An Overview Of The BOREXino Tether Measurement System

Steve Hardy

August 15, 2006

1 Overview of System

This document will serve as a detailed description of the electronics that provide the data acquisition and read out for the tether measuring hardware to be used in Borexino. The hardware consists of two wheels (one is a pulley, the other is simply a narrow cylinder with a knurled edge), these wheels compress the tether via two springs attached to the shafts about which these wheels rotate (See Figure 1). One of the wheels can move freely on the shaft, the other is directly coupled to its shaft via a roll pin (I may change this to using a keyed shaft), this shaft is also coupled to the shaft of a rotary encoder via two small cotter pins. This encoder, together with the read-out electronics provides a method to determine the precise length of the tether that has passed through the wheels.

2 Rotary Encoder

The rotary encoder chosen for this device is a quadrature encoder, meaning that it uses two infrared optical interrupters, and two discs (alternating transparent and opaque regions) on the same shaft, but offset by 90°. If the decoding circuitry sees a pulse from encoder A before encoder B it knows it is going in one direction, and vice versa. Since this type of encoder is a relative encoder (absolute encoders are much more expensive and output a voltage linearly proportional to the position, however this would create the problem we faced with the previous method at the end points), the encoder has 32 detents on the shaft to ensure that that the data is read when there is no mixture of opaque and translucent regions in the path of the optical interrupter. This particular encoder has a minimum life of 1,000,000 cycles (a cycle being a complete turn through all 32 detents and back again), and a relatively small operating torque (3.5 in-oz when at a detent, 1.5 in-oz when between detents). The unit runs on 5V and has two outputs, one for each of the two optical interrupters which are then read out by the decoder IC.

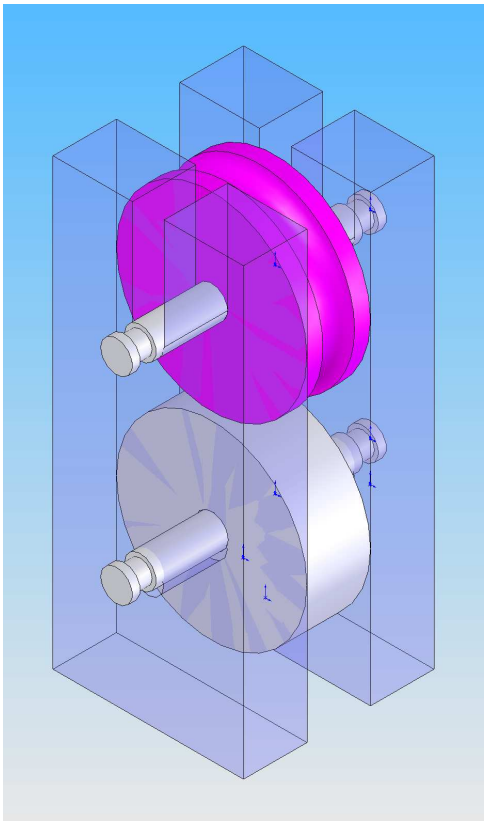


Figure 1: Mount for wheels. Springs are not shown because they are a pain to draw. One of the shafts will be longer than the other to couple to the encoder.

3 Decoder IC

The heart of the system is the Agilent Tecnologies (now Avago Technologies) HCTL-2022 quadrature decoder / counter interface IC. The chip includes filtered inputs, the decoder logic, a 32 bit binary up down counter, a 32 bit data latch (with inhibit) and an octal four bit data buffer(See Figure 2).

Currently, this chip is driven by a TTL clock running at 20 MHz, but can be increased up to 33 MHz if needed. The rotary encoder has 32 detents per revolution, thus each rotation of the 1.5" wheel to which the encoder shaft is connected yields 32 counts. The 32 bit counter allows one to count to $2^{32} - 1$, thus the full range of the counter is 134,217,727 complete rotations of the shaft; this corresponds to the ability to measure a tether 1,606,514,789 cm long (almost 10,000 miles). During normal operation, the output of the counter is continuously fed into the data latch, however, when one wants to obtain the

