Mini Retro Chip Tester

ABOUT

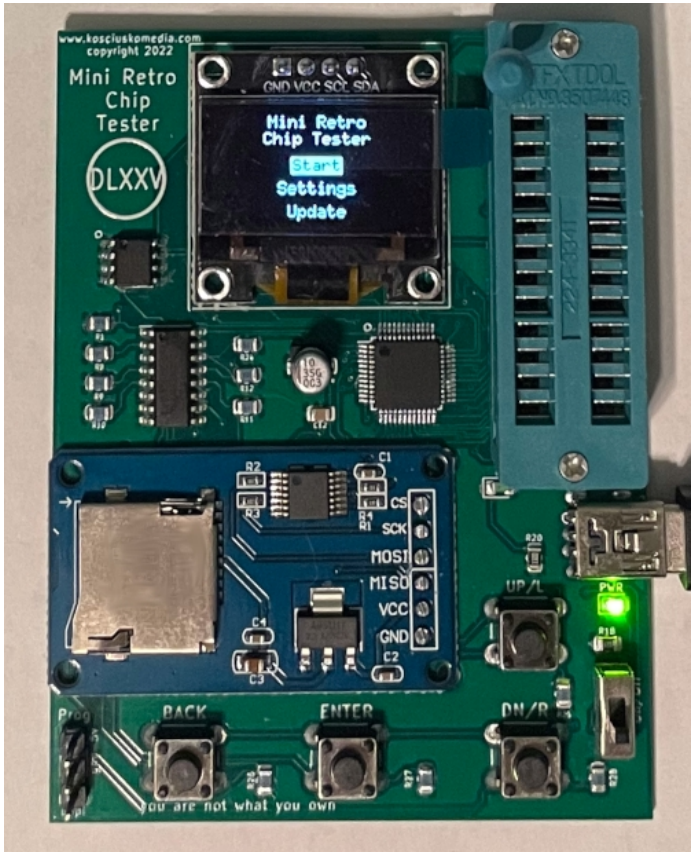
- The Mini Retro Chip Tester was first thought of a few years ago. Since that time I have been working on other projects and the tester off and on. Finally, I have a prototype. This tester will allow you to test all of the chips already on the tester and you can also update the chip database with other similar chips. However, because I have decided to make it more powerful, I will not be updating the chip database myself. The good news is, if you have a certain chip you'd like to add to the database it is fairly easy to do. With that, let's go over how this chip tester works.

Start Up

- When you first turn the tester on you will see the Kosciusko Media logo. It does not purposely stay up for a specific length of time. While this logo is showing all of the chips in the database are being loaded. Therefore, the more chips in the database, the longer this logo will show.

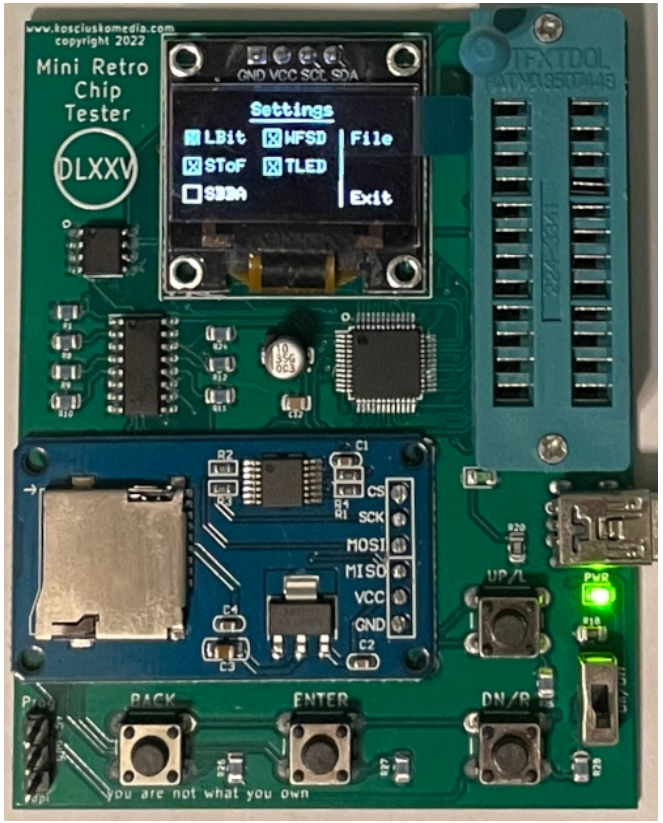
Menu Screen

- After all of the chips are loaded into memory, the menu screen will appear. You have the option to [Start], go to [Settings], or to [Update] the chips database. Let’s look at [Settings] first



