Breadboard, IC, & Circuit Debugging Kit

Date: 2013 Dec 13

Team Member Names and Signatures:

Michael Kwan
mkwan4@uic.edu

Kaushal Patel
kpate38@uic.edu

Scott Wu
swu31@uic.edu

Kai Zhao
zhao68@uic.edu

Faculty Advisor:

Wenjing Rao
wenjing@ece.uic.edu

Table of Contents

II. Project Goals

The project goal is to design and build a logic circuit debugging tool kit includes a breadboard tester, an integrated circuit (IC) tester, an IC detector, and a circuit detector. The breadboard tester shows the independent set of each pin to detect for defects by checking if every column of the breadboard is in their own independent set. The IC tester takes an IC in the zero insertion force (ZIF) socket and its model number to detect which part of the IC is not working properly. If the IC model number is unknown, then the IC detector can be used to detect if the functionality of the IC in the ZIF socket matches any of the IC in the data library. The circuit detector takes a circuit, the number of inputs and outputs, and connections to the inputs and outputs the location of the inputs and outputs to display the function of each output in terms of the inputs in Sum of Products (SoP) and Product of Sums (PoS) form.

III. Product Need

Dilemmas between starting over or giving up occur when building circuits with ICs on breadboards does not work properly. The situation is more aggravating when the circuit does not work because of faulty ICs or defective breadboards. Our circuit debugging kit can solve both these problems and lessen the frustration already caused by building circuit on a breadboard. The user will be able to find out if either the breadboard or the integrated circuit is defective. If neither is defective, the user will know to check for other problems associated with breadboard circuits such faulty wires, faulty wiring, or wrong logic design. For educational purposes, instructors have to check if students built the circuit correctly. Asking students to demonstrate a few inputs does that verify that the circuit will work for all inputs. On the other hand, asking students to demonstrate every possible input will be time consuming to manually flip the switches and read the outputs. Therefore, a circuit detector, which shows the function of the outputs in terms of the inputs, can be used to help students and instructors verified the built circuit.

The current method to test for a faulty breadboard is to use a multimeter to conduct connectivity tests between one pinhole, four other pinholes in the same column, one pinhole in adjacent column on the left, and one pinhole in the adjacent column on the right. This method is very tedious and time consuming because each column requires at least six pin comparisons pin holes in the column output the original input (Figure 1). Our debugging kit conducts connectivity tests for thirty pinholes instantaneously.

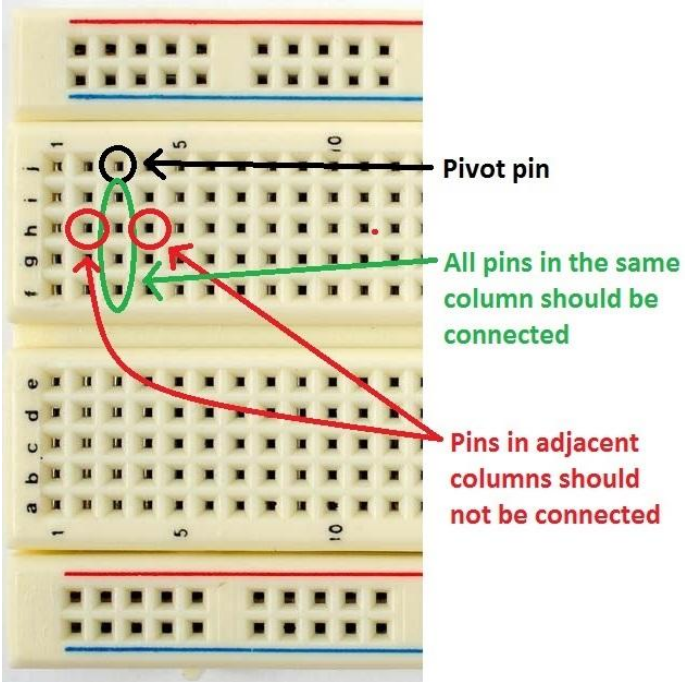


Figure 1: Breadboard showing the current methods of testing a breadboard. Testing every column in a functional breadboard should show a positive continuity test between the pivot pinhole and four other pinholes in the same column and a negative continuity test between the pivot pinhole and any pinholes in the adjacent columns.

One of the current methods to test an IC is to buy an IC tester such as the LEAPER-1A for $320, the Instek GUT 6000A for $1,060, and the BK Precision 570A for $1,495 (Figure 2). Although most of the digital IC testers currently in the market has a large data library, they are very expensive and impractical for educational purposes. An alternative method to test an IC is to build circuits with the IC and check if the integrated circuit is giving the correct output for every possible combination of inputs. However, the IC diagnosed incorrectly if the circuit to test the IC was built with faulty wires, faulty wiring, or wrong logic. At a small fraction of the cost of an IC tester, our debugging kit is capable of testing and detecting an IC by simply placing it in the ZIF socket and running the program.

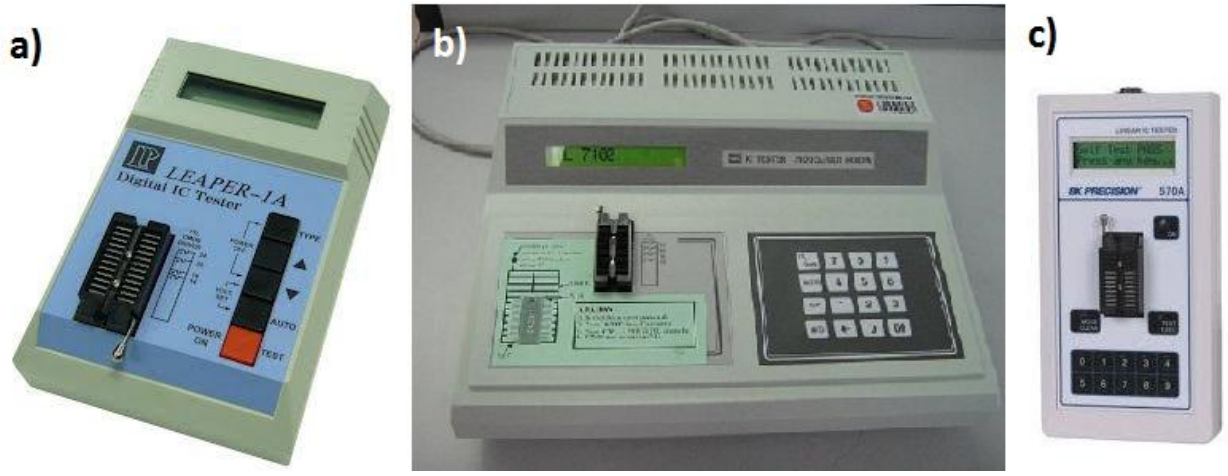


Figure 2: IC testers. a) LEAPER-1A, b) Instek GUT 6000A, and c) BK Precision 570A

There is currently no methods for testing or detecting logic circuits on breadboards. The circuit detector in our debugging kit can be connected to inputs and outputs of the circuit to help students and instructors to quickly verify the circuit.

IV. Technical Specifications

Breadboard Tester

The breadboard tester is a 30 pin (5 rows and 6 columns) circuit that connects to the microprocessor and the breadboard to test. The breadboard tester can easily be plugged to and unplugged from the breadboard. Then, the microcontroller sends a signal (+3.3 V) to a pivot pin. Every nearby pin should attempt to read the signal to check for a connection between itself and the pivot pin. If the signal is not detected in the same column or if the signal is detected in an adjacent column, then display the location of the error on the LCD screen.

IC Tester

An IC can be placed onto the ZIF socket and the IC model number can be entered into the microcontroller (e.g. “74LS08” or “AND” via touchscreen or keyboard). If the circuit under test behaves like a 74LS08 for every set of inputs and outputs, then the IC should be working properly. Otherwise, our device will notify the user whether the entire IC is faulty and which gates are not working properly (Figure 3).

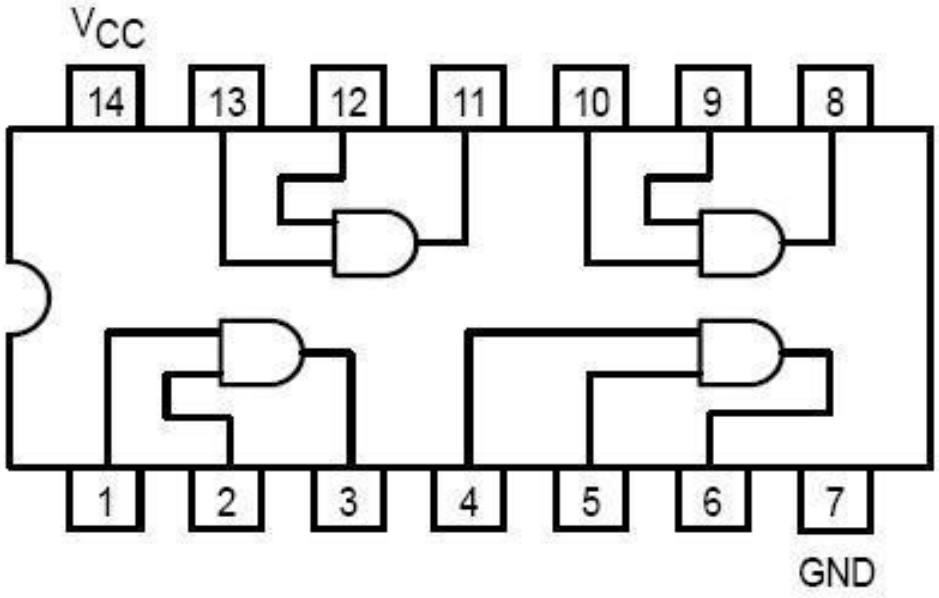


Figure 3: 74LS08 (Quad AND), showing that the IC contains multiple gates. Therefore, it is possible for some of the gates in the IC to be faulty.

IC Detector

If the IC model number is not available, then the user can choose the “detect” option. The microcontroller can go through all ICs in the data library and attempt to find an IC model that behaves exactly like the inserted IC. If the device finds a match, then it will display the model number to the user. Otherwise, either the IC is faulty or the IC model is not in the data library.

Circuit Detector

The circuit detector supports circuits up to 8 inputs and 8 outputs. Similar to the breadboard tester, the circuit detector will have pins coming out of the microprocessor to connect to the input and output pins of the circuit. The user have to specify the number of inputs and outputs before the detector attempts to record all the outputs for every possible input and generate the truth table. The circuit detector will use all the minterms of each output to compute the function in simplified canonical SoP and PoS form. Users can then compare the simplified canonical functions to the intended circuit function.

Manufacturer Acceptance Testing

Touchscreen LCD Display

The LCD must have no or minimal dead pixels and must be visible in normal, indoor lighting. The LCD display should display the correct menus and available user options upon startup. The touch screen input should be working correctly on every part of the display. All inputs and outputs must be displayed visibly.

Header Pin Cluster

The pin cluster should not have any bent or loose pins. The pins must be correctly spaced to fit into breadboards. The columns of header pins should be simple to attach and detach from the breadboard columns and rows. Each pin should be number from 0-29 to help users easily determine which side to plug it in.

ZIF Socket

The ZIF socket must have the same pin spacing as a common breadboard for IC insertion. All pin holes must be tested and working. ICs should fit snugly and be easily removed when testing is finished.

Wires

There will be a great deal of wires that connects the microcontroller to the header pins. The wires should be flexible, durable, and well protected.

V. Engineering Design Alternatives

Design Alternative One: Single Pins (not used)

This single pin design is similar to just inserting a wire into a breadboard. The user inserts *n* pins into *n* pinholes of the breadboard and our device will display the independent set of each of the n pins. This design allows for the most flexibility and ease of use. Also, using separate pins will make bent pins very easy to replace. Each pin will be numbered so the user knows where each pin is inserted. We have chosen the number of pins to be 30.