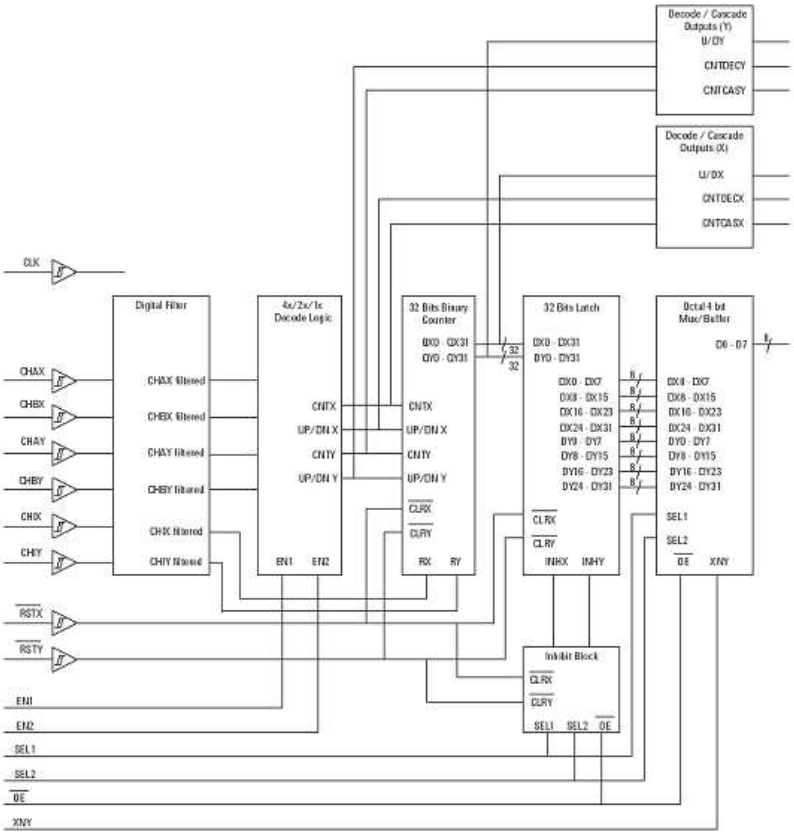


Figure 2: Logic Diagram of HCTL-2022

reading of the counter, an inhibit signal (active low) is fed to the latch which locks the current value into the latch until the inhibit is removed. This is important because the counter and latch have a 32 bit (4 byte) range, but most microcontrollers, and all parallel ports, can only read a byte at a time, so, the inhibit prevents any new data to be loaded into the latch whilst the computer is ready each of the 4 bytes (the counter continues to count during this time, it just doesn't parse its output into the latch). Two byte selection lines also control the 32:8 multiplexer, allowing the read out code to obtain each byte independtly. The chip also has a counter reset (active low) pin which clears the counter and data latch. Table 1 summarizes the states of the inhibit and multiplexer signals during the read out process (note, execution occurs on falling edge of a clock pulse):

Step	SEL1	SEL2	$\overline{OE}$	Action
1	L	H	L	Set inhibit; Read MSB
2	H	H	L	Read 2^{nd} byte
3	L	L	L	Read 3^{rd} byte
4	H	L	L	Read LSB
5	X	X	H	Complete inhibit logic reset

Table 1: Four byte read sequence

4 Display

In order that the operators of the insertion system are able to easily obtain the length of the tether in the detector without having to walk over to the computer every time (it is still unclear whether or not we will even be permitted to have the computer monitor visible during source insertion) a remote LCD display has been added to the system. This display will most likely be mounted in the gas system control box behind an acrylic window, but we could theoretically mount it in a suitable enclosure located anywhere in CR4. The display is an Optrex C51848NFQJ-LG-ACN 16x2 (16 characters per line, two lines) green LCD display. The display module has eight data bits which can be used to send the ascii values for the characters to be displayed on the screen, or in configuration mode, to set various options to configure the display. To control the display, first the Register Select (RS) bit must be set according to what mode you want the display to be in(0 = configuration, 1 = print), then one places a value on the eight data lines corresponding to either a command, or an ascii character value, then the Enable bit must be cycled on and off. It seems that modern day computers are a lot faster than one would think, thus, one must give the display time to commit the changes before sending another data stream (experimentation with the Perl code indicates that a 1 ms delay is optimum)[1]. Table 2 summarizes the command controls used to configure the display.

The display actually has 80 display locations (however, only 32 of these are visible at one time), thus Table 2 provides a means to shift the entire display. Each display location has an address, with the first one on the top row having the hexadecimal address of 0x0, and the first address on the second line with the hexadecimal address of 0x40 (the second location is 0x01 and 0x41 respectively, and it increments by one for each subsequent address). These locations are set via the "Set Display Address" command listed in the table above. The CGRAM address is not used in this application (it allows access to Japanese characters, and also symbols). The module also allows one to read the current values from the display via a Read / Write line (Write = 0, Read = 1), however there is no need for this in the present application, therefore, this bit is kept grounded.