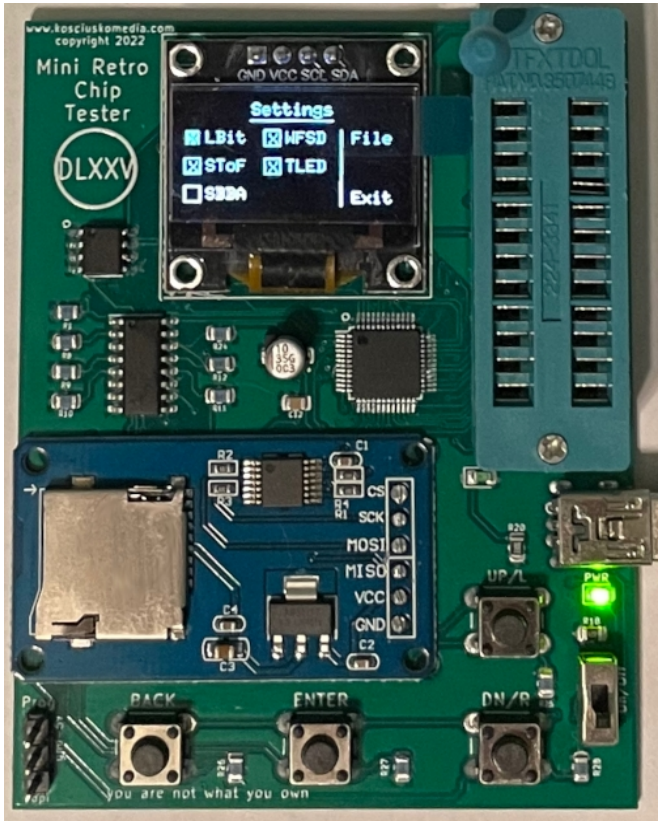
Settings

- The settings screen gives you a few options.
- Lbit – Uses only 0 and 15 when using a random test function on 4 bit RAM. Where as normal mode is to use 1 through 15. It reduces time while still putting either a 0 or a 1 in each bit.
- SToF – Stop on Fail. This will end the test automatically if any address fails.



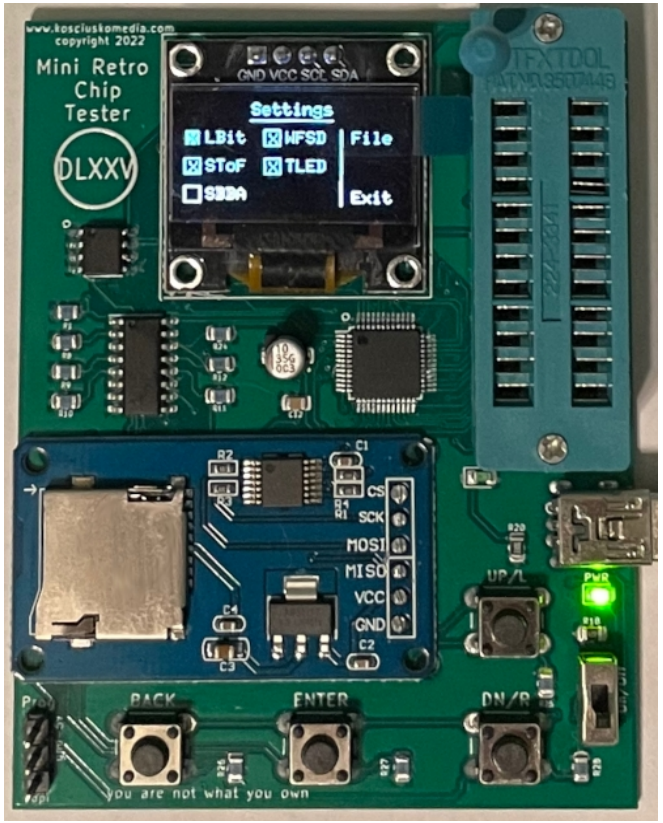
Settings

- SBBA – Show Bad Bit Address. This will stop the test and show you the column and row of a bad bit or word address. It will renew the test after 2 seconds unless you have SToF enabled.
- WFSD – Write Failures to SD. This will write the bad address of a RAM chip to an SD card based on the file name given when you select [File].



Settings

- TLED – Test LED. Turns LED that flashes during testing on or off.
- File – Clicking this will open a new screen to select the file name to save bad addresses to.
- Exit – This exits [Settings]



File Name

- To choose a file name open the [File] screen and use the [UP] button to choose a letter or a number. Up will scroll from A to 9 and then restart at A. Once you have a letter selected, push the [DN/Right] button to move to the next character slot. Once you have chosen a file name move to [SAVE] and click the [Enter] button on the device. You can then exit.

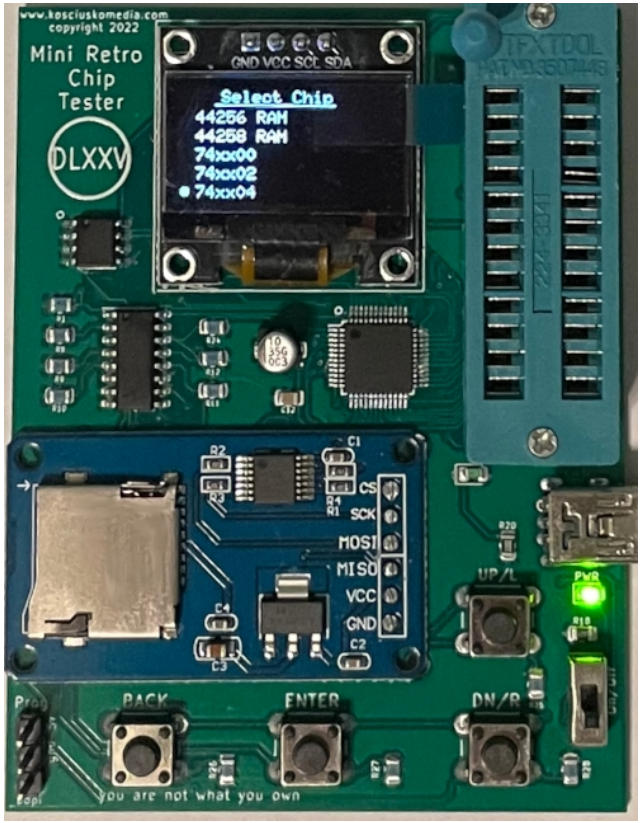
- **Please note one of the reasons this project is being redesigned is because of the low power. Sometimes the SD reader will not work. There is nothing to be done if it doesn't. That is why this tester is cheaper than originally planned. If you must have an SD save function or the ability to update the chips via SD then this will not be the tester for you.