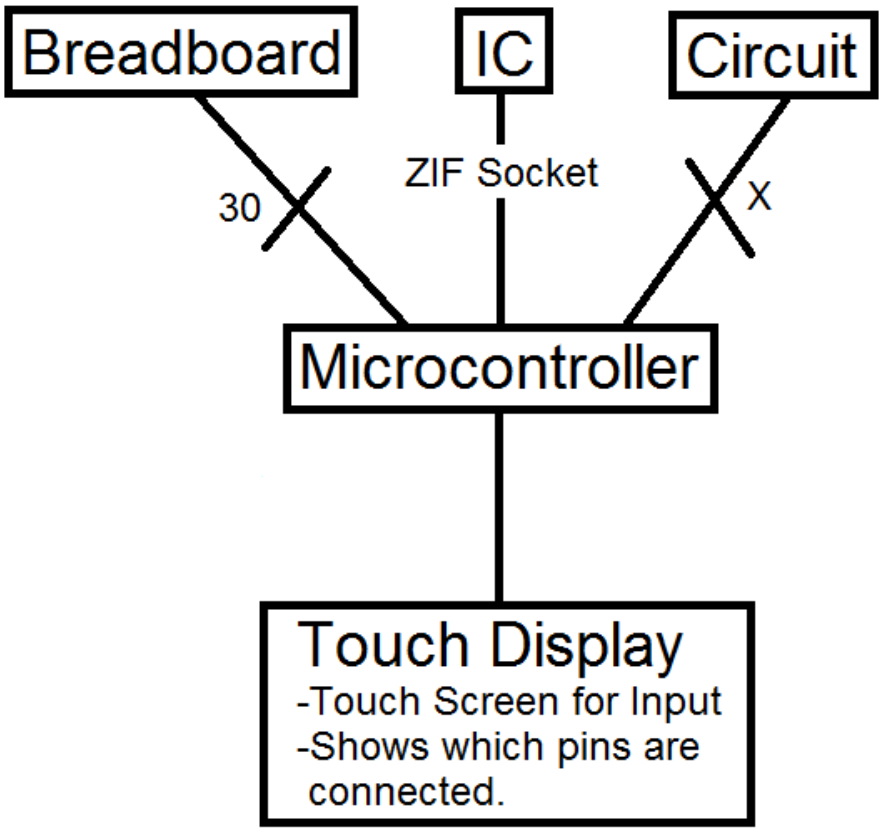


Figure 4: 5x6 Pin Cluster with Touchscreen LCD

Design Alternative Two: Computer Display (semi-used)

The device will just consists of the pins for testing ICs and the breadboard and a microcontroller to select which pins to test. Our device will be powered via USB. All the information our device gets from testing either a portion of a breadboard or an IC will be sent to the computer and the computer will be used as a display as well as the processing component. The user will use the computer to decide to test portions of the breadboard or whether or not an IC is functioning correctly. The design will ultimately save us and the consumer money because we would not need to purchase a LCD display nor a microprocessor. The downside is we would have to figure out how to transport data between our debugging kit and the user's existing computer.

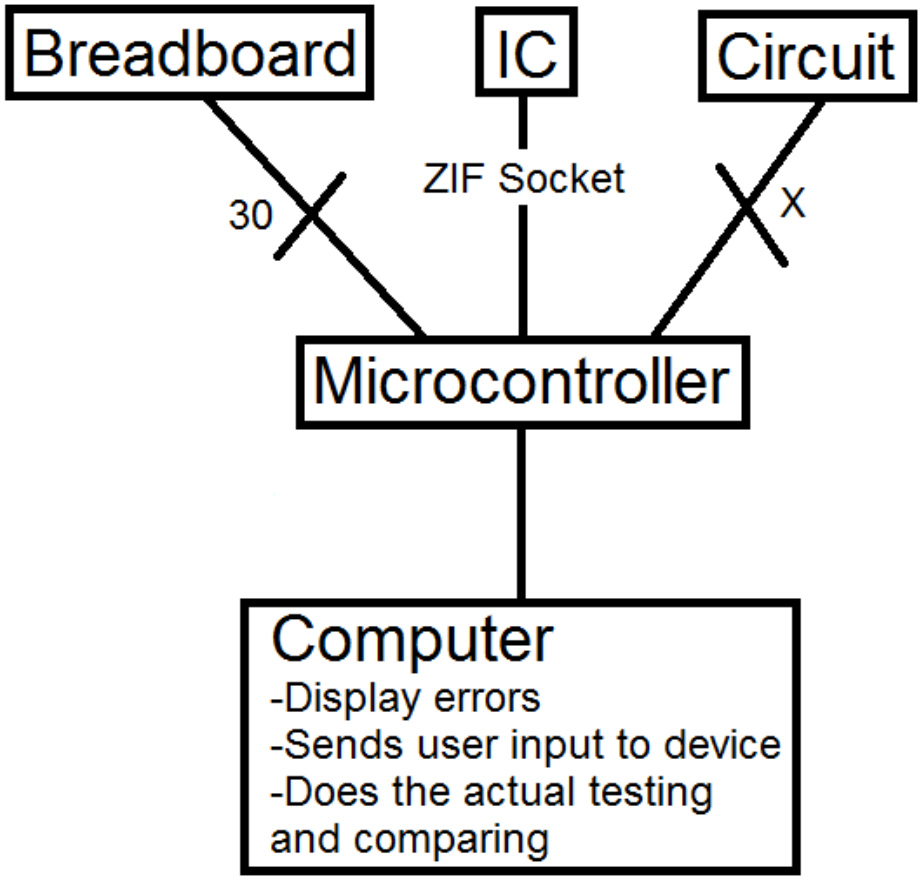


Figure 5: 5x6 Pin Cluster with Computer Display

Design Alternative Three: breadboard with built-in tester (optionally semi-used)

The device will be almost the same as our original design except the debugging kit will be built into the breadboard for users to build and test circuit on one device. The microprocessor will be connected to the debugging kit, which is built over the breadboard (Figure 6). The LCD display will be connected to the microprocessor and placed at the top of the debugging kit portion of the breadboard. If there is something wrong with the breadboard, the tester will detect it and show it on the LCD display. The breadboard portion will be detachable and can be replaced if there are problems with it.

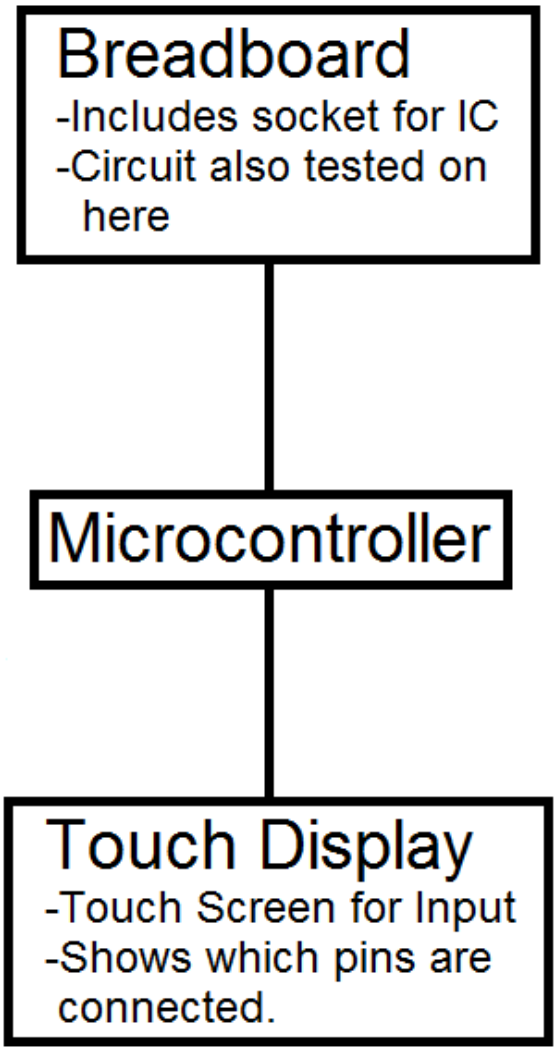


Figure 6: Breadboard with Built-in Tester and Touchscreen LCD

Design Alternative Evaluation Criteria

Our chosen design alternative evaluation criteria are the following:

01. Cost to Consumer
02. Development Cost
03. Technical Knowledge on Design
04. Time Needed to Complete Design and Development
05. Potential Hazards in Developing and Using the Product
06. Compliance with Regulations or Standards
07. Easy of Satisfying Technical Specifications
08. Ability to Manufacture
09. Easy of Product Use
10. Lifespan
11. Portability

Selection of Design Alternative

		Default		Design Alternative 1		Design Alternative 2		Design Alternative 3	
Criteria	Weight	Rank	Score	Rank	Score	Rank	Score	Rank	Score
Cost To Consumer	9	2	18	4	36	1	9	3	27
Developme nt Cost	8	4	32	2	16	1	8	3	24
Technical knowledge on design	10	3	30	2	20	1	10	4	40
Time needed to complete design and developmen t	7	4	28	2	14	1	7	3	21
Potential hazards in developing and using product	7	3	21	4	28	1	7	2	14

Compliance with regulations or standards	8	2	16	4	32	1	8	3	24
Ease of satisfying the technical specifications	6	4	24	2	12	1	6	3	18
Ability to manufacture	5	3	15	4	20	1	5	2	10
Ease of product use	9	2	18	4	36	1	9	3	27
Lifespan	8	1	8	4	32	3	24	2	16
Portability	5	1	5	4	20	3	15	2	10
Total Score			215		266		108		231

Each criterion was given a weight between 1 and 10 (1 being not important and 10 being very important) and each design was given a rank between 1 and 4 (1 being the worst and 4 being the best). The score for each criterion was calculated by multiplying a design's score by the weight of the criterion. The design's total score is computed by adding every row in the score column. The design with the highest score is the best design whereas the design with the lowest score is the worst design.

Technical Knowledge on Design

This was chosen as the most important criterion because if we lacked the knowledge, we would not be able to implement the many functions of our product. This is the main reason we do not want to focus on creating design number two (computer display) because we are unsure of how to import and export data to and from our device and a computer.

Ease of Use

We felt the simpler it is to use our product, more consumers will be attracted to use it. We wanted the minimal amount of ambiguity with pin insertion and user input. We will attempt to design the testing pins in a way where the majority of ECE students would have no problem using our product with just simple menu instructions displayed on the LCD.

Cost to Consumer

Our product was designed with students in mind. We want to keep the price low enough that at least universities would consider purchasing a few to have in the labs on campus. This would allow easy access for students. At the same time, we are trying our best to keep cost low so individual students can afford to purchase one if they choose to.

Based on our evaluation criteria, the best design alternative based on our chosen criteria is the design that will use a computer as a display and the processing unit (design two)*. Unfortunately, we are not confident we can implement this idea. Nonetheless, we will attempt this idea depending on the time. We will first attempt our original design (design four). The breadboard tester and IC tester portion is required in both designs, so we will focus on completing the design of that portion first. We program a microcontroller and use a LCD touch screen for display and user input. We will attempt to use a computer as the display and processing unit only if we can get our original design working first.

Design and Production Cost Analysis

Resource Requirements

Quantity	Part	Part Description	Estimated Cost
10	Header Pins	5 Pin Connector	$3.67 for 10
10	Header Pin Housing	1x6 Clippable Housing	$0.67 for 10
5	SIPO's	Serial In, Parallel Out	$1.95 for 5
5	PISO's	Parallel In, Serial Out	$1.75 for 5
1	28 Pin Socket	ZIF Socket, 28 Pin, 0.3"	$2.95
1	LCD Screen	Serial TFT LCD - 3.2" with Touchscreen	$85.00
1	Microcontroller	Header Board for LPC2294	$49.99
		Total	$146.97

Cost Estimate

The primary cost of our resources will be the LCD screen and the microcontroller.The price for this project is estimated to be around $147 after shipping and handling. Other expenses will occur if we choose to enclose our device in housing for a better user interface.