[1]This is most likely due to the fact that the circuitry on the display board is CMOS, which is much slower than 7400 series TTL logic present on the rest of the electronics

Command	D7	D6	D5	D4	D3	D2	D1	D0
Clear Display	0	0	0	0	0	0	0	1
Display & Cursor Home	0	0	0	0	0	0	1	x
Character Entry Mode	0	0	0	0	0	1	I/D	S
Display On/Off & Cursor	0	0	0	0	1	D	U	B
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x
Function Set	0	0	1	8/4	2/1	10/7	x	x
Set CGRAM Address	0	1	A	A	A	A	A	A
Set Display Address	1	A	A	A	A	A	A	A

I/D: 1*=Increment, 0=Decrement	R/L: 1*=Right shift, 0=Left shift
S: 1=Display shift on, 0*=Display shift off	8/4: 1*=8 bit interface, 0=4 bit interface
D: 1=Display on, 0*=Display off	2/1: 1=2 line mode, 0*=1 line mode
U: 1=Cursor underline on, 0*=Underline off	10/7: 1=5x10 dot format, 0*=5x7
B: 1=Cursor blink on, 0*=Cursor blink off	D/C: 1=Display shift, 0*=Cursor move
x = Don't care	* = Intialisation setting

Table 2: Command control codes

5 The Parallel Port

As previously mentioned, all of the communication between the computer and the electronics is via a standard parallel port. Table 3 summarizes the parallel port in this context.

Figure 3 is a pictorial representation of the DB-25 connector.

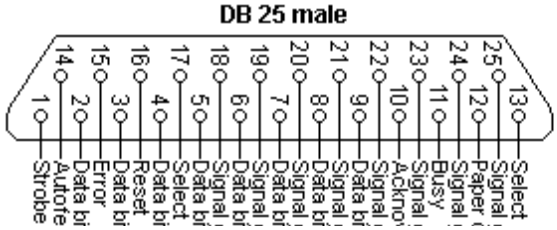


Figure 3: Male DB-25 connector pin out

The address of the Data register on most[2] computers is 0x378, and the Control register is located at 0x37A. One also should notice that the three of the four pins on the control register are hardware inverted (i.e. if the line has a logic 1 signal on it, the computer will read it as logic 0). Table 4 summarizes

[2]You can find the address under the device manager in windows

Pin #	Pin Name	Direction	Register	Destination
1	Strobe†	In/Out	Control	Data Select
2	Data 0	In/Out*	Data	Data 0
3	Data 1	In/Out*	Data	Data 1
4	Data 2	In/Out*	Data	Data 2
5	Data 3	In/Out*	Data	Data 3
6	Data 4	In/Out*	Data	Data 4
7	Data 5	In/Out*	Data	Data 5
8	Data 6	In/Out*	Data	Data 6
9	Data 7	In/Out*	Data	Data 7
14	Linefeed†	In/Out	Control	Enable / $\overline{OE}$
16	Initialize	In/Out	Control	$\overline{RST}$ / SEL1
17	Printer Select†	In/Out	Control	RS / SEL2
18-25	Ground	-	-	-
†=Hardware Inverted		*=Direction specified by bit 5 of Control register		

Table 3: Parallel port characteristics. Pins 10-13, 15 not connected in this application; they go to the status register, which is read-only.

Bit #	Connection	Inverted
0	Strobe	Y
1	Linefeed	Y
2	Initialize	N
3	Printer Select	Y
4	Not used in this application	-
5	Data Lines as Input	N
6 & 7	Unused	-

Table 4: Contents of the Control register

the Control register (the data register is very simple, bit 0 of the register goes to the Data 0 line, and so on):

The fifth bit of the control register is very important; when held at logic 0, the eight data lines (pins 2-9) are output only, conversely when the fifth bith of the control register is held at logic 1, the data pins will input data to the computer.

6 The Read Out Circuit

The design of the circuitry was motivated by the desire to keep everything (display and counter) controlled by one parallel port without the need for any human interaction; this presented a bit of a challenge from an engineering standpoint because the parallel port only has 12 input/output lines, but the display requires 10 and the decoder chip requires 12. Another key consideration was to keep the

number of clock cycles needed to read the decoder, compute the position, and output it to the display to a minimum since the CR4 computer will have to read out a plethora of items, in addition to running the calibration software. Many options were explored that would allow everything to be controlled from one parallel port, however they were all either tremendously ineffecient in terms of clock cycles, or just didn't reduce the number of lines needed to 12 or less.

In the end, I determined that if I could group the lines carefully, I could use one line (the Strobe pin on the parallel port) as a "Data Select" signal, which would operate bilateral switches in a double-pole double-throw configuration. A regular multiplexer wouldn't work because they operate only in one direction, and I required that the data lines be bidirectional. The 74HC4016 IC, a quad bilateral switch, was chosen for this purpose. These chips have four switches, that allow current to flow in either direction, each with it's own active hi enable input which would open, or close, the switch (See Figure 4).