Update

- This was originally planned as a way to update the chip database via SD. Unfortunately, this design has a flaw. It should have been designed with more power in mind. That being the case, some times the SD reader will not work, though it often does. So, if you want to update the chip database there is a program you can use with the Arduino IDE and an Arduino Nano to update the database in that manner. Instructions are further in this document.

Start Testing

- When you select [Start] a new screen will open that shows all of the chips in the data base. Use the Up and Down button on the device to choose which chip you want to test. Do not use the wrong chip, although you are unlikely to cause any damage because most of each chip’s pins are made for 5v you might send voltage to a pin that can’t accept it. So be aware.



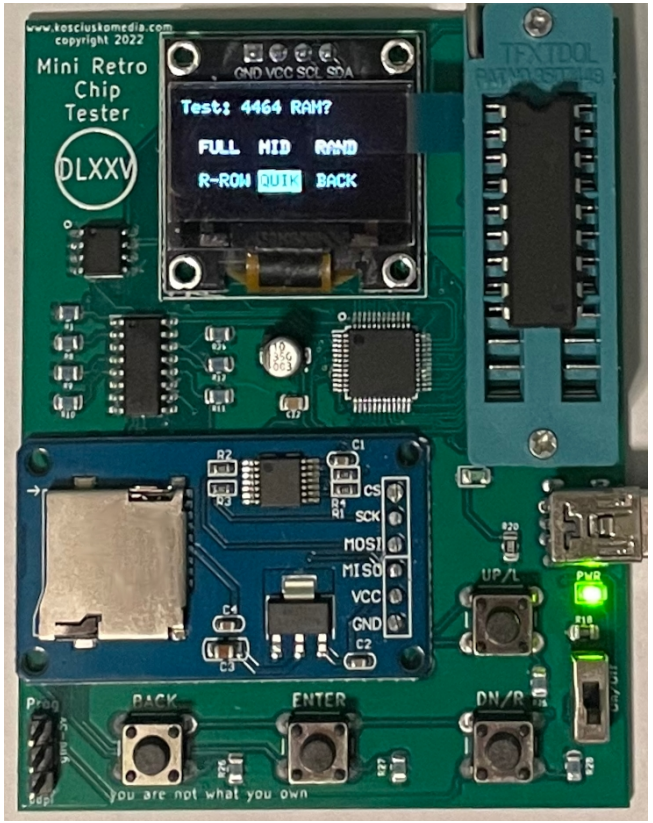
Test Select

- This section applies to RAM only. After you have selected the chip you want to test a new screen will show that allows you to select the type of test you want to run. One thing to keep in mind is that using faster tests won’t make any difference with 1 bit RAM chips as each ram address can only be either a zero or one. The tester will test each one of these individually. If you are using 4 bit RAM then selecting different types of tests will make the tests faster as you can either test each address by counting from 1 to 15, counting each of the four bits individually, or simply testing the address for either 0 or 15.

Test Select

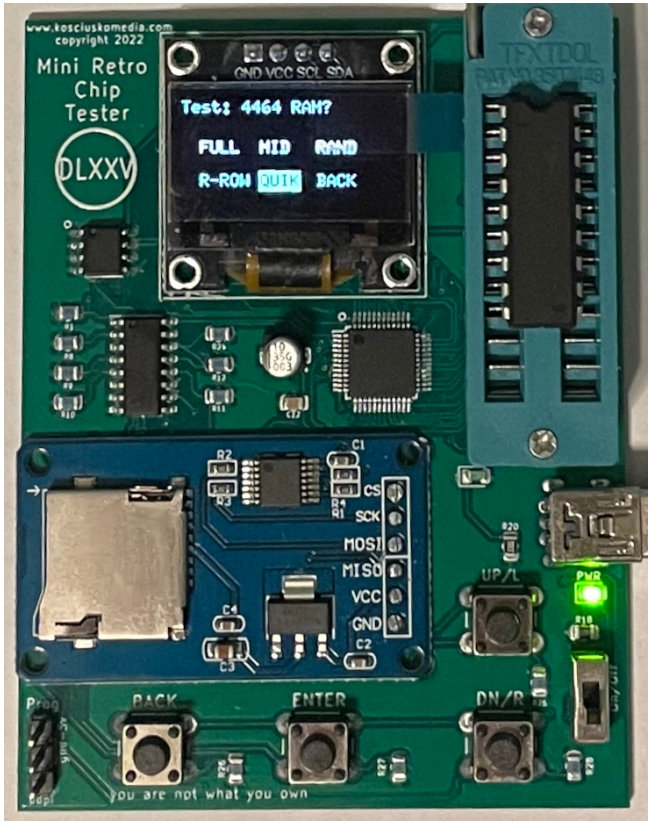
The following are the RAM test types.