Task Allocation and Schedule

Date	Month/Week	Phase of Project	Task to be Completed
May-June	Summer Part 1	Ordering Parts	Make final decision on parts and ordering/receiving them.
July-August	Summer Part 2	Proof of Concept Output/Input Control	Modify any flawed designs. Finalize the design of the logic of our device.
08/26/2013	Week 1	Building a 5x6 pins device and two 1x15 pins device for IC and Digital Logic Circuit testing	Test the sturdiness of the soldered header pins. Also check for loose connections and connections that exists when it should not. Also test how easy/difficult it is to send and recieve signals to each point.
09/02/2013	Week 2	Programming the Microcontroller for Pin Connectivity	Work on chaining the SIPOs together so that we can send more than just 8 bits out using only 1 clock and data signal. Also work on figuring out how to actually store the data so that it will show up in the SIPOs as expected.
09/09/2013	Week 3	Programming the Microcontroller for Pin Connectivity	Work on actually being able to send and read each point of the device that connects to the breadboard.
09/16/2013	Week 4	Programming the Microcontroller for Pin Connectivity	Work on sending a current to only 1 point at a time and reading the signals of neighboring points.
09/23/2013	Week 5	Programming the Microcontroller for IC and Digital Logic Circuit Verification	Work on being able to store IC and Logic functions into memory. Then work on sending out the data control the devices.
09/30/2013	Week 6	Programming the Microcontroller for IC and Digital Logic Circuit Verification	Work on reading every output. Work on matching the output with the expected outputs.
10/07/2013	Week 7	Programming the Microcontroller for IC and Digital Logic Circuit Verification	If the outputs does not match the expected outputs and if it is an regular logic IC, then work on attempting to identify which gates do

			and do not work.
10/14/2013	Week 8	Programming the Microcontroller for Output on to LCD Display and Keypad Input	Learn the commands of the LCD. Attempt to show the bitmaps for some inputs.
10/21/2013	Week 9	Programming the Microcontroller for Output on to LCD Display and Keypad Input	Continue to attempt to identify which logic gates do and do not work. Then attempt to show that information on the LCD display.
10/28/2013	Week 10	PCB Mounting and Programming the Microcontroller for IC Detection (if time permits)	Create the design in AutoCAD for drill files, silkscreen, copper traces, etc. Putting together the finalized circuit and attempt further testing.
11/04/2013	Week 11	PCB Mounting and Programming the Microcontroller for IC Detection (if time permits)	Create the design in AutoCAD for drill files, silkscreen, copper traces, etc. Putting together the finalized circuit and attempt further testing.
11/11/2013	Week 12	Debugging/Testing	Testing and debugging the Breadboard tester until we obtain our desired results.
11/18/2013	Week 13	Debugging/Testing	Testing and debugging the IC/Circuit tester until we obtain our desired results.
11/25/2013	Week 14	Debugging/Testing	Testing and debugging the Keypad Input and LCD Output until we obtain our desired results.
12/02/2013	Week 15	Final Testing and Results	Complete Final Testing
12/08/2013	Week 16	Final Testing and Results	Complete Report

Simulation/Modeling Results

See Appendix A, the product user's manual.

Description of the User Interface

See Appendix C for software flowchart.

User Interface

1. The device should be plugged into a breadboard. Then the microcontroller sends a signal (+5 V) to a pin. Every other pin should attempt to read the signal to check for continuity between itself and the pivot pin. If pins in the same column as the pivot pin do not detect the signal, then let the user know that an error has occurred with the column. If pins on adjacent columns do detect the signal, then let the user know that an error has occurred with the adjacent column.

2. An IC can be placed onto the breadboard and the IC model number (i.e. LM741) will be entered using the provided keypad. The user will then attach two rows of header pins to the breadboard; one row will be placed above the IC and one will be place below to cover all the inputs and outputs. The user will then initiate the test from the keypad. If the circuit under test behaves like LM741 for every possible input and output, then the circuit passes the test. Otherwise, the user will be notified that the IC is faulty.

3. There will be an IC scan option. This method will most likely add more time to the process. The IC tester will go through the entire list of ICs and check if the IC matches any of the pre-existing ICs in the list. If an IC is detected, then the output will show the label that the IC is suppose to be. If no IC is detected, then that IC is either faulty or not included in our library.

Design Changes Made During ECE 397

New Design

The first change we made to our design was deciding to use the Raspberry Pi instead of the Header Board which was retired. The Raspberry Pi was also a fraction of the price and was much more practical. It was a linux-based processor which we needed for displaying more than just characters. Moreover, it had more GPIO ports which we thought we would need. After deciding on using the Pi, we also needed a keyboard, mouse, and a wireless adapter.

The next change we made in our design took place after our attempt to implement the breadboard tester using shift registers. We realized that there were too many logic disturbances when using the shift registers which prevented our design from working properly. To solve this problem, we abandoned the shift registers and used the I2C to 16-bit I/O expanders. The switch to the I/O expanders made the design simpler but required us to code in Python.

The last design change we made was the circuit tester. Instead of having the user inputting a SoP/PoS function, we asked for just the number of input and outputs being tested.

The program now acts like a circuit detector; it uses different combinations of the given number of inputs to detect the output values.

Resource Requirement

Quantity	Part	Part Description	Estimated Cost
36	Crimp Wires	Stranded 26AWG wires with male terminals on both ends	$12.60
16	5 x 1 Crimp Wire Housing	5-pin housing for the crimp wires	$1.38
1	Raspberry Pi	Linux based, portable computer, with I2C, SPI, and GPIO ports.	$35.00
4	MCP23017	I2C to 16-bit IO Expander	$4.80
1	28-pin ZIF Socket	Makes for easy connecting or programming to DIP ICs	$2.95
1	26-pin Ribbon Cable	Connects the RaspberryPi IOs to the rest of the circuitry	$2.95
1	USB Wireless Adapter	To access the RaspberryPi using a VNC server	$5
1	LCD Touchscreen	To make the system stand-alone	$17
8	10K 9-SIP Resistors	Pull down resistors	$2.12
1	4GB SD Card	OS and file storage	$5
		Total	$88.80

Cost Estimate

The estimated cost is higher than expected due to the components being purchased at retail prices. The price could very well be lowered to $65 by reducing the price of all components excluding the RaspberryPi and the LCD.

Product Measurements

Our product meets and exceeds our initial specifications. The breadboard tester and IC tester works as intended. The circuit tester is now a circuit detector and requires less input from the user, which increases user-friendliness.

Additional Issues

- *Economic* – There are no economic factors that impact the manufacture or sale of this product.
- *Environmental* – There are no issues related to the manufacture and possible disposal of the product that will impact the environment. Our product is a green design and is lead free.
- *Political* – There are no political issues that impact the manufacture or sale of this product.
- *Global* – This product was not designed for an international marketplace, but could be expanded for global use in the future.
- *Social* – There are no social issues or privacy concerns that impact the use of this product.
- *Ethical* – The product team abided by the IEEE code of ethics.
- *Health & Safety* – There are no health and safety concerns over the manufacture or normal use of the product.
- *Quality / Reliability* – The product should be fairly reliable. Any technical troubles or bad components should be easily fixed or replaced.
- *Manufacturability* – This product is manufacturable by already-existing methods with presently-available components.
- *Sustainability / Maintainability* – This product may require crimp wire replacements from time to time if they begin to wear out or become faulty. Software/Firmware upgrades can be directly downloaded from the device.

Conclusion

During the course of the design work, we came up with ideas, collaborated, and presented our ideas. Our ideas were almost never the best and many of them were not plausible, but we would not have known that unless we discussed them with each other. This is because we were not aware current existing products and flaws with our own ideas. The brainstorming of ideas really forced us to work in a group to figure what we are capable of. Then we designed the project based on our combined capabilities.

Working in a group had its ups and downs. There were quite a few times where we had disagreements on the project ideas and the design. Eventually, we were able to overcome those arguments and decide on a project idea we could all agree on. Working in a group was not all arguing and disagreements. When thinking of design alternatives, we were able to think of new ideas based on another person's bad idea.

Lastly, through this design process, we were given the opportunity to present our ideas to the rest of the class and a faculty advisor. This was a good chance to attempt to convey our ideas and receive constructive criticism.

Appendix A

User's Manual

A. Product Overview

The Breadboard, IC, and Circuit Debugging Kit is the all-in-one solution for circuit testing. This product features six 5-pin clusters specifically designed for testing portions of a breadboard for faults. The IC tester is able to check whether or not an IC functional. The IC detector will automatically try to detect the name of the IC that is inserted. The circuit tester allows for eight inputs and eight outputs and will display functions in SOP and POS form.

B. Initial Setup

Basic Setup (Required)

Now that you've gotten your brand new Breadboard, IC, & Circuit Debugging Kit, it's time to get it up and running.

1. Plug in the provided USB cable into an available power source, such as a computer with a USB port or a USB port wall outlet. Plug the Micro-USB end of the cable into the Raspberry Pi.
2. The Raspberry Pi should power on. When the OS has loaded, use the LCD touchscreen to double click the icon on the desktop named 'LXTerminal'
3. Instructions on using the program are covered in section **C. Using the Debugging Program**.

Additional Setup (Optional connecting to the internet for VNC for larger displays)

If you are connecting to the internet via cable:

1. Plug in an Ethernet cable into any available port that provides internet. Usually this should be from the wall, a router, or a switch.
2. Plug in the Ethernet cable into the available Ethernet port on the Raspberry Pi for internet.

If you are connecting to the internet via wireless:

1. Plug in the provided wireless adapter into an available USB port on the Raspberry Pi.
2. Use an external keyboard and plug it into an available USB port on the Raspberry Pi.
3. Double click the icon named 'WPA_GUI' on the desktop using the LCD touchscreen.
4. Connect to the internet by choosing the desired network SSID and entering the correct password.

How to Connect via a VNC Server

A Virtual Network Computing (VNC) Server will allow you to access and control the Breadboard, IC, & Circuit Debugging Kit with another computer or device. A VNC server is preinstalled on the Raspberry Pi, but additional software must be downloaded onto the device that wants to connect to the Raspberry Pi. (Note: Internet setup is **required** before connecting via VNC Server)

1. To be able to connect using another computer, download a VNC viewer such as TightVNC Java Viewer.
2. Open the zip file and run the file named ‘tightvnc-jviewer’
3. In the first box, enter the IP address of the Raspberry Pi. In the second box, enter the port number. Then, click connect.
4. A popup should appear showing the desktop of the Raspberry Pi.

C. Using the Debugging Program

Once ‘LXTerminal’ is running, the program is ready to be started.

1. To run the debugging program enter “`sudo python icccdebug`” into ‘LXTerminal’.
2. Once the debugging program is running, the main menu will appear and prompt for input.:

“`Enter "0" to exit, "1" to test breadboard, "2" to test IC, "3" to detect IC, and "4" to detect circuit`”

   a. Inputting ‘0’ will end the program.
   b. Inputting ‘1’ will run the breadboard tester. (Note: Header pins used for testing should be connected before running this program.)
   c. Inputting ‘2’ will run the IC tester. (Note: IC to be tested should be placed in the ZIF socket before running this program.)
   d. Inputting ‘3’ will run the IC detector. (Note: IC to be detected should be placed in the ZIF socket before running this program.)
   e. Inputting ‘4’ will run the circuit tester. (Note: Circuit to be tested should be connected before running this program.)

The Breadboard Tester

1. To use the breadboard tester, place the set of six by five header pins into the portion of the breadboard to be tested. (Note: Header pins are **NOT** required to be placed in a specific orientation for testing.)
2. After the header pins are connected to the desired portion of the breadboard, inputting ‘1’ at the main menu will run the breadboard testing program.
3. A 5 x 6 matrix of numbers will appear. For example:

```
0  0  7 12 17 22
1  5  8 13 18 22
2  6  9 14 19 22
3  0 10 15 20 22
4  0 11 16 21 22
```

   Each column of numbers corresponds to a set of header pins used for testing. The first column corresponds to the red set of header pins, the second column for orange, the third column for yellow, the fourth column for green, the fifth column for blue and the sixth column for purple.