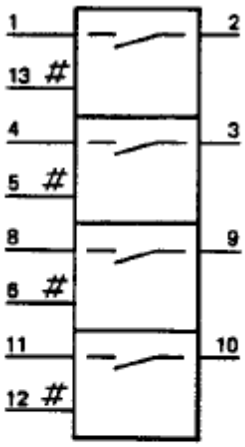


Figure 4: The 4016 bilateral switch

By using two of these switches (one with the data select signal sent to its enable input, and the other with the data select signal inverted[3] as its enable input) you could open one switch and close the other. Therefore, if the parallel port's Data 0 line was connected to the input of each of the two switches, and output of one switch was sent to the Data 0 line of the decoder chip, and the output of the other was sent to the Data 0 input on the display, you could, by changing the value of the data select signal, decide which of the two lines was connected to the parallel port's Data 0 line (See Figure 5).

One has to be somewhat careful with these switches due to the fact that we are using logic signals here, the switch can have one of three outputs[4] depending on its state:

[3] a 7404 hex inverter was used for this purpose

[4] I'm being careless about saying "input" and "output" with regard to these switches. The

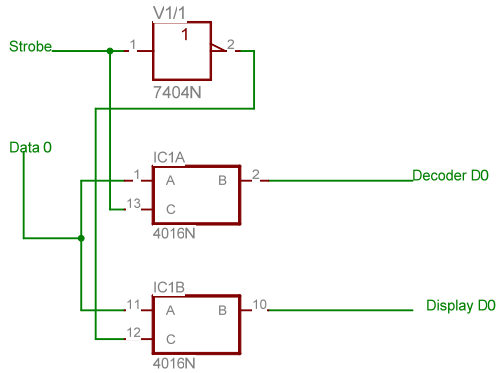


Figure 5: Schematic of the bilateral switches configured as a DPDT switch

- Logic 0 - If the switch is closed and its input is a logic 0 signal
- Logic 1 - If the switch is closed and its input is a logic 1 signal
- High Impedance - If the switch is open. Remember the golden rule of logic signals : "nothing is not nothing", i.e., even a logic 0 signal which is nominally 0V is not the same as an open circuit

Through experimentation with these switches, I noticed that when an output is in the high impedance state, the output floats up and down a lot, this presents a problem for certain signals (namely, the output enable and the counter reset signals) which need to be in a certain state, even when the switch is open (we only want the counter reset signal to be in a 0 state when the program tells it to, not on its own accord). To counteract this, 10KΩ pull-up, or pull-down, resistors (to 5V and 0V respectively) were placed across these terminals in accordance with which signal it was to maintain.

The data lines from the printer port were split, via the bilateral switches, with one set of eight going to the data outputs for the decoder chip, and the other set to the inputs for the LCD display. These switches, as well as all of the others are controlled via the Strobe signal, with a logic 0 routing the lines to the display, and a logic 1 routing them to the decoder IC. The Linefeed signal (Pin 14 on the parallel port) splits via one of the switches and goes to either the LCD's enable input (Strobe = logic 0), or, the decoder IC's output enable input (Strobe = Logic 1). The Init signal (pin 16) splits to either the decoder's reset input (Strobe = logic 0), or, the decoder's SEL1 input (Strobe = logic 1). Finally, the Printer Select line (Pin 17) splits into either the LCD's Register Select input (Strobe = logic 0), or the decoder's SEL2 input (Strobe = Logic

switches have two lines that are either inputs or outputs, it just depends on which direction the current is flowing, but you get the idea

1). So, you can see that by setting the strobe signal to logic 0, we gain access to all of the lines controlling the LCD display (and also the counter reset signal as well, but remember this is an active low signal on a non-inverted line, so, we just make sure that pin 16 is set hi before setting the strobe to low). By the same logic, setting the strobe to logic 1 allows access to all of the lines controlling the decoder IC. This is most likely better explained by Table 5.

Line(s)	Strobe=0 Connection	Strobe=1 Connection
D0-D7	Display D0-D7	Decoder D0-D7
Line Select	Display Enable	Decoder $\overline{OE}$
Initialize	Decoder $\overline{RST}$	Decoder SEL1
Printer Select	Display Register Select	Decoder SEL2

Table 5: Data routing table

However, if you recall from the table above, in order to read what is on the data lines, bit five of the control register must be set HI, otherwise you will not read the true value.[5]

One look at the attached schematic reveals that this current scheme of using the strobe signal to route the signals requires many copies of the strobe signal to be sent to the bilateral switches. This presents a problem because the parallel port has a limited output current, and the gates that it feeds have a minimum operating current. Thus, a TTL IC has a fanout capability defined by $F = \lfloor \frac{I_{in}}{I_{out}} \rfloor$, where $\lfloor \, \rfloor$ is the floor function. The value of F for typical ICs is around 4, so, to ensure that everything triggers, a 74HC125 quad line driver IC was used. The strobe signal coming from the parallel port was fed to one segment of the line driver, and each of four copies were sent to their own line driver, some of which were inverted first. This is illustrated better in the schematic at the end of this document.