- Full – This will count and test 0-15 at each address (4 bit).
- Med – This will test for a 0 and 1 at each address (4 bit).
- Rand – This will test a random address (column and row) as many times as there are columns



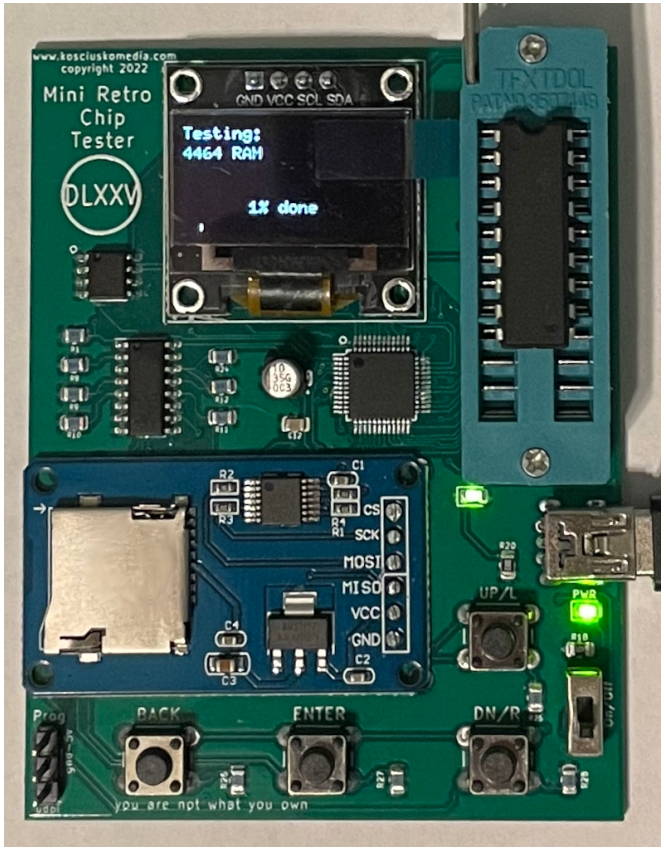
Test Select

- R-Row – This will test start with column zero and go to the last column and test a random row each time.
- Quik – This will test each address with either 0 or 15. The fastest non-random test. (4 bit)



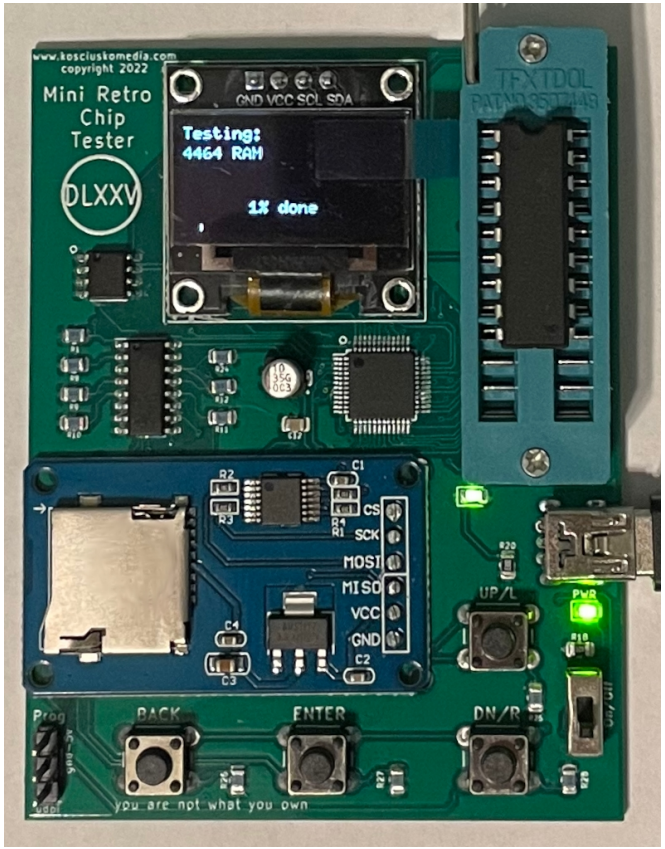
Testing

- During the test each address will be tested for either a 0 or 1 based on the RAM type and the test selected. Some larger RAM chips can take quite a while to finish testing. This is another limitation with this design and future designs will use a different micro-processor but at a higher cost.



Testing

When the test has finished it will show whether the chip was good or how many bad addresses there were based on your settings. If you have SToF checked, then the test will end as soon as it fails. You can also stop the test at any time by clicking the back button on your device.