   Every distinct number represents a distinct connection. Duplicate numbers represents pins that share the same connection.

The IC Tester

1. To use the IC tester, place the IC into the ZIF socket with the reference of the IC facing the lever of the ZIF socket as shown on the left of Figure 7. (Note: Make sure the lever is in a **vertical** position before inserting the IC.)

Figure 7: Left shows ZIF socket before IC insertion. Right shows ZIF socket after IC insertion

2. After the IC is inserted, push down the lever into a **horizontal** position to lock it in place as shown on the right of Figure 7.
3. Now, input ‘2’ at the main menu and the IC testing menu will appear.

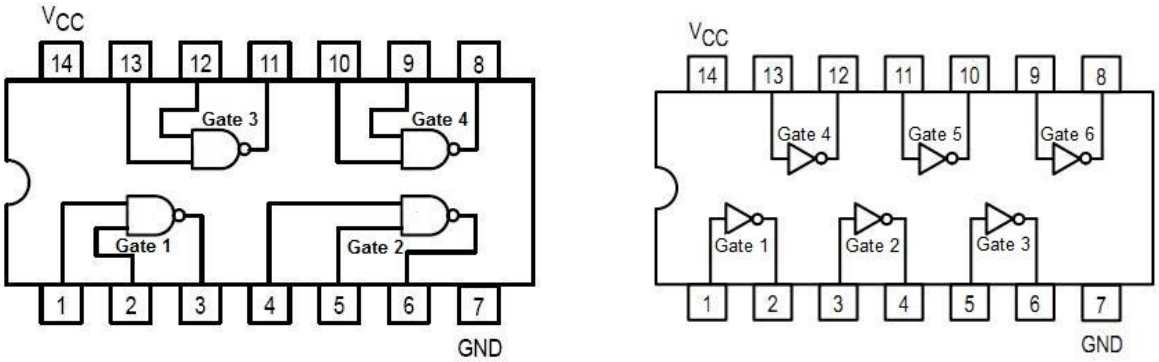
```
Enter "0" to return, "1" for NAND, "2" for NOR, "3" for NOT, "4" for AND, and
"5" for OR, "6" for XOR
```

   a. Inputting ‘0’ or ‘return’ will end the IC tester and return to the main menu,
   b. Inputting ‘1’ or ‘nand’ will check if the inserted IC is a 74LS00 (Quad NAND).
   c. Inputting ‘2’ or ‘nor’ will check if the inserted IC is a 74LS02 (Quad NOR).
   d. Inputting ‘3’ or ‘not’ will check if the inserted IC is a 74LS04 (Hex Inverter).
   e. Inputting ‘4’ or ‘and’ will check if the inserted IC is a 74LS08 (Quad AND).
   f. Inputting ‘5’ or ‘or’ will check if the inserted IC is a 74LS32 (Quad OR).
   g. Inputting ‘6’ or ‘xor’ will check if the inserted IC is a 74LS86 (Quad XOR).

4. A message will indicated if the IC tested is correct. If incorrect values are outputted by the IC, a message will specify the gate that produced the wrong output as well as the input and output that gave an unexpected result. (Note: The gates of specific ICs are numbered beginning with the lower left gate as gate 1 ascending to the lower right and will continue ascending beginning at the upper left ascending to the upper right.)

The IC Detector

1. To use the IC detector, place the IC into the ZIF socket with the reference of the IC facing the lever of the ZIF socket. (Note: Make sure the lever is in a **vertical** position before inserting the IC.)



2. After the IC is inserted, push down the lever into a **horizontal** position to lock it in place.
3. After the IC is locked in place, inputting ‘3’ at the main menu will run the breadboard testing program.

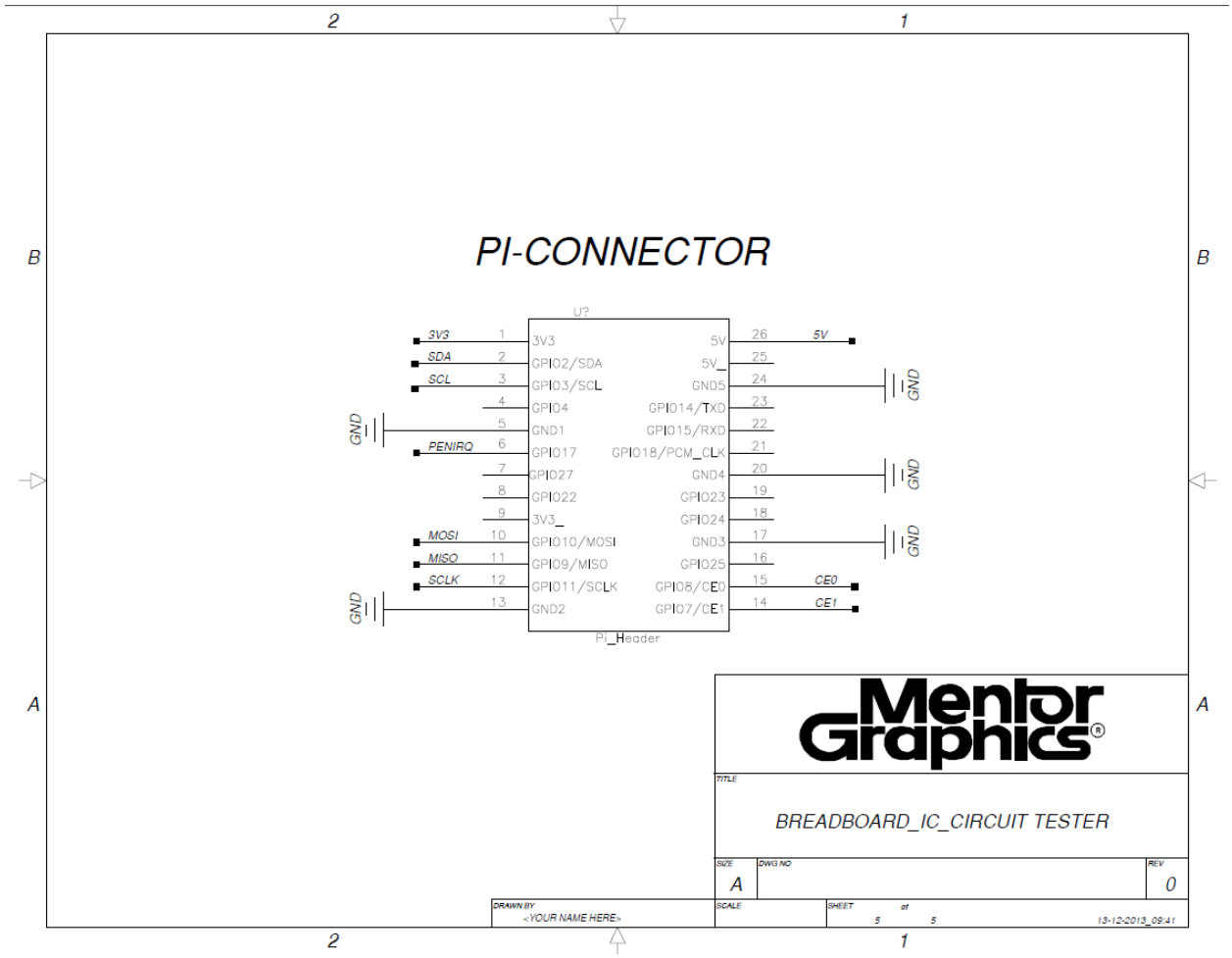
4. The program will indicated which IC is detected. The program will also indicated if the IC is not one that is recognized.

The Circuit Tester

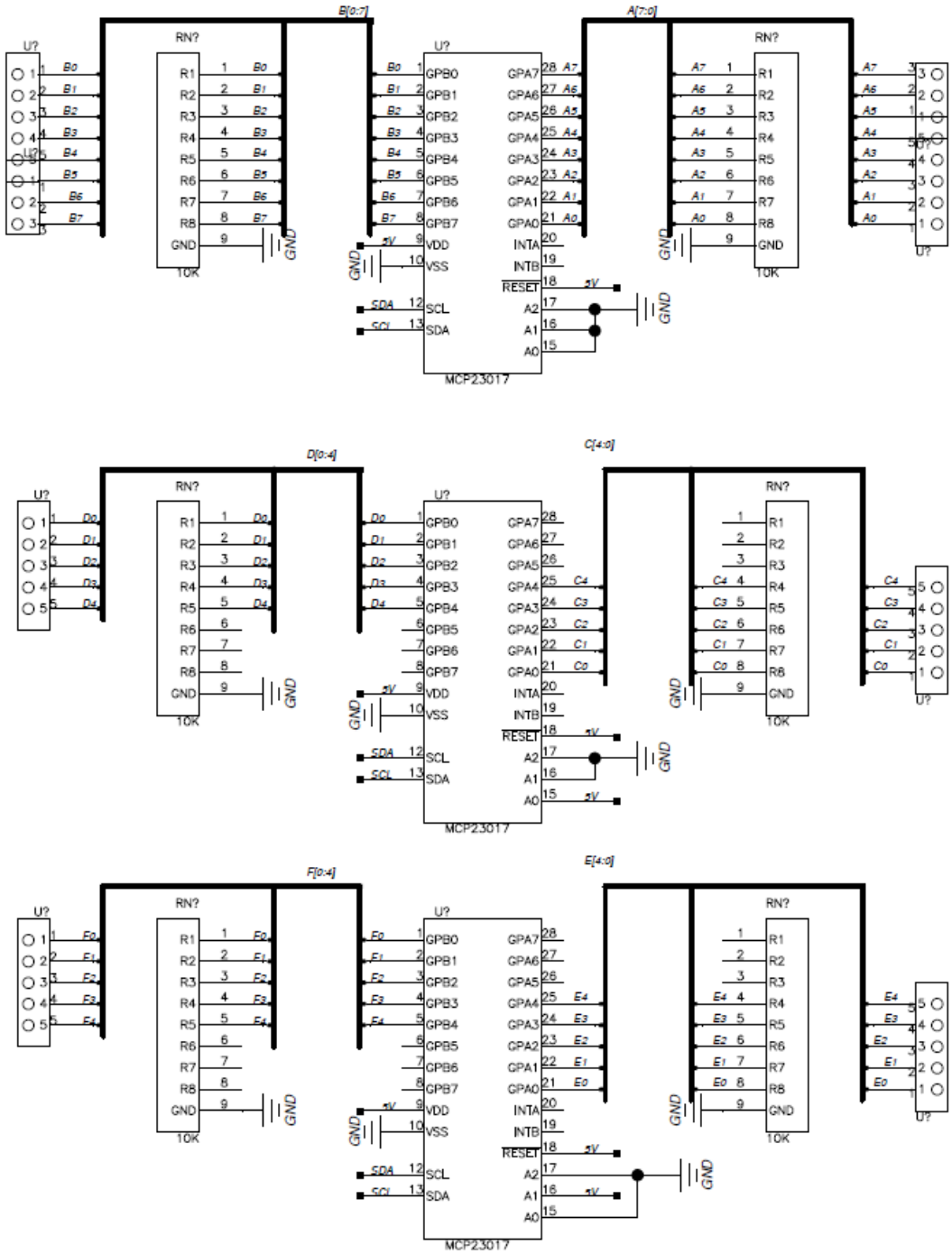
1. To use the circuit tester connect the inputs of the circuit to be tested to the 5 red pins and 3 white wires of the debugging kit and the outputs of the circuit to be tested to the 5 orange wires and 3 black wires of the debugging kit.
2. After the circuit is properly connected to the debugging kit, inputting ‘4’ at the main menu will bring up the circuit testing menu.
3. The program will then prompt for the number of inputs and number of outputs of the circuit to be tested.
4. After that, the circuit testing program will then display both the SOP and POS forms of each output of the circuit. (Note: The results the circuit testing program will contain letters that represent the input values eg. a for input 1, b for input 2, etc.)