In the final product, the distance between the rotary encoder and the rest of the circuit (and also the display and the circuit) will be on the order of 10 meters. At this distance, TTL signals are prone to attenuation and noise pickup. A four fold solution to this problem is proposed (but not yet implemented):

- Firstly, a line driver will be used on the output for each signal in order to boost the signals back to the proper strength.
- Secondly, a line receiver[6]will be used at the receiving end of the line.
- Thirdly, shielded twisted pair cable will be used for all signal transmission. Since the conduit containing the cable will likely travel along the ceiling of CR4, it will be near both fluorescent light tubes and the large motors

[5]When I speak about the strobe line having a certain signal, I am referring to the actual strobe line – not the value that one writes to the port (recall this line is hardware inverted).

[6]Avago HCPL-2602 line receiver. It is actually an optoisolator, but has a line terminator built into the IC, so, one does not have to worry about reflections causing further attenuation of the next signal. It also has a very high CMR, which is crucial in this application

on the clean room blowers, both of which produce a lot of noise. Using a twisted pair cable, with a large twist rate will reduce the loop area between the wires (which is directly related to the strength of magnetic coupling to the signal). At long distances, the surface area of the shielding becomes larger, thus there is a greater capacitance between the wires, and cross talk can occur in parallel cabling.

- Finally, the shielding of the twisted pair cable will be grounded at *one end only*. This is done because the source of interference will be different at each end of the cable and grounding both ends of the cable would cause a current to flow.

In an effort to protect the computer's parallel port from anything detrimental that might happen, an isolation technique using optocouplers will be employed. Since optocouplers only work in one direction, the couplers will be placed before the bilateral switches so that the signals are already isolated when they get to the switch. I plan to use several quad logic optocouplers[7] for this purpose. These ICs are bi-directional, which does not mean what one would think in that they are referring to the fact that some of the N isolators in the IC transmit in one direction, and the others transmit in the other (i.e. one isolator segment will not function in both directions).

7 User Interface

The user interface for the entire system is a script written in Perl with a Tk frontend. The program takes care of initializing the display and decoder, and provides a means for the user to start polling the status of the counter every second, and also to stop it and reset the counter. The script automatically updates the display if the value of the counter has changed. In order to communicate with the parallel port, the program uses the inpout32.dll library file[8] along with the Win32::API extension[9]. In order to instantiate the read / write routines the following must be placed in the top of the code:
$GetPortVal = new Win32::API("inpout32", "Inp32", [I],I);
$SetPortVal = new Win32::API("inpout32", "Out32", [I,I], I);
Then, to communicate with the parallel port the following commands are used:[10]
$ReadVal = $GetPortVal→Call(ADDRESS);
$SetPortVal→Call(ADDRESS, DATA);

The rest of the code is self explanatory for the most part, so, I will not go into further detail here. However, Figure 6 will probably prove useful when trying to decipher the code.

[7] Avago HCPL-901J
[8] This file *must* be placed in the same directory as the perl script itself
[9] It also requires Time::HiRes
[10] Recall that ADDRESS is either 0x378→Data register, or, 0x37A→Control register

Register	CONTROL (0x37A)								Hex	DATA (0x378)								Hex
Line Name	nStrobe	nLine	Init	nSelect	IRQ	Bi-Direct.	x	x	Value	D0	D1	D2	D3	D4	D5	D6	D7	Value
Register Bit	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	
Display ON	1	1	1	1	x	0	x	x	0xF	1	1	0	0	0	0	0	0	0x3
Enable ON	1	0	1	1	x	0	x	x	0xD	x	x	x	x	x	x	x	x	
Enable OFF	1	1	1	1	x	0	x	x	0xF	x	x	x	x	x	x	x	x	
Print Mode	1	1	1	0	x	0	x	x	0x7	ASCII Character Value								
Clear Display	1	1	1	1	x	0	x	x	0xF	1	0	0	0	0	0	0	0	0x1
2 Line Mode	1	1	1	1	x	0	x	x	0xF	1	1	0	1	1	1	0	0	0x3B
Goto Line 2	1	1	1	1	x	0	x	x	0xF	0	0	0	0	0	0	1	1	0xC0
Goto Line 1	1	1	1	1	x	0	x	x	0xF	0	0	0	0	0	0	0	1	0x80
Reset	1	1	0	1	x	x	x	x	0xB	x	x	x	x	x	x	x	x	
Read MSB	0	1	0	0	x	1	x	x	0x22	MSB Data On Lines								
Read 2nd Byte	0	1	1	0	x	1	x	x	0x26	2nd Byte On Lines								
Read 3rd Byte	0	1	0	1	x	1	x	x	0x2A	3rd Byte On Lines								
Read LSB	0	1	1	1	x	1	x	x	0x2E	LSB On Lines								
Inhibit Reset	0	0	0	0	x	1	x	x	0x20	x	x	x	x	x	x	x	x	

