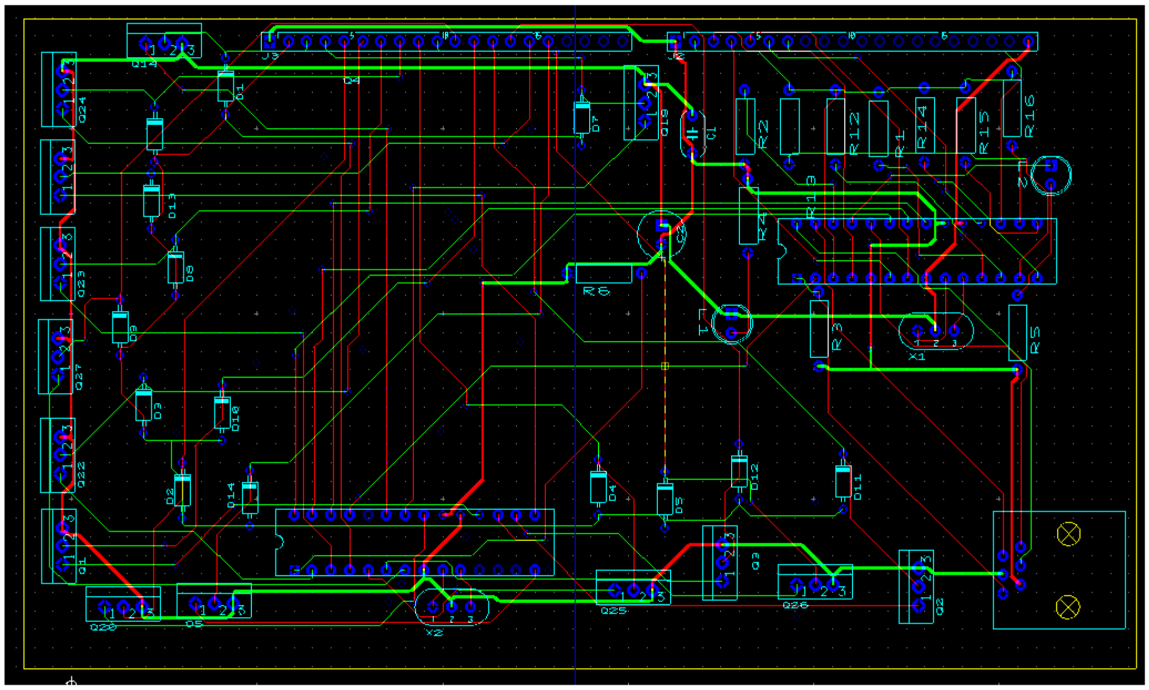
```
                        //Outputs "11" or state 3 on LEDs
                        output_high(pin_c0);
                        output_float(pin_c1);
                        output_float(pin_c2);
                        output_float(pin_a2);
                        output_float(pin_c4);
                        output_float(pin_c5);
                        delay_ms(500);
                        if (charlie >= 927) {          res= 19;  }
                        else if (charlie >=918)  {    res=18; }
                        else if (charlie >=909)  {    res=17; }
                        else if (charlie >=899)  {    res=16; }
                        else if (charlie >=882)  {    res=15; }
                        else if (charlie >=863)  {    res=14; }
                        else if (charlie >=851)  {    res=13; }
                        else if (charlie >=831)  {    res=12; }
                        else if (charlie >=802)  {    res=11; }
                        else if (charlie >=778)  {    res=10; }
                        else if (charlie >=759)  {    res=9; }
                        else if (charlie >=728)  {    res=8; }
                        else if (charlie >=695)  {    res=7; }
                        else if (charlie >=672)  {    res=6; }
                        else if (charlie >=639)  {    res=5; }
                        else if (charlie >=590)  {    res=4; }
                        else if (charlie >=538)  {    res=3; }
                        else if (charlie >=460)  {    res=2; }
                        else  {    res=1; }
                        w=7;

                }
        }
                delay_ms(3000);
                g=res/48;
                if (g == 1) { output_low(pin_a0); output_low(pin_b0); output_low(pin_b1);
output_high(pin_b2); k=48;}
                else { output_low(pin_a0); output_low(pin_b0); output_high(pin_b1);
output_low(pin_b2);k=0;}
                delay_ms(2000);
                res=res-k;

                g=res/24;
                if (g == 1) { output_low(pin_a0); output_low(pin_b0); output_high(pin_b1);
output_high(pin_b2); k=24;}
                else { output_low(pin_a0); output_high(pin_b0); output_low(pin_b1);
output_low(pin_b2);k=0;}
                delay_ms(2000);
                res=res-k;

                g=res/16;
                if (g == 1) { output_low(pin_a0); output_high(pin_b0); output_low(pin_b1);
output_high(pin_b2); k=48;}
                else { output_low(pin_a0); output_high(pin_b0); output_high(pin_b1);
output_low(pin_b2);k=0;}
                delay_ms(2000);
                res=res-k;
}
```