Appendix B

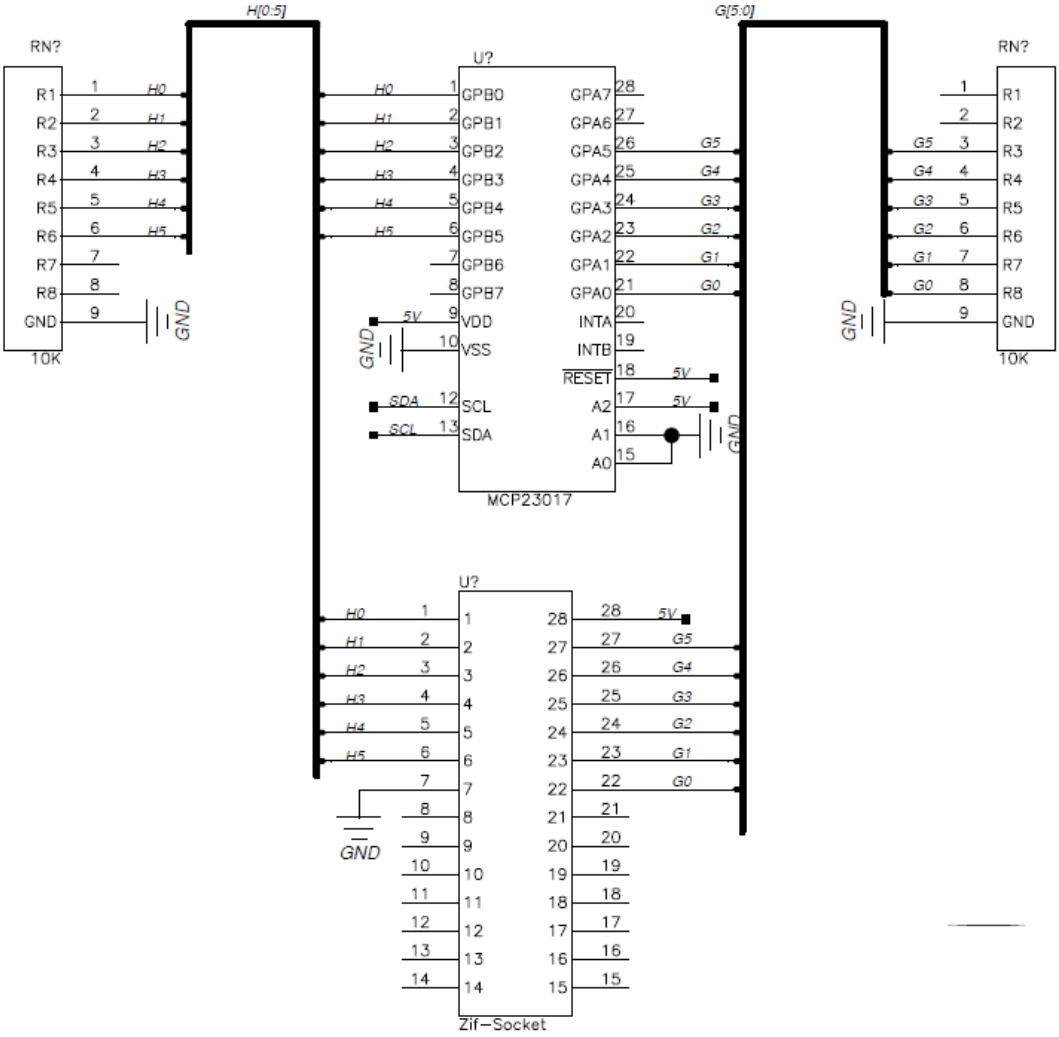
Part	Part #	Vendor(Links)
Header Pins	0966	http://www.pololu.com/catalog/product/966
Header Pin Housing	1905	http://www.pololu.com/catalog/product/1905/specs
Crimped Wires	74HC595	http://www.pololu.com/product/1805
Input/Output Port Expander	MCP20317	http://www.adafruit.com/products/732
28 Pin Socket	9175	https://www.sparkfun.com/products/9175
LCD Screen	11677	https://www.sparkfun.com/products/11677
Raspberry Pi	RASPBRRY-MODB-512M	http://www.newark.com/raspberry-pi/raspbrry-modb-512m/model-b-assembled-board-only/dp/43W5302?COM=raspi-group
Wireless USB Adapter	ENUWI-G2	http://www.newegg.com/Product/Product.aspx?Item=N82E16833180017

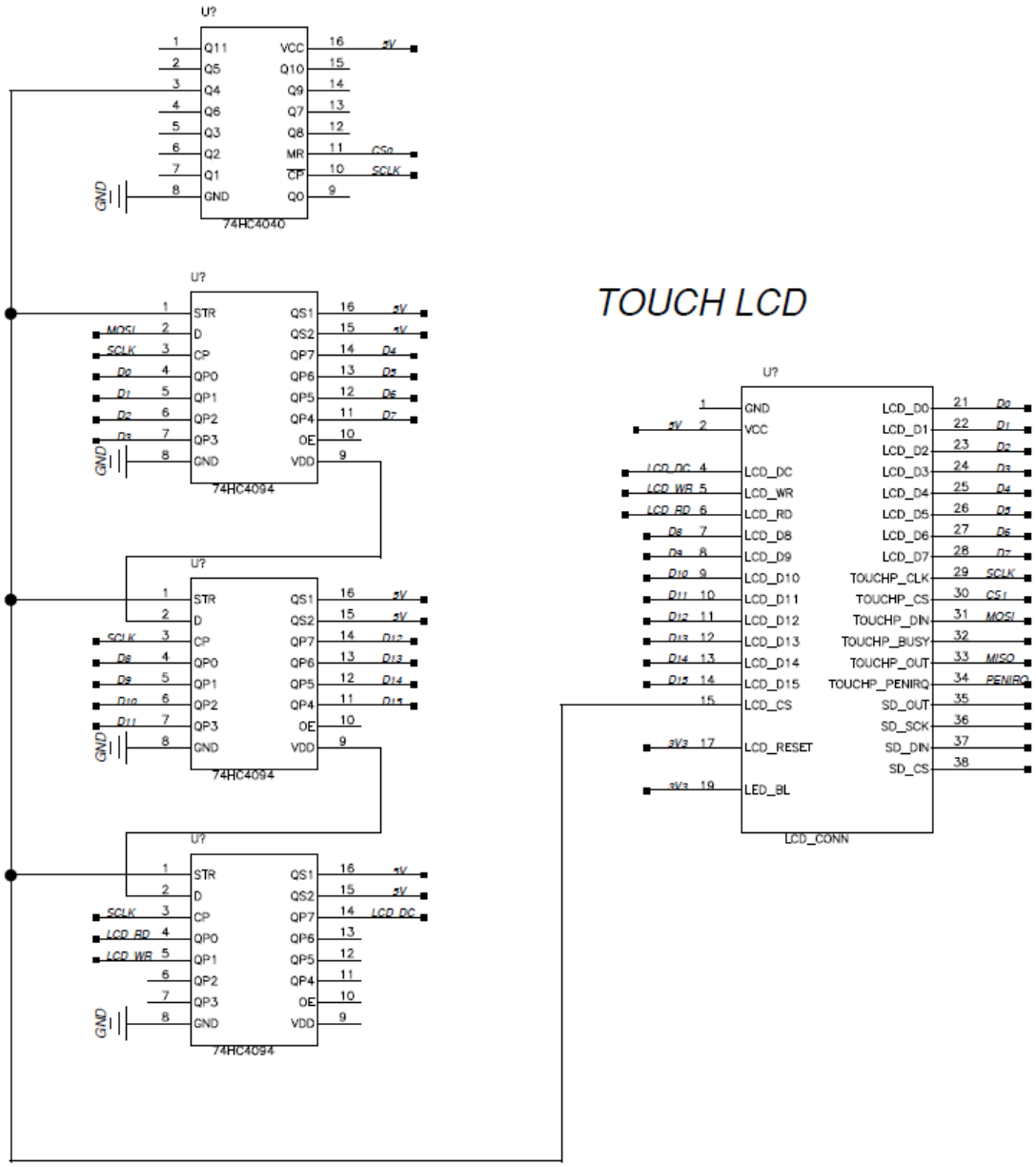

PI-CONNECTOR
U?
3V3
SDA
SCL
PENIRQ
MOSI
MISO
SCLK
GND
5V
CE0
CE1
3V3
GPIO2/SDA
GPIO3/SCL
GPIO4
GND1
GPIO17
GPIO27
GPIO22
3V3_
GPIO10/MOSI
GPIO9/MISO
GPIO11/SCLK
GND2
5V
5V_
GND5
GPIO14/TXD
GPIO15/RXD
GPIO18/PCM_CLK
GND4
GPIO23
GPIO24
GND3
GPIO25
GPIO8/CE0
GPIO7/CE1
PI_Header
Mentor Graphics
TITLE
BREADBOARD_IC_CIRCUIT TESTER
SIZE
A
DWG NO
REV
0
DRAWN BY
<YOUR NAME HERE>
SCALE
SHEET
of
5
5
13-12-2013_09:41

BREADBOARD & CIRCUIT TESTER



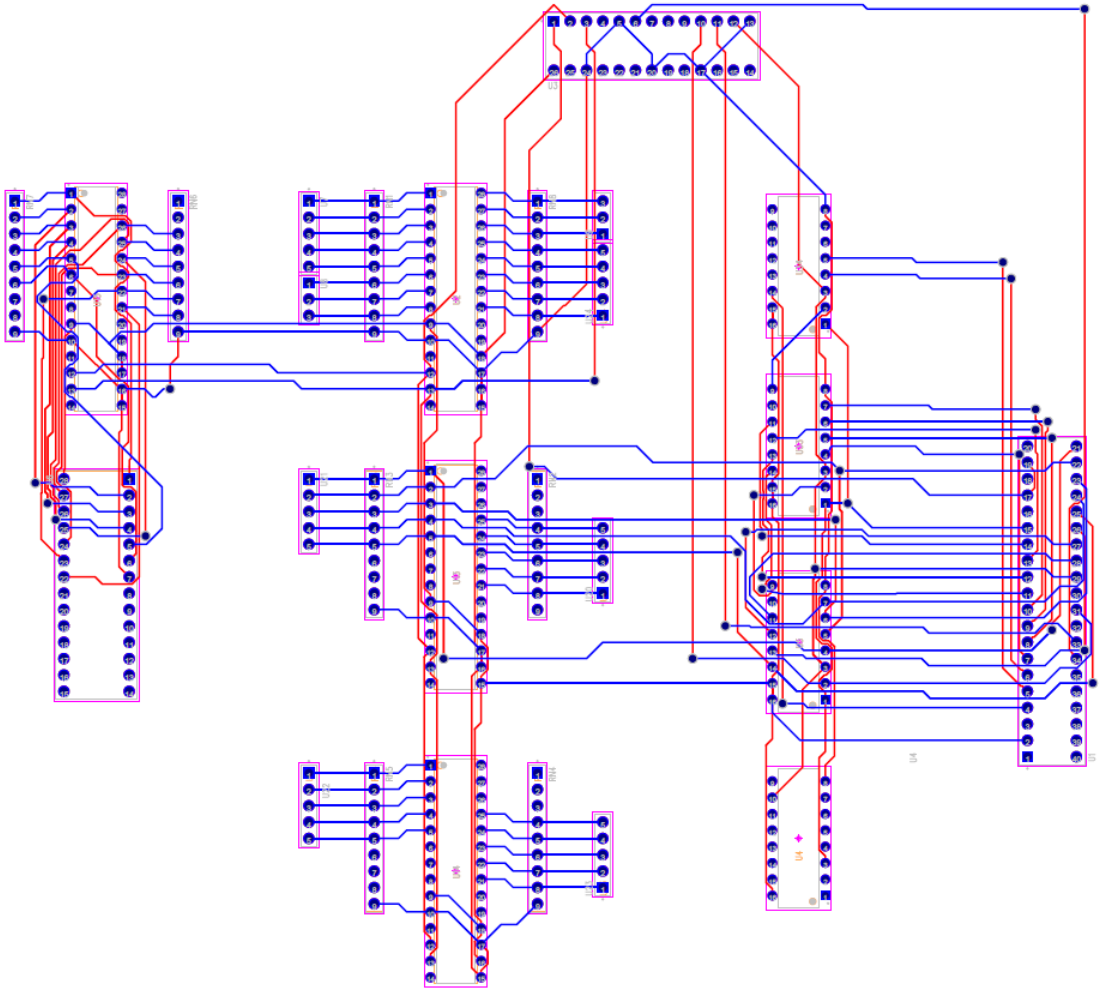
IC TESTER


H[0:5]
G[5:0]
RN?
R1
R2
R3
R4
R5
R6
R7
R8
GND
10K
U?
GPB0
GPB1
GPB2
GPB3
GPB4
GPB5
GPB6
GPB7
VDD
VSS
SCL
SDA
GPA7
GPA6
GPA5
GPA4
GPA3
GPA2
GPA1
GPA0
INTA
INTB
RESET
A2
A1
A0
5V
MCP23017
Zif-Socket