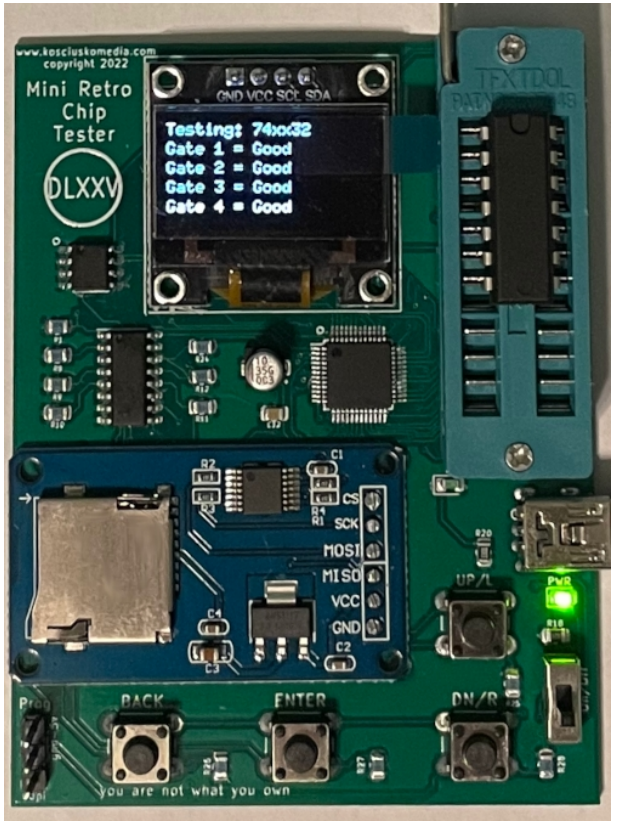
7400 Series Testing

You can test several 7400 series chips as well. You can also add others if they are similar in design. As of now, you can test NAND, NOR, AND, OR, XOR, Inverters, Parallel-In shift registers and Serial-In shift registers. The complete list of testable chips is at the end of this document.

7400 Series Testing

Testing logic gates.

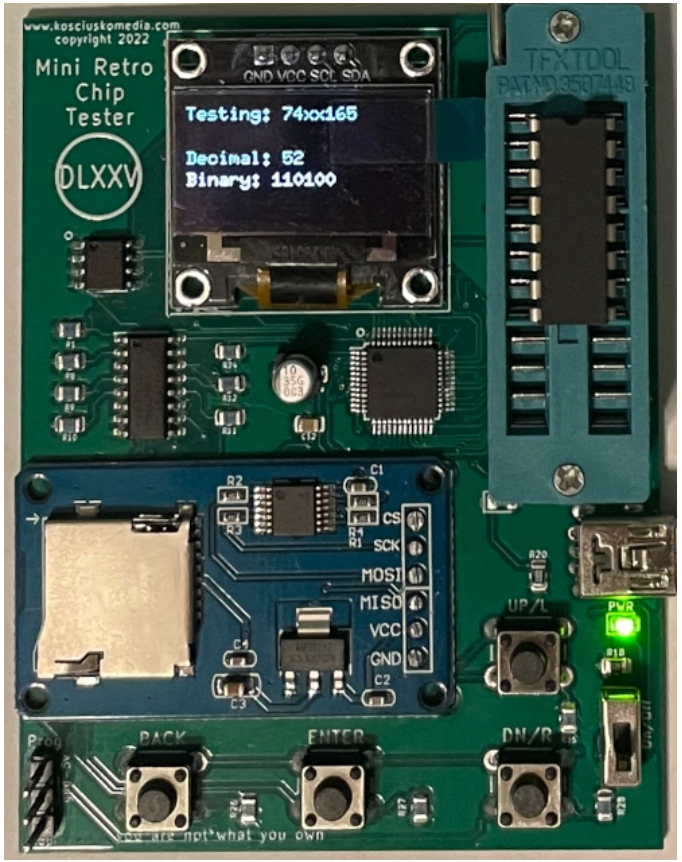
When you test a logic gate there is no “test over” screen. It simply shows you if each gate is functioning properly as it goes along. It gives you time to see the results as the test is being conducted and once the test is finished. Once done, you will return to the Menu Screen.



7400 Series Testing

Testing shift registers

When you test a shift register it will be tested based on the size of the bus. If the bus size is 8 then the test will count from 0 to 255 testing each input and output pin along the way. Once this test is done you will see the pass or fail screen.



Updating

Because these boards were a trial and there are plans to redesign them, I will not be doing any updates myself. They are cheaper than originally planned because of that. Also, I am including the code to download and also a program to add new chips to the eeprom located on the pcb. You will need to reflash the testing program after you update the eeprom. You are also welcome to change the code of the main program to your liking.

Updating

First we will go over how to update in general then we will go over how the chip database is laid out so that you can add chips to it if you'd like. **Note: the chips will have to follow certain types of layouts to be able to be added and tested.

- First – You need an Arduino Nano. They are quite cheap at around $4-$5.
- You also need the Arduino IDE software. This is free and easy to find online. Just download that and install.
- You will also need an add-on for the Arduino IDE called MegaCoreX.

Updating

To get MegaCoreX you can simply go to Tools/Manage Libraries within Arduino and search for MegaCoreX. Once you have found it, install it.