PIC 2:

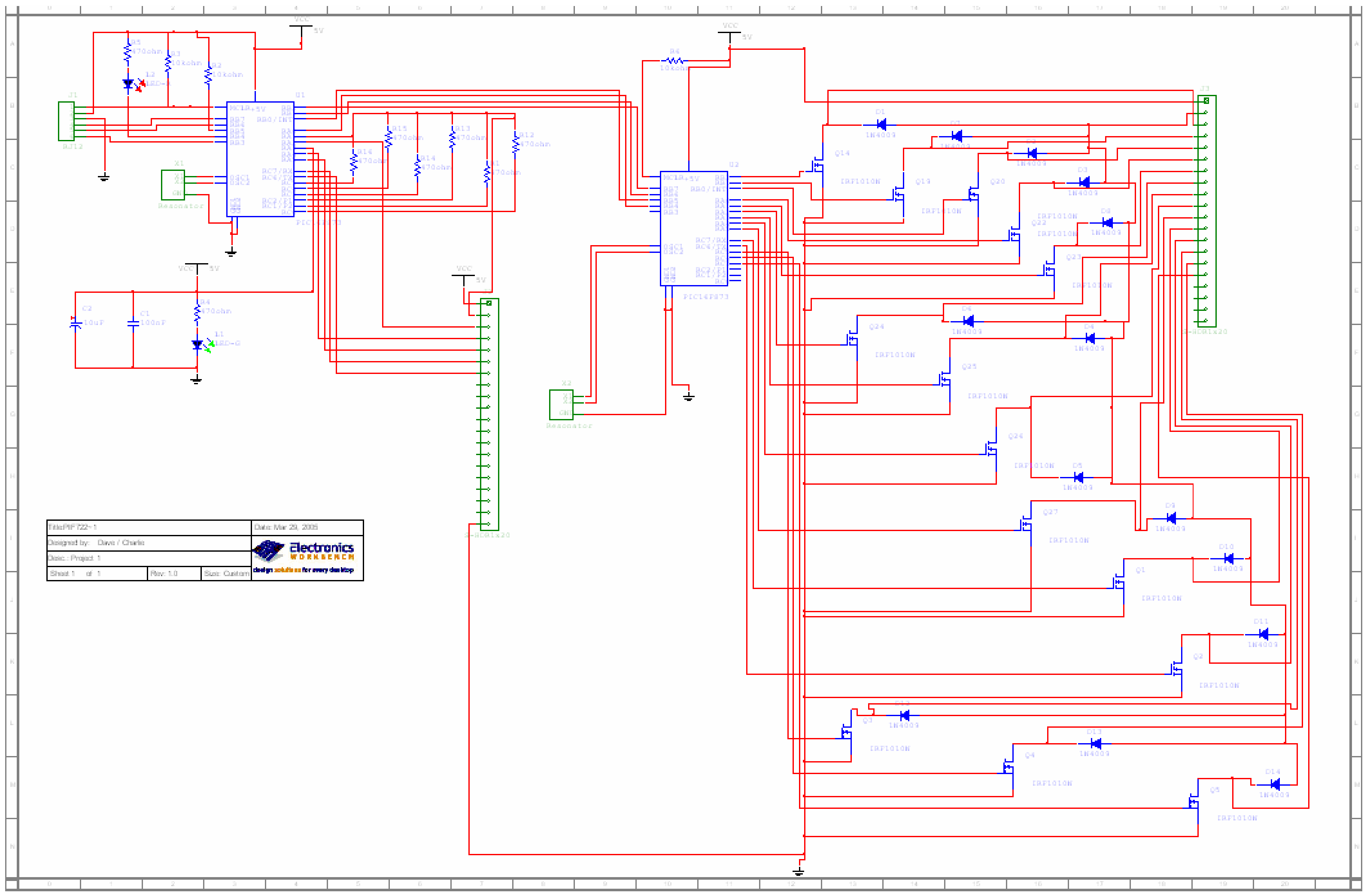
```
#include "PIC_A_D.h"

main() {
  signed int16 m;
          long int charlie;
          int w=0;
          int res;
          int h=0;
          int g=0;
          int k=0;

  //These next 6 lines set certain hardware on the processor.
  //Ignore them for now.
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  //Set up adc port to read from channel 1 (pin AN1)
  setup_adc(ADC_CLOCK_INTERNAL);
  setup_adc_ports(ALL_ANALOG);
  set_adc_channel(1);

  //Blink PIC LED (sanity check to make sure all is working).
          output_low(pin_c4);
          output_low(pin_c5);
  output_low(pin_b4);
  delay_ms(500);
  output_high(pin_b4);
          delay_ms(500);
          output_low(pin_b4);
  delay_ms(500);
  output_high(pin_b4);
          delay_ms(500);


  //Main loop
  while (1) {
          h=input(pin_c0);
          g=input(pin_c1);
          k=input(pin_c2);


          if (input(pin_c0)==0 && input(pin_c1)==0 && input(pin_c2)==0)
          {output_low(pin_b2); output_low(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
          output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
          output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); }

          else if (input(pin_c0)==0 && input(pin_c1)==0 && input(pin_c2)==1)
```

```
        {output_high(pin_b2); output_low(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); output_low(pin_c3);}

        else if (input(pin_c0)==0 && input(pin_c1)==1 && input(pin_c2)==0)