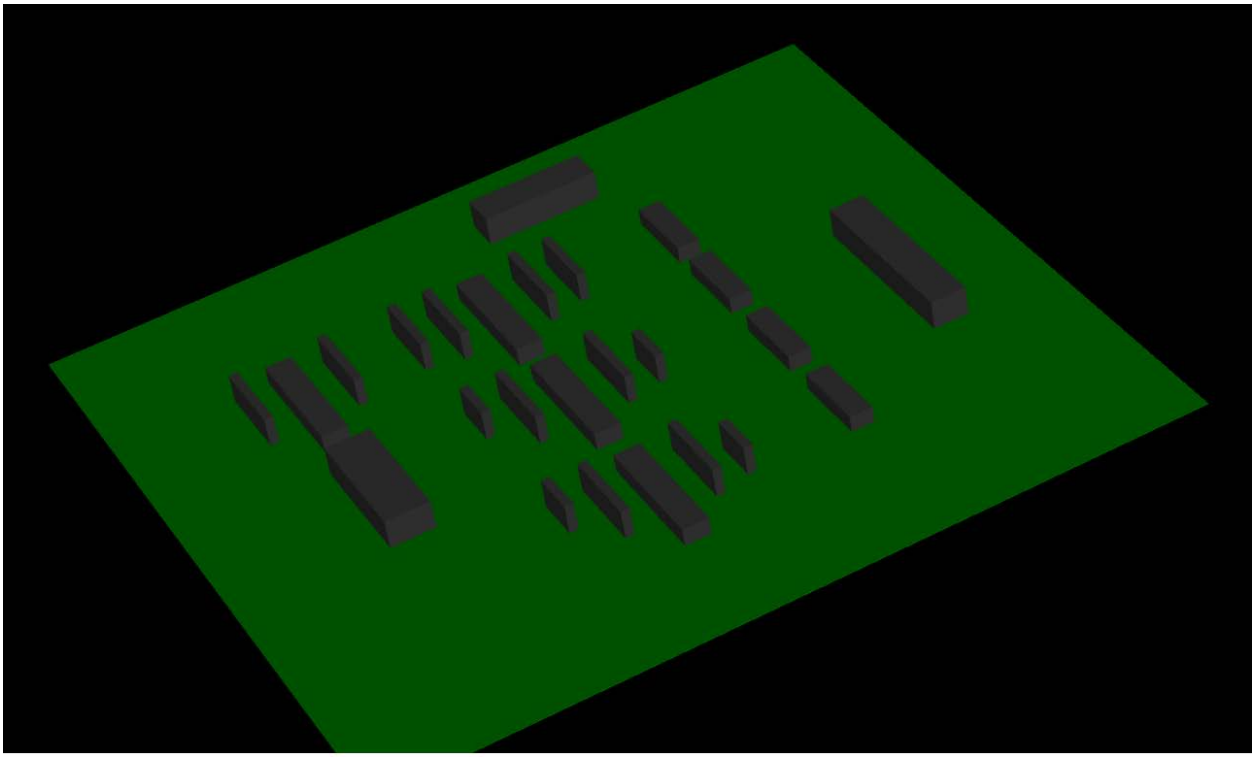

TOUCH LCD
U?
74HC4040
Q11
Q5
Q4
Q6
Q3
Q2
Q1
GND
VCC
Q10
Q9
Q7
Q8
MR
CP
Q0
5V
SCLK
74HC4094
STR
D
CP
QP0
QP1
QP2
QP3
OE
VDD
QS1
QS2
QP7
QP6
QP5
QP4
MOSI
LCD_RD
LCD_WR
LCD_DC
LCD_CONN
LCD_D0
LCD_D1
LCD_D2
LCD_D3
LCD_D4
LCD_D5
LCD_D6
LCD_D7
LCD_D8
LCD_D9
LCD_D10
LCD_D11
LCD_D12
LCD_D13
LCD_D14
LCD_D15
LCD_CS
LCD_RESET
LED_BL
TOUCHP_CLK
TOUCHP_CS
TOUCHP_DIN
TOUCHP_BUSY
TOUCHP_OUT
TOUCHP_PENIRQ
SD_OUT
SD_SCK
SD_DIN
SD_CS
3V3

PCB Layout:

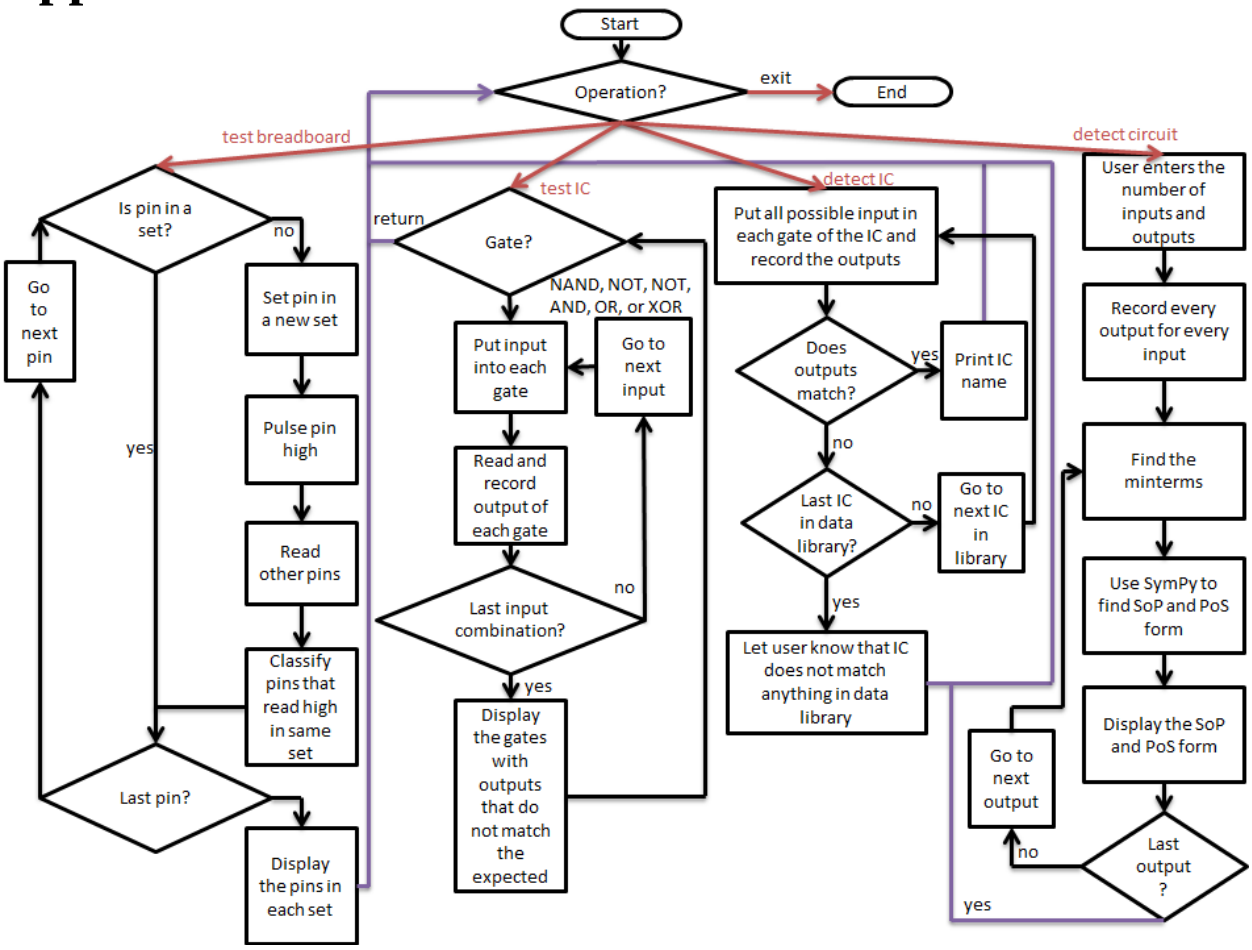
Dual-Layer



3D-View



Appendix C



Appendix D

Hardware requirements:

Raspberry Pi was used to run the program.
Breadboard was used for all necessary circuitry.
MCP20317 (Input/Output Port Expanders) were used to increase the amount of input and output ports because the Raspberry Pi did not have enough GPIO ports for our design.
A wireless network adapter was used to allow for more portability.
SD card for loading the operating system.

Software Requirements:

Raspbian OS was our operating system of choice because of its overall ease of use and bundled software. The code used for the program is written in Python 2.7.
PuTTY was used to Secure Shell into the Raspberry Pi in order to prepare the necessary tools for running the Python code.
A Virtual Network Computing Server allows users to connect to and control the Raspberry Pi with another computer.