Next you will need to install jtag2updi. Which can be found HERE This will be installed onto your Arduino Nano to make it a UPDI programmer. Simply go to where you downloaded jtag2updi and open the Arduino file (.ino). Once you have opened it plug your Nano into your computer and press the upload button. This will install jtag2updi onto your Nano.

Updating

Now you can open up the eeprom chip updater or the main program to install them onto your tester. If you are updating the eeprom chip data base you will have to reinstall the main program afterwards. But before you can update the tester you need to change some settings in the Arduino IDE.

When updating the tester you will need to go to Tools/boards/MegaCoreX and select Atmega4809. After you have done that change [CLOCK] to 20Mhz and change the [Programmer] to jtag2updi.

Updating

There is only one more step. You will now connect a wire from the Arduino 5v pin to the 5v pin on the bottom left of the tester (the top pin of the three). You will also need to add a wire from the arduino GND pin to the middle pin of the bottom three on the tester. Finally, you need to connect a wire from pin D6 on the Nano to the bottom pin of the three on the bottom of the tester. This will allow you to upload the code from the Arduino IDE and Nano to the chip on the tester board. With that done you can cane open up one of the files either the eeprom updater or the main program and click on upload and the tester will be updated.

Updating

One more thing. As stated earlier, this board is lacking in power, which is one of the reasons it is being redesigned. If you have the Nano connected to the tester correctly and the OLED doesn’t work or it won’t update, you can plug another cable into the tester’s usb to give it more power. This will not hurt either the tester or the Nano as this is still delivering 5v it is just adding to the ability to use more current. It is not a given that you will have to do this but you might need to.

Adding Chips

The chip database on this tester is set up in a very specific way. First, each chip has 48 bytes worth of data. That data will start at a specific address in the eeprom. Because the address for each chip's data could start at a number higher than 255 we need to use two bytes on the eeprom to store the address of each chip's data. The first two bytes of the eeprom stores the address of the first chip's data. The next two store the address of the second chip's data and so on. You do not need to worry about this as the program will automatically decide where each chip's data will go. This is just here to give you an idea of how it functions.

Adding Chips

To store this data, the program needs to know how many chips there are so whenever you add a chip you need to make sure all of the following changes are made to the code before you upload it (or flash it) to the chip tester.