        {output_high(pin_b2); output_low(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); output_low(pin_c3);}

        else if (input(pin_c0)==0 && input(pin_c1)==1 && input(pin_c2)==1)
        {output_low(pin_b2); output_high(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); output_low(pin_c3);}

        else if (input(pin_c0)==1 && input(pin_c1)==0 && input(pin_c2)==0)
        {output_low(pin_b2); output_low(pin_b1); output_high(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); output_low(pin_c3);}

        else if (input(pin_c0)==1 && input(pin_c1)==0 && input(pin_c2)==1)
        {output_low(pin_b2); output_low(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_high(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_high(pin_c4); output_low(pin_c3);}

        else if (input(pin_c0)==1 && input(pin_c1)==1 && input(pin_c2)==0)
        {output_low(pin_b2); output_low(pin_b1); output_low(pin_b0); output_low(pin_a0);
output_low(pin_a1);
        output_low(pin_a2); output_low(pin_a3); output_low(pin_a4); output_low(pin_a5);
output_low(pin_c7);
        output_low(pin_c6); output_low(pin_c5); output_low(pin_c4); output_high(pin_c3);}
        }
}
```

Ultiboard File:



Multisim File:


Title:PIFT22~1
Date: Mar 29, 2005
Designed by: Dave / Charlie
Desc.: Project 1
Sheet 1 of 1
Rev: 1.0
Size: Custom
Electronics WORKBENCH
design solutions for every desktop
VCC
5V
PIC16F873
IRF1010N
1N4009
Resonator
470ohm
10kohm
10uF
100nF