Appendix E

```
import smbus
import time
import math
from sympy.logic import SOPform
from sympy.logic import POSform

def getTextSegment(bitValue, length): # given a value, return array of bits
    returnValue = [0 for i in range (0, length)]
    for i in range (0, length):
        returnValue[length - 1 - i] = int(bitValue % 2)
        bitValue /= 2;
    return returnValue

def parseSOP(sopString, numberOfInputs): # given a sop string, parse it to
make it easier to read
    sopString = str.replace(sopString, " ", "")
    for i in range(0, int(numberOfInputs)):
        sopString = str.replace(sopString, "Not(" + chr(ord('a') + i) + ")",
chr(ord('a') + i) + "'")
    sopString = str.replace(sopString, "Or", "")
    if str.find(sopString, "And") < 0:
        sopString = str.replace(sopString, ",", "+")
    sopString = str.replace(sopString, ",And", "+")
    sopString = str.replace(sopString, "),", "+")
    sopString = str.replace(sopString, "And", "")
    sopString = str.replace(sopString, "(", "")
    sopString = str.replace(sopString, ")", "")
    sopString = str.replace(sopString, ",", "")
    return sopString

def parsePOS(posString, numberOfInputs): # given a pos string, parse it to
make it easier to read
    posString = str.replace(posString, " ", "")
    for i in range(0, int(numberOfInputs)):
        posString = str.replace(posString, "Not(" + chr(ord('a') + i) + ")",
chr(ord('a') + i) + "'")
    if str.find(posString, "And(") >= 0:
        posString = str.replace(posString, "And(", "")
        posString = posString[0:len(posString)-1]
    posString = str.replace(posString, ",Or", "")
    posString = str.replace(posString, "),", ")")
    posString = str.replace(posString, "Or", "")
    posString = str.replace(posString, ",", "+")
    return posString

def testBreadboard():
    bus = smbus.SMBus(1)                                            # Rev
1 Pi uses 0, Rev 2 Pi uses 1
    DEVICE = 0x20                                                   #
Device address (A0-A2)
    IODIRA = 0x00                                                   # Pin
direction register
    IODIRB = 0x01                                                   # Pin
direction register
```

```
    OLATA  = 0x14                                                    #
Register for outputs
    OLATB  = 0x15                                                    #
Register for outputs
    GPIOA  = 0x12                                                    #
Register for inputs
    GPIOB  = 0x13

    Matrix = [[0x20, 0x00, 0x14, 0x12], [0x20, 0x01, 0x15, 0x13], [0x21,
0x00, 0x14, 0x12], [0x21, 0x01, 0x15, 0x13], [0x22, 0x00, 0x14, 0x12], [0x22,
0x01, 0x15, 0x13]]
    # device, iodir, olat, gpio
    IndependentSets = [[0 for j in range(6)] for i in range(5)]         # 1st
D is the 5 bits, 2nd D is the 6 devices
    for i in range (5):
        for j in range(6):
            IndependentSets[i][j] = -1

    for Device in range(0,6):
        bus.write_byte_data(Matrix[Device][0],Matrix[Device][2],0)      #
initialize all registers to output data 0
        bus.write_byte_data(Matrix[Device][0],Matrix[Device][1],0xFF)   # set
all registers to input
        bus.write_byte_data(Matrix[Device][0],0x0C,0)                   # set
all registers to input
        bus.write_byte_data(Matrix[Device][0],0x0D,0)                   # set
all registers to input

    IndependentSetsCounter = 0                                          #
keep track of the number of independent sets
    for Data in range(0,30):
        if (IndependentSets[Data%5][int(Data/5)] == -1):                #
create a new independent set if needed
            IndependentSets[Data%5][int(Data/5)] = IndependentSetsCounter
            IndependentSetsCounter = IndependentSetsCounter + 1
        dataBit = int(math.pow(2, Data%5))
        bus.write_byte_data(Matrix[Data/5][0],Matrix[Data/5][1],0xFF-dataBit)
# set direction
        bus.write_byte_data(Matrix[Data/5][0],Matrix[Data/5][2],dataBit)#
pulse the data
        for Device in range(0,6):                                       #
look through all 6 registers
            register =
bus.read_byte_data(Matrix[Device][0],Matrix[Device][3]) # read the data
            if register > 0:                                            # if
the register > 0
                bitsToUpdate = getTextSegment(register, 8);                #
then update the independent sets
                for bit in range(5):
                    if bitsToUpdate[7-bit] > 0 and
IndependentSets[bit][Device] == -1:
                        IndependentSets[bit][Device] = IndependentSetsCounter
- 1
        bus.write_byte_data(Matrix[Data/5][0],Matrix[Data/5][2],0)      # set
register to output data 0
```

```
        bus.write_byte_data(Matrix[Data/5][0],Matrix[Data/5][1],0xFF)   # set
register to input
    for i in range (5):
        print("%2d %2d %2d %2d %2d %2d" % (IndependentSets[i][0],
IndependentSets[i][1], IndependentSets[i][2], IndependentSets[i][3],
IndependentSets[i][4], IndependentSets[i][5]))
    print("done with breadboard tester\n")

def testCircuit():
    bus = smbus.SMBus(1)                                            # Rev
1 Pi uses 0, Rev 2 Pi uses 1
    DEVICE = 0x20                                                   #
Device address (A0-A2)
    IODIRA = 0x00                                                   # Pin
direction register
    IODIRB = 0x01                                                   # Pin
direction register
    OLATA  = 0x14                                                   #
Register for outputs
    OLATB  = 0x15                                                   #
Register for outputs
    GPIOA  = 0x12                                                   #
Register for inputs
    GPIOB  = 0x13

    # device, iodir, olat, gpio
    bus.write_byte_data(DEVICE, IODIRA, 0x00)                       # set
direction
    bus.write_byte_data(DEVICE, OLATA, 0x00)                        #
initialize all registers to output data 0
    bus.write_byte_data(DEVICE, IODIRB, 0xFF)                       # set
direction
    bus.write_byte_data(DEVICE, OLATB, 0x00)                        #
initialize all registers to output data 0

    numberOfInputs = input("Number of Inputs: ")
    numberOfOutputs = input("Number of Outputs: ")
    numberOfTestVectorInputs = int(math.pow(2, numberOfInputs))
    numberOfTestVectorOutputs = int(math.pow(2, numberOfOutputs))
    output = [0 for i in range (0, numberOfTestVectorInputs)]
    for Data in range(0, int(numberOfTestVectorInputs)):
        bus.write_byte_data(DEVICE, OLATA, Data)                    #
write the data to output bus
        output[Data] = getTextSegment(bus.read_byte_data(DEVICE, GPIOB),
numberOfOutputs) # read data from input bus

    inputs = ['a' for i in range (0, numberOfInputs)]
    for i in range(0, int(numberOfInputs)):                         #
convert the inputs to letters
        inputs[i] = chr(ord('a') + numberOfInputs - 1 - i)

    for outputWalker in range(0, numberOfOutputs):                  # for
every output
        minterms = []                                               #
reset the set of minterms
        for Data in range(0, numberOfTestVectorInputs):
```

```
            if output[Data][numberOfOutputs - 1 - outputWalker] > 0:    # add
to set if minterms if output > 0
                minterms.append(getTextSegment(Data, numberOfInputs))
        sopString = str(SOPform(inputs, minterms, ))                    #
compute output in SOP form
        posString = str(POSform(inputs, minterms, ))                    #
compute output in POS form
        print("output" + str(outputWalker) + " in SOP = " +
parseSOP(sopString, numberOfInputs))
        print("output" + str(outputWalker) + " in POS = " +
parsePOS(posString, numberOfInputs))
    print("done with circuit detector\n")

bus = smbus.SMBus(1) # Rev 2 Pi uses 1

DEVICE = 0x23 # Device address (A0-A2)
IODIRA = 0x00 # Pin direction register
IODIRB = 0x01 # Pin direction register
OLATA  = 0x14 # Register for outputs
OLATB  = 0x15 # Register for outputs
GPIOA  = 0x12 # Register for inputs
GPIOB  = 0x13 # Register for inputs

correctNANDList=[['1','1','1','0'],['1','1','1','0'],['1','1','1','0'],['1','
1','1','0']]
correctNORList=[['1','0','0','0'],['1','0','0','0'],['1','0','0','0'],['1','0
','0','0']]
correctNOTList=[['1','0'],['1','0'],['1','0'],['1','0'],['1','0'],['1','0']]
correctANDList=[['0','0','0','1'],['0','0','0','1'],['0','0','0','1'],['0','0
','0','1']]
correctORList=[['0','1','1','1'],['0','1','1','1'],['0','1','1','1'],['0','1'
,'1','1']]
correctXORList=[['0','1','1','0'],['0','1','1','0'],['0','1','1','0'],['0','1
','1','0']]

row4List=[['0','0','1','1'],['0','0','1','1'],['0','0','1','1'],['0','0','1',
'1']]
column4List=[['0','1','0','1'],['0','1','0','1'],['0','1','0','1'],['0','1','
0','1']]
row2List=[['0','1'],['0','1'],['0','1'],['0','1'],['0','1'],['0','1']]

def testIC():
    while True:
        gateName = raw_input("Enter \"0\" to return, \"1\" for NAND, \"2\"
for NOR, \"3\" for NOT, \"4\" for AND, and \"5\" for OR, \"6\" for XOR\n > ")
        if gateName == "0" or gateName.lower() == "return" or
gateName.lower() == "end":
            break
        elif gateName == "1" or gateName.lower() == "nand" or
gateName.lower() == "74ls00":
            NANDTest()
        elif gateName == "2" or gateName.lower() == "nor" or gateName.lower()
== "74ls02":
            NORTest()
        elif gateName == "3" or gateName.lower() == "not" or gateName.lower()
== "74ls04":
```

```
            NOTTest()
        elif gateName == "4" or gateName.lower() == "and" or gateName.lower()
== "74ls08":
            ANDTest()
        elif gateName == "5" or gateName.lower() == "or" or gateName.lower()
== "74ls32":
            ORTest()
        elif gateName == "6" or gateName.lower() == "xor" or gateName.lower()
== "74ls86":
            XORTest()
        else:
            print("invalid input: " + gateName)

#NAND
def NANDTest():
    bus.write_byte_data(DEVICE,IODIRA,0x24) #00100100
    bus.write_byte_data(DEVICE,IODIRB,0x09) #00001001
    NANDList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NANDList[0][0]=stringA[5]
    NANDList[1][0]=stringA[2]
    NANDList[2][0]=stringB[4]
    NANDList[3][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
    bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NANDList[0][1]=stringA[5]
    NANDList[1][1]=stringA[2]
    NANDList[2][1]=stringB[4]
    NANDList[3][1]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x09) #00001001
    bus.write_byte_data(DEVICE,OLATB,0x24) #00100100
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
```

```
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NANDList[0][2]=stringA[5]
    NANDList[1][2]=stringA[2]
    NANDList[2][2]=stringB[4]
    NANDList[3][2]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x1B) #00011011
    bus.write_byte_data(DEVICE,OLATB,0x36) #00110110
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NANDList[0][3]=stringA[5]
    NANDList[1][3]=stringA[2]
    NANDList[2][3]=stringB[4]
    NANDList[3][3]=stringB[7]

    match = True
    gateWorks = True
    for i in range(0,4):
        for j in range(0,4):
            if NANDList[i][j] != correctNANDList[i][j]:
                match = False
                gateWorks = False
            if match == False:
                    print("NAND Gate " + str(i+1) + " of 74LS00 outputs " +
NANDList[i][j] + " when " + row4List[i][j] + " and " + column4List[i][j] + "
are inputted.")
                    match = True
    if gateWorks == True:
        print("74LS00 is functioning properly.")

#NOR
def NORTest():
    bus.write_byte_data(DEVICE,IODIRA,0x09) #00001001
    bus.write_byte_data(DEVICE,IODIRB,0x24) #00100100
    NORList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NORList[0][0]=stringA[7]
    NORList[1][0]=stringA[4]
```

```
NORList[2][0]=stringB[2]
NORList[3][0]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x24) #00100100
bus.write_byte_data(DEVICE,OLATB,0x09) #00001001
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
NORList[0][1]=stringA[7]
NORList[1][1]=stringA[4]
NORList[2][1]=stringB[2]
NORList[3][1]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
NORList[0][2]=stringA[7]
NORList[1][2]=stringA[4]
NORList[2][2]=stringB[2]
NORList[3][2]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x36) #00110110
bus.write_byte_data(DEVICE,OLATB,0x1B) #00011011
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
NORList[0][3]=stringA[7]
NORList[1][3]=stringA[4]
NORList[2][3]=stringB[2]
NORList[3][3]=stringB[5]

match = True
gateWorks = True
for i in range(0,4):
    for j in range(0,4):
        if NORList[i][j] != correctNORList[i][j]:
            match = False
            gateWorks = False