Adding Chips

This is the database chip code. Let's look at how it functions.

```
int tmpNumChips = 17;
byte tmpNameLen[17]{0x08, 0x08, 0x09, 0x09, 0x09, 0x09, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x07, 0x07, 0x07};
char tmpName[17][20]{{"2114 RAM"}, {"4464 RAM"}, {"41256 RAM"}, {"41257 RAM"}, {"44256 RAM"}, {"44258 RAM"}, {"74xx00"},
{"74xx02"}, {"74xx04"}, {"74xx06"}, {"74xx08"}, {"74xx14"}, {"74xx32"}, {"74xx86"}, {"74xx165"}, {"74xx373"}, {"74xx595"}};
byte tmpType[17]{0x03, 0x02, 0x01, 0x01, 0x02, 0x02, 0x06, 0x07, 0x04, 0x04, 0x08, 0x04, 0x09, 0xA, 0xB, 0xC, 0x05};
byte tmpPins[17]{0x12, 0x12, 0x10, 0x10, 0x14, 0x14, 0xE, 0xE, 0xE, 0xE, 0xE, 0xE, 0xE, 0xE, 0x10, 0x14, 0x10};
byte tmpBusSize[17]{0xA, 0x08, 0x09, 0x09, 0x09, 0x09, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08};

//0x04 - inverter, 0x05 - s-in shift register, 0x06 NAND, 0x07 NOR, 0x08 = AND, 0x09 OR, 0xA XOR, 0xB p-in shift register, 0xC - 8 bit flip flop
// type 1 = 1bit, 2 = 4 bit, 3 = 2114

// 1 = 5v, 2 = gnd, 3 = 12v, 4 = !cs, 5  = !we, 6 = !CAS, 7 = !RAS, 8 = D_in, 9 = D_out, 10 = !output enable
 //11 = DQ1, 12 = DQ2, 13 = DQ3, 14 = DQ4, 15 = NC, 16 = D0, 20 = A0, 21 = A1, 22 = A2, 23 = A3, 24 = A4, 25 = A5, 26 = A6, 27 = A7, 28 = A8, 29 = A9
 //30 = A1, 31 = Y1, 32 = A2, 33 = Y2, 34 = A3, 35 = Y3, 36 = A4, 37 = Y4, 38 = A5, 39 = Y5, 40 = A6, 41 = Y6,
 //42 = A, 43 = B, 44 = C, 45 = D, 46 = E, 47 = F, 48 = G, 49 = H, 50 = SER in, 51 = RCLK, SRCLK = 52, 53 = !SRCLR, 54 = ser out, 55 = SH!LD, 56 = CLK,
 //57 = CLK INH, 58 = !ser out,
 //
 //60 = A1, 61 = B1, 62 = Y1, 63 = A2, 64 = B2, 65 = Y2, 66 = A3, 67 = B3, 68 = Y3, 69 = A4, 70 = B4, 71 = Y4 -- 2 input gates
 //80 = D0, 81 = D1, 82 = D2, 83 = D3, 84 = D4, 85 = D5, 86 = D6, 87 = D7, 88 = O0, 89 = O1, 90 = O2, 91 = O3, 92 = O4, 93 = O5, 94 = O6
 //95 = O7, 96 = Latch Enable

byte tmpPinFunc[17][24]{{0x1A, 0x19, 0x18, 0x17, 0x14, 0x15, 0x16, 0x06, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0xE, 0xD, 0xC, 0xB, 0x1D, 0x1C, 0x1B, 0x01},
//{0xF, 0x08, 0x05, 0x07, 0x14, 0x16, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1B, 0x19, 0x18, 0x17, 0x1A, 0x09, 0x06, 0x02},
//{0xF, 0x08, 0x05, 0x07, 0x14, 0x16, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1B, 0x19, 0x18, 0x17, 0x1A, 0x09, 0x06, 0x02},
{0xA, 0xB, 0xC, 0x05, 0x06, 0x1A, 0x19, 0x18, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1B, 0x17, 0x16, 0x15, 0x14, 0xD, 0x06, 0xE, 0x02},
{0x1C, 0x08, 0x05, 0x07, 0x14, 0x16, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1B, 0x19, 0x18, 0x17, 0x1A, 0x09, 0x06, 0x02},
{0x1C, 0x08, 0x05, 0x07, 0x14, 0x16, 0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1B, 0x19, 0x18, 0x17, 0x1A, 0x09, 0x06, 0x02},
{0xB, 0xC, 0x05, 0x07, 0xF, 0x14, 0x15, 0x16, 0x17, 0x01, 0x00, 0x00, 0x00, 0x00, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0xA, 0x06, 0xD, 0xE, 0x02},
{0xB, 0xC, 0x05, 0x07, 0xF, 0x14, 0x15, 0x16, 0x17, 0x01, 0x00, 0x00, 0x00, 0x00, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0xA, 0x06, 0xD, 0xE, 0x02},
{0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x44, 0x42, 0x43, 0x47, 0x45, 0x46, 0x01}, //74xx00
{0x3E, 0x3C, 0x3D, 0x41, 0x3F, 0x40, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x01}, //74xx02
{0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x25, 0x24, 0x27, 0x26, 0x29, 0x28, 0x01}, //74xx04
{0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x25, 0x24, 0x27, 0x26, 0x29, 0x28, 0x01}, //74xx06
{0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x44, 0x42, 0x43, 0x47, 0x45, 0x46, 0x01}, //74xx08
{0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x25, 0x24, 0x27, 0x26, 0x29, 0x28, 0x01}, //74xx14
{0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x44, 0x42, 0x43, 0x47, 0x45, 0x46, 0x01}, //74xx32
{0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x44, 0x42, 0x43, 0x47, 0x45, 0x46, 0x01}, //74xx86
{0x37, 0x38, 0x2E, 0x2F, 0x30, 0x31, 0x3A, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0x32, 0x2A, 0x2B, 0x2C, 0x2D, 0x39, 0x01}, //74xx165
{0xA, 0x58, 0x50, 0x51, 0x59, 0x5A, 0x52, 0x53, 0x5B, 0x02, 0x00, 0x00, 0x00, 0x00, 0x60, 0x5C, 0x54, 0x55, 0x5D, 0x5E, 0x56, 0x57, 0x5F, 0x01}, //74HC373
```

Adding Chips

- Int tmpNumChips = 17 – This is how many chips there are total. If you add a chip then this will be changed to 18.
- TmpNameLen[17] = { 0x08, etc….}; - This tells the program how many characters the chip's name is. The maximum is 20. You can use Hex as I have done here or you can simply use a decimal number. This is an array, thus, you have to change the number 17 to 18 if you add a chip and so on.
- Char tmpName[17][20] = {"2114 RAM", etc..}; - This is the actual name of the chip. Remember that the nameLen above needs to match the number of characters in this name, including spaces.

Adding Chips

This is a 2 dimensional array. The [20] is just there to show the number of characters there could be in each name. The [17] shows how many total names there are. If you add a chip you will need to change 17 to 18 and so on.

- TmpType[17] = { 0x03, etc...}; - This is the type of chip. Each chip type is a different number. For instance, 0x03 (or 3) is a 2114 RAM chip. It needed it's own special number because it is different than most other chips. 1 is 1 bit RAM, 2 is 4 bit RAM etc. The list is commented out underneath these variables.

Adding Chips

- TmpPins[17] = { 0x12, etc...}; - This is the number of physical pins on the chip. Here it is written in Hex 0x12 = 18. You can use decimal numbers if you prefer. Remember again, to change the [17] to [18] if you add a chip and so on.
- TmpBusSize[17] = {0x08, etc...}; - This is only needed for RAM chips. It tells the tester how many columns and rows each chip has. Most chips will have an equal number of columns and rows (looking at you 2114 ><) So, just look at your datasheet and see how many pins say A0, A1, etc. etc. That should mean it is an address pin.

Adding Chips

- TmpPinFunc[17][24] = { 0x1A, etc...}; - This is also a two dimensional array. The [24] tells use the number of pins (or max number of pins in reality). The 17 is the number of chips we have.

