

SERIAL EEPROM TUTORIAL - 93C56

The purpose of this tutorial is to acquaint you with the operation of the EPROM+ programming system when used with a serial eeprom. Serial eeproms are found in a variety of electronic products and modules including printer cartridges and automobile body controller modules. This tutorial requires the use of an actual part, the 93C56 serial eeprom. As the tutorial progresses you will perform actual reading and programming on the contents of a real part. This is exactly the same as if the part was attached to the circuit board of an actual product.

GETTING STARTED

This tutorial requires the following items: 1 - EPROM+ programming system programming unit (#AR-32A), 1 - Serial eeprom adapter (#ASERSM1) with a clip or probe set which connects to the 93C56 part. Note: If you are using the ASEREE2 adapter use a 93C56 part in a dip package inserted far left in the adapter socket.

Before we begin confirm that the EPROM+ system is connected to a host computer and powered on. Be sure that the ASERSM1 adapter is installed in the AR-32A socket fully left justified. Confirm that the clip/probe cable is securely attached to the left group of header pins (93XX) with the brown wire on the right. The adapter switch can be set to either the LV or 5V setting. The 4 position dip switch on the adapter should be set to all open.

Start the software, wait for the program to load and display the DEVICE SELECTION TABLE. At the bottom left you will see ENTER DEVICE TYPE ->. Type 93C56 and press ENTER. This configures the system to work with the 93C56 part.

MAIN SCREEN

You are now viewing the main command screen. Note that after you select the part, the system displays the switch setting for the programming unit (lower left) plus information about the adapter. Also note that the position of the part or in this case the plug is also indicated. You may recall this information whenever you wish by pressing the "S" key at the SELECT COMMAND prompt. Press any key to clear this information. The system now displays SELECT COMMAND ->. For every high level system command, this is where you start. Commands are selected by pressing a key, either a number or letter. Before we proceed to actual exercises let's start with a simple example; the buffer editor. Since you will be doing most of your work in the buffer editor choose this mode now. Note that the buffer editor is command 5 in the list.

| ANDROMEDA RESEARCH LABS EPROM+ PROGRAMMING SYSTEM - V5.9D | | |
|---------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| DEVICE = 93C56 (PRESS S KEY) | DEVICE SIZE = 0 - FF | Upp = 5.00 |
| COMMAND LIST | | |
| 0 - EXIT PROGRAM/SYSTEM OPTIONS | A - SAVE BUFFER TO DISK FILE | |
| 1 - PROGRAM DEVICE FROM DISK FILE | B - SAVE DEVICE TO DISK FILE | |
| 2 - PROGRAM DEVICE FROM BUFFER | D - DIRECTORY OF FILES | |
| 3 - READ DISK INTO BUFFER | F - DISPLAY/SELECT PROG ALGORITHM | |
| 4 - READ DISK FILE INTO BUFFER | K - DEVICE/FILE CHECKSUM | |
| 5 - BUFFER EDITOR | P - SET DRIVE/PATH | |
| 6 - VERIFY DEVICE IS ERASED | R - REMOVE/DELETE FILE | |
| 7 - COPY DEVICE | S - DIP