        if match == False:
```

```
                    print("NOR Gate " + str(i+1) + " of 74LS02 outputs " +
NORList[i][j] + " when " + row4List[i][j] + " and " + column4List[i][j] + "
are inputted.")
                    match = True
    if gateWorks == True:
        print("74LS02 is functioning properly.")

#NOT
def NOTTest():
    bus.write_byte_data(DEVICE,IODIRA,0x2A) #00101010
    bus.write_byte_data(DEVICE,IODIRB,0x15) #00010101
    NOTList=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NOTList[0][0]=stringA[6]
    NOTList[1][0]=stringA[4]
    NOTList[2][0]=stringA[2]
    NOTList[3][0]=stringB[3]
    NOTList[4][0]=stringB[5]
    NOTList[5][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x15) #00010101
    bus.write_byte_data(DEVICE,OLATB,0x2A) #00101010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    NOTList[0][1]=stringA[6]
    NOTList[1][1]=stringA[4]
    NOTList[2][1]=stringA[2]
    NOTList[3][1]=stringB[3]
    NOTList[4][1]=stringB[5]
    NOTList[5][1]=stringB[7]

    match = True
    gateWorks = True
    for i in range(0,6):
        for j in range(0,2):
            if NOTList[i][j] != correctNOTList[i][j]:
                match = False
                gateWorks = False
            if match == False:
```

```
                    print("NOT Gate " + str(i+1) + " of 74LS04 outputs " +
NOTList[i][j] + " when " + row2List[i][j] + " is inputted.")
                    match = True
    if gateWorks == True:
        print("74LS04 is functioning properly.")

#AND
def ANDTest():
    bus.write_byte_data(DEVICE,IODIRA,0x24) #00100100
    bus.write_byte_data(DEVICE,IODIRB,0x09) #00001001
    ANDList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ANDList[0][0]=stringA[5]
    ANDList[1][0]=stringA[2]
    ANDList[2][0]=stringB[4]
    ANDList[3][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
    bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ANDList[0][1]=stringA[5]
    ANDList[1][1]=stringA[2]
    ANDList[2][1]=stringB[4]
    ANDList[3][1]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x09) #00001001
    bus.write_byte_data(DEVICE,OLATB,0x24) #00100100
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ANDList[0][2]=stringA[5]
    ANDList[1][2]=stringA[2]
    ANDList[2][2]=stringB[4]
    ANDList[3][2]=stringB[7]
```

```
    bus.write_byte_data(DEVICE,OLATA,0x1B) #00011011
    bus.write_byte_data(DEVICE,OLATB,0x36) #00110110
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ANDList[0][3]=stringA[5]
    ANDList[1][3]=stringA[2]
    ANDList[2][3]=stringB[4]
    ANDList[3][3]=stringB[7]

    match = True
    gateWorks = True
    for i in range(0,4):
        for j in range(0,4):
            if ANDList[i][j] != correctANDList[i][j]:
                match = False
                gateWorks = False
            if match == False:
                    print("AND Gate " + str(i+1) + " of 74LS08 outputs " +
ANDList[i][j] + " when " + row4List[i][j] + " and " + column4List[i][j] + "
are inputted.")
                    match = True
    if gateWorks == True:
        print("74LS08 is functioning properly.")

#OR
def ORTest():
    bus.write_byte_data(DEVICE,IODIRA,0x24) #00100100
    bus.write_byte_data(DEVICE,IODIRB,0x09) #00001001
    ORList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ORList[0][0]=stringA[5]
    ORList[1][0]=stringA[2]
    ORList[2][0]=stringB[4]
    ORList[3][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
    bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
```

```
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ORList[0][1]=stringA[5]
    ORList[1][1]=stringA[2]
    ORList[2][1]=stringB[4]
    ORList[3][1]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x09) #00001001
    bus.write_byte_data(DEVICE,OLATB,0x24) #00100100
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ORList[0][2]=stringA[5]
    ORList[1][2]=stringA[2]
    ORList[2][2]=stringB[4]
    ORList[3][2]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x1B) #00011011
    bus.write_byte_data(DEVICE,OLATB,0x36) #00110110
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    ORList[0][3]=stringA[5]
    ORList[1][3]=stringA[2]
    ORList[2][3]=stringB[4]
    ORList[3][3]=stringB[7]

    match = True
    gateWorks = True
    for i in range(0,4):
        for j in range(0,4):
            if ORList[i][j] != correctORList[i][j]:
                match = False
                gateWorks = False
            if match == False:
                    print("OR Gate " + str(i+1) + " of 74LS32 outputs " +
ORList[i][j] + " when " + row4List[i][j] + " and " + column4List[i][j] + "
are inputted.")
                    match = True
    if gateWorks == True:
        print("74LS32 is functioning properly.")
```

```
#XOR
def XORTest():
    bus.write_byte_data(DEVICE,IODIRA,0x24) #00100100
    bus.write_byte_data(DEVICE,IODIRB,0x09) #00001001
    XORList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    XORList[0][0]=stringA[5]
    XORList[1][0]=stringA[2]
    XORList[2][0]=stringB[4]
    XORList[3][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
    bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    XORList[0][1]=stringA[5]
    XORList[1][1]=stringA[2]
    XORList[2][1]=stringB[4]
    XORList[3][1]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x09) #00001001
    bus.write_byte_data(DEVICE,OLATB,0x24) #00100100
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    XORList[0][2]=stringA[5]
    XORList[1][2]=stringA[2]
    XORList[2][2]=stringB[4]
    XORList[3][2]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x1B) #00011011
    bus.write_byte_data(DEVICE,OLATB,0x36) #00110110
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
```

```
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    XORList[0][3]=stringA[5]
    XORList[1][3]=stringA[2]
    XORList[2][3]=stringB[4]
    XORList[3][3]=stringB[7]

    match = True
    gateWorks = True
    for i in range(0,4):
        for j in range(0,4):
            if XORList[i][j] != correctXORList[i][j]:
                match = False
                gateWorks = False
            if match == False:
                    print("XOR Gate " + str(i+1) + " of 74LS86 outputs " +
XORList[i][j] + " when " + row4List[i][j] + " and " + column4List[i][j] + "
are inputted.")
                    match = True
    if gateWorks == True:
        print("74LS86 is functioning properly.")

def gateDetector():
    bus.write_byte_data(DEVICE,IODIRA,0x24) #00100100
    bus.write_byte_data(DEVICE,IODIRB,0x09) #00001001
    detectList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

    bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
    bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    detectList[0][0]=stringA[5]
    detectList[1][0]=stringA[2]
    detectList[2][0]=stringB[4]
    detectList[3][0]=stringB[7]

    bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
    bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
    longA = bus.read_byte_data(DEVICE,GPIOA)
    longB = bus.read_byte_data(DEVICE,GPIOB)
    stringA=""
    stringB=""
    for i in range(8):
        stringA += str(getTextSegment(longA, 8)[i])
    for i in range(8):
        stringB += str(getTextSegment(longB, 8)[i])
    detectList[0][1]=stringA[5]
```

```
detectList[1][1]=stringA[2]
detectList[2][1]=stringB[4]
detectList[3][1]=stringB[7]

bus.write_byte_data(DEVICE,OLATA,0x09) #00001001
bus.write_byte_data(DEVICE,OLATB,0x24) #00100100
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][2]=stringA[5]
detectList[1][2]=stringA[2]
detectList[2][2]=stringB[4]
detectList[3][2]=stringB[7]

bus.write_byte_data(DEVICE,OLATA,0x1B) #00011011
bus.write_byte_data(DEVICE,OLATB,0x36) #00110110
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][3]=stringA[5]
detectList[1][3]=stringA[2]
detectList[2][3]=stringB[4]
detectList[3][3]=stringB[7]

matchNAND = True
for i in range(0,4):
    for j in range(0,4):
        if detectList[i][j] != correctNANDList[i][j]:
            matchNAND = False
        if matchNAND == False:
            break
    if matchNAND == False:
        break
if matchNAND == True:
    print('The IC should be 74LS00 = NAND.')
    return

matchAND = True
for i in range(0,4):
    for j in range(0,4):
        if detectList[i][j] != correctANDList[i][j]:
            matchAND = False
        if matchAND == False:
            break
    if matchAND == False:
        break
```

```
if matchAND == True:
    print('The IC should be 74LS08 = AND.')
    return

matchOR = True
for i in range(0,4):
    for j in range(0,4):
        if detectList[i][j] != correctORList[i][j]:
            matchOR = False
        if matchOR == False:
            break
    if matchOR == False:
        break
if matchOR == True:
    print('The IC should be 74LS32 = OR.')
    return

matchXOR = True
for i in range(0,4):
    for j in range(0,4):
        if detectList[i][j] != correctXORList[i][j]:
            matchXOR = False
        if matchXOR == False:
            break
    if matchXOR == False:
        break
if matchXOR == True:
    print('The IC should be 74LS86 = XOR.')
    return

bus.write_byte_data(DEVICE,IODIRA,0x09) #00001001
bus.write_byte_data(DEVICE,IODIRB,0x24) #00100100
detectList=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][0]=stringA[7]
detectList[1][0]=stringA[4]
detectList[2][0]=stringB[2]
detectList[3][0]=stringB[6]

bus.write_byte_data(DEVICE,IODIRA,0x09) #00001001
bus.write_byte_data(DEVICE,IODIRB,0x24) #00100100

bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
```

```
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][0]=stringA[7]
detectList[1][0]=stringA[4]
detectList[2][0]=stringB[2]
detectList[3][0]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x24) #00100100
bus.write_byte_data(DEVICE,OLATB,0x09) #00001001
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][1]=stringA[7]
detectList[1][1]=stringA[4]
detectList[2][1]=stringB[2]
detectList[3][1]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x12) #00010010
bus.write_byte_data(DEVICE,OLATB,0x12) #00010010
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][2]=stringA[7]
detectList[1][2]=stringA[4]
detectList[2][2]=stringB[2]
detectList[3][2]=stringB[5]

bus.write_byte_data(DEVICE,OLATA,0x36) #00110110
bus.write_byte_data(DEVICE,OLATB,0x1B) #00011011
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][3]=stringA[7]
detectList[1][3]=stringA[4]
detectList[2][3]=stringB[2]
detectList[3][3]=stringB[5]
```

```
matchNOR = True
for i in range(0,4):
    for j in range(0,4):
        if detectList[i][j] != correctNORList[i][j]:
            matchNOR = False
        if matchNOR == False:
            break
    if matchNOR == False:
        break
if matchNOR == True:
    print('The IC should be 74LS02 = NOR.')
    return

bus.write_byte_data(DEVICE,IODIRA,0x2A) #00101010
bus.write_byte_data(DEVICE,IODIRB,0x15) #00010101
detectList=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]

bus.write_byte_data(DEVICE,OLATA,0x00) #00000000
bus.write_byte_data(DEVICE,OLATB,0x00) #00000000
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][0]=stringA[6]
detectList[1][0]=stringA[4]
detectList[2][0]=stringA[2]
detectList[3][0]=stringB[3]
detectList[4][0]=stringB[5]
detectList[5][0]=stringB[7]

bus.write_byte_data(DEVICE,OLATA,0x15) #00010101
bus.write_byte_data(DEVICE,OLATB,0x2A) #00101010
longA = bus.read_byte_data(DEVICE,GPIOA)
longB = bus.read_byte_data(DEVICE,GPIOB)
stringA=""
stringB=""
for i in range(8):
    stringA += str(getTextSegment(longA, 8)[i])
for i in range(8):
    stringB += str(getTextSegment(longB, 8)[i])
detectList[0][1]=stringA[6]
detectList[1][1]=stringA[4]
detectList[2][1]=stringA[2]
detectList[3][1]=stringB[3]
detectList[4][1]=stringB[5]
detectList[5][1]=stringB[7]

matchNOT = True
for i in range(0,6):
    for j in range(0,2):
        if detectList[i][j] != correctNOTList[i][j]:
            matchNOT = False
```

```
            if matchNOT == False:
                break
        if matchNOT == False:
            break
    if matchNOT == True:
        print('The IC should be 74LS04 = NOT.')
        return

    print('IC not recognized.')

while True:
    userInput = raw_input("Enter \"0\" to exit, \"1\" to test breadboard,
\"2\" to test IC, \"3\" to detect IC, and \"4\" to detect circuit\n > ")
    if userInput == "0" or userInput.lower() == "exit" or userInput.lower()
== "end":
        break
    elif userInput == "1" or userInput.lower() == "breadboard":
        testBreadboard()
    elif userInput == "2" or userInput.lower() == "test":
        testIC()
    elif userInput == "3" or userInput.lower() == "detect":
        gateDetector()
    elif userInput == "4" or userInput.lower() == "circuit":
        testCircuit()
    else:
        print("invalid input: " + userInput)
```