  The way this is setup it goes from pin 1 to the last pin. That last pin could be pin 16, 18, 14. It doesn't matter, we need to add all chips in the same manner. Each pin function has a number assigned to it. See the appendix or the comment in the code to see what each number represents. In order to tell the tester what each pin does we need to give them in order but not when we are talking about the right side of the chip.

Adding Chips

For example: If we have a 14 pin chip we will have 7 pins on the left side. It will go from pin 1 to pin 7 (or 0 to 6 in our code). We need to look at the data sheet for the chip we want to add and see what each pin does. Let's say for instance, pin 1 is GND. Well, GND is 0x02 so we'd put 0x02 in the first spot like this {0x02, ..}

then we'd add the code for pin 2 after the comma. After pin 7 though we will stop and add 0x00 all the way up until there are only 7 spots left out of the 24 then we would start at pin 8 and complete the list of functions. That is why for each pin function array you will see multiple 0x00 in the middle. These are pins unused from the ZIF socket.

Adding Chips

So, the last pin of our chip will be pin 24 of the ZIF socket. Therefore we acknowledge that by making function 24 equal our last pin function and so forth.

It’s not as complicated as it might seem but this is just here in case you’d like to add another or more chips. There are still 17 chips it tests from the outset.

Current Chips

- 2114 RAM
- 4464 RAM
- 41256 RAM
- 41257 RAM
- 44256 RAM
- 44258 RAM
- 74xx00
- 74xx02

Current Chips

- 74xx04
- 74xx06
- 74xx08
- 74xx14
- 74xx32
- 74xx86
- 74xx165
- 74xx373

Current Chips

- 74xx595

Chip Type Codes

- 0x01 – 1 bit RAM
- 0x02 – 4 bit RAM
- 0x03 – 2114 RAM
- 0x04 – Hex Inverter
- 0x05 – Serial In Shift Register
- 0x06 – NAND Gate
- 0x07 – NOR Gate
- 0x08 – AND Gate

Chip Type Codes

- 0x09 – OR Gate
- 0x0A – XOR Gate
- 0x0B – Parallel In Shift Register
- 0x0C – Flip Flop

Pin Function Codes

- 1 – VCC
- 2 – GND
- 4 – !CS
- 5 – !WE
- 6 - !CAS
- 7 - !RAS
- 8 – Data In
- 9 – Data Out

Pin Function Codes

- 10 – !OE
- 11 – DQ1
- 12 – DQ2
- 13 – DQ3
- 14 – DQ4
- 15 – N/C
- 20 – A0
- 21 – A1

Pin Function Codes

- 22 – A2
- 23 – A3
- 24 – A4
- 25 – A5
- 26 – A6
- 27 – A7
- 28 – A8
- 29 – A9

Pin Function Codes

- 30 – A1 (Inverter)
- 31 – Y1 (Inverter)
- 32 – A2 (Inverter)
- 33 – Y2 (Inverter)
- 34 – A3 (Inverter)
- 35 – Y3 (Inverter)
- 36 – A4 (Inverter)
- 37 – Y4 (Inverter)

Pin Function Codes

- 38 – A5 (Inverter)
- 39 – Y5 (Inverter)
- 40 – A6 (Inverter)
- 41 – Y6 (Inverter)
- 42 – A (Shift Register)
- 43 – B (Shift Register)
- 44 – C (Shift Register)
- 45 – D (Shift Register)

Pin Function Codes

- 46 – E (Shift Register)
- 47 – F (Shift Register)
- 48 – G (Shift Register)
- 49 – H (Shift Register)
- 50 – Serial In
- 51 – RCLK
- 52 - SRCLK

Pin Function Codes

- 53 – !SRCLR (Shift Register)
- 54 – Serial Out (Shift Register)
- 55 – SH!LD (Shift Register)
- 56 – CLK (Shift Register)
- 57 – CLK INH (Shift Register)
- 58 – !Serial Out
- 60 – A1 (Logic Gate)
- 61 – B1 (Logic Gate)

Pin Function Codes

- 62 – Y1 (Logic Gate)
- 63 – A2 (Logic Gate)
- 64 – B2 (Logic Gate)
- 65 – Y2 (Logic Gate)
- 66 – A3 (Logic Gate)
- 67 – B3 (Logic Gate)
- 68 – Y3 (Logic Gate)
- 69 – A4 (Logic Gate)

Pin Function Codes

- 70 – B4 (Logic Gate)
- 71 – Y4 (Logic Gate)
- 80 – D0 (Latch)
- 81 – D1 (Latch)
- 82 – D2 (Latch)
- 83 – D3 (Latch)
- 84 – D4 (Latch)
- 85 – D5 (Latch)

Pin Function Codes

- 86 – D6 (Latch)
- 87 – D7 (Latch)
- 88 – O0 (Latch)
- 89 – O1 (Latch)
- 90 – O2 (Latch)
- 91 – O3 (Latch)
- 92 – O4 (Latch)
- 93 – O5 (Latch)

Pin Function Codes

- 94 – O6 (Latch)
- 95 – O7 (Latch)
- 96 – Latch Enable (Latch)