Figure 6: List of Commands

8 Summary

Ultimately, I feel that this system will provide a mechanically simple way of determining the length of the tether to the point where the largest uncertainty in position comes from the amount of slack tether after the encoder hardware itself. The system is extremely inexpensive to construct (the LCD display being the most expensive item at just over $30), and is also small enough to not take up too much valuable real estate in the glove box. Most importantly, the system is very clean; the only objects that come into contact with the tether are the two wheels, and possibly a grommet at the entrance and exit of the box inside which the hardware will be mounted. The box can be opened in CR4, and the tether installed, and then the entire assembly placed into the glovebox with the drum and auger system, and mounting the box inside the glove box can be performed using thumb screws, or something equally as manageable through the thick gloves.

The following pages contain the master schematic (minus the line drivers and optoisolators), and a few photographs of the prototype circuit.

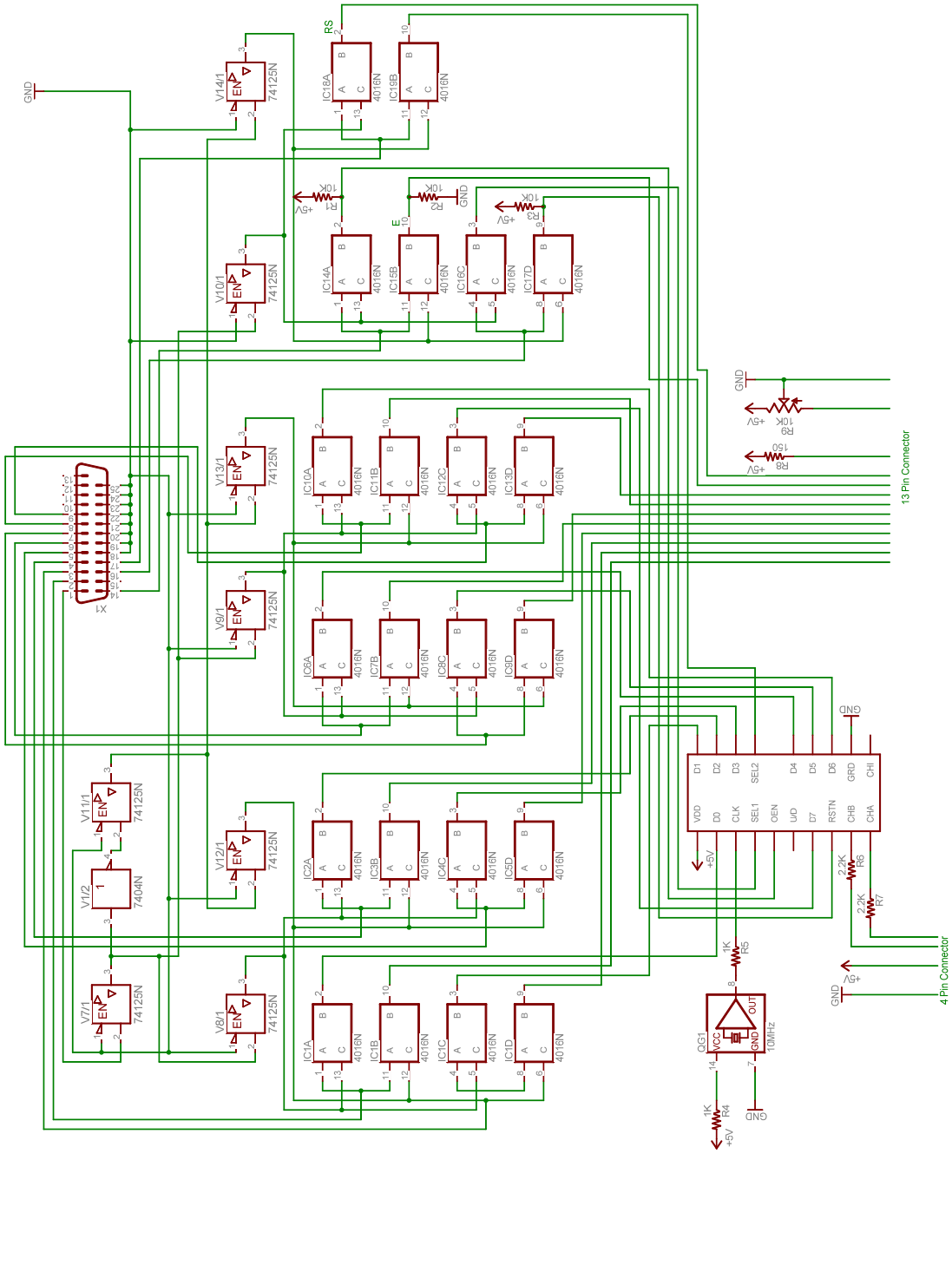
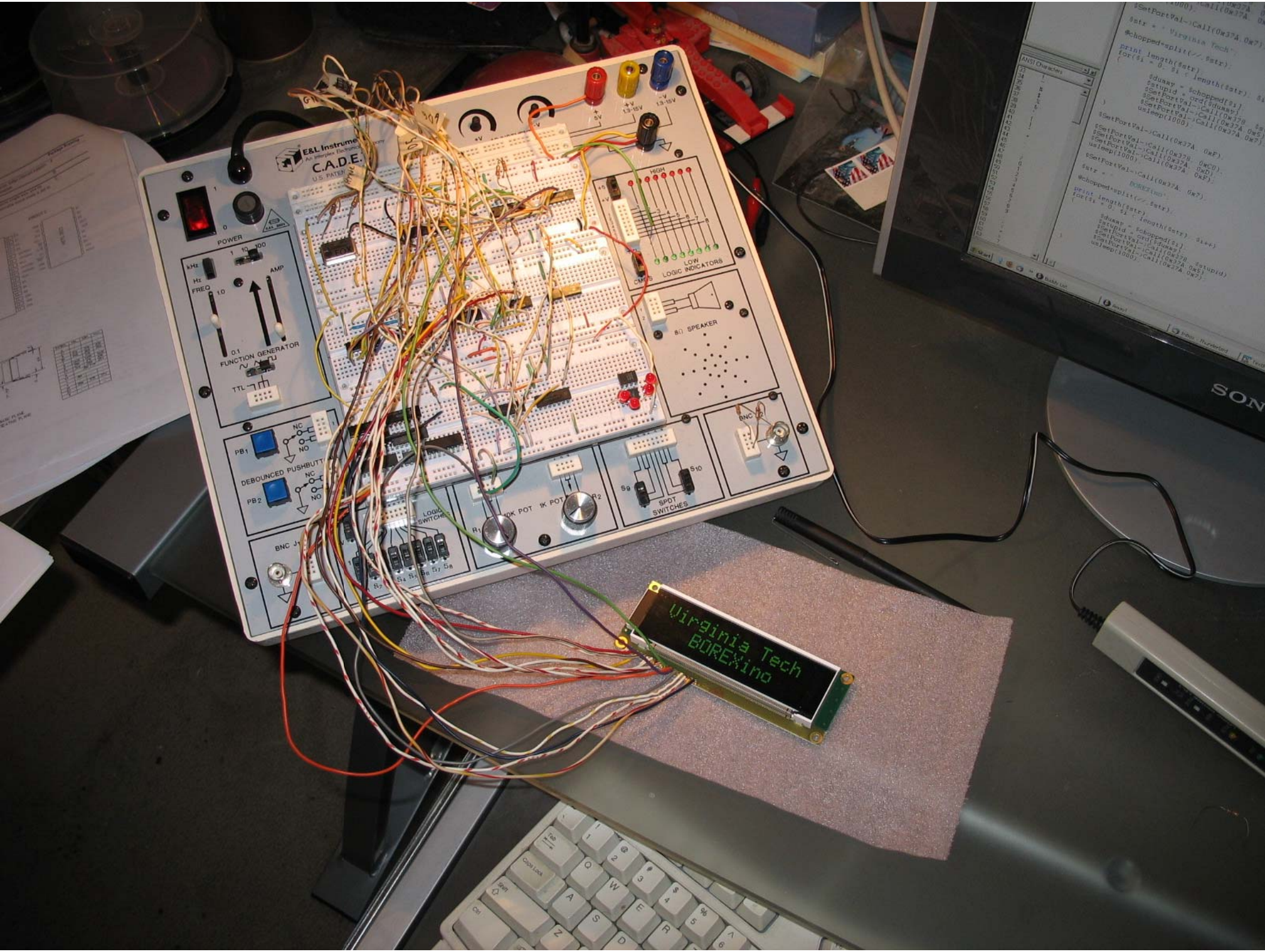
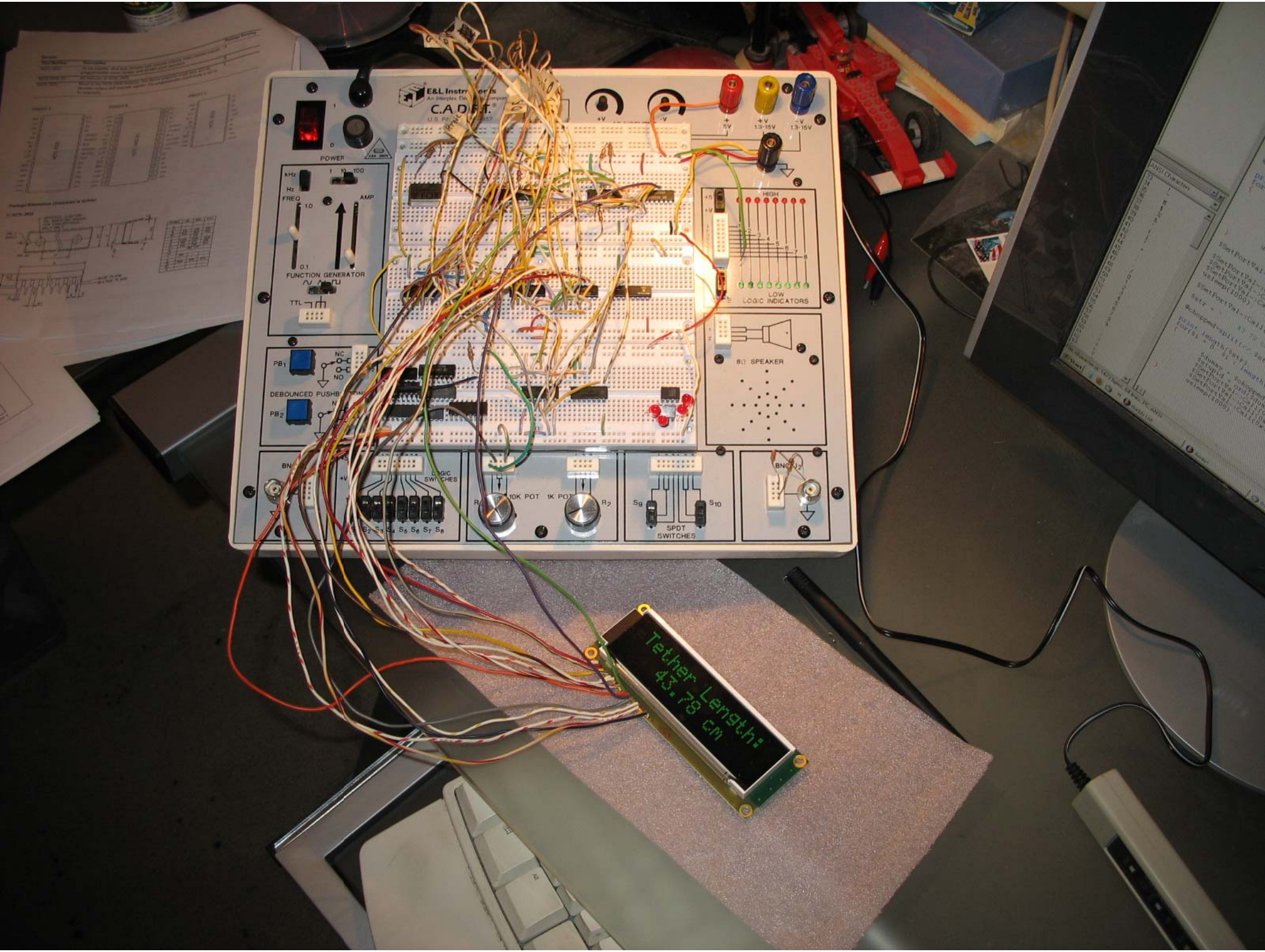


Figure 7: Schematic of electronics


C.A.D.E.
POWER
FUNCTION GENERATOR
LOGIC INDICATORS
SPDT SWITCHES
LOGIC SWITCHES
DEBOUNCED PUSHBUTTONS
Virginia Tech
BOREMino


E&L Instruments
C.A.D.E.T.
POWER
FUNCTION GENERATOR
FREQ
AMP
TTL
HIGH
LOW
LOGIC INDICATORS
8Ω SPEAKER
DEBOUNCED PUSHBUTTONS
LOGIC SWITCHES
10K POT
1K POT
SPDT SWITCHES
Tether Length:
43.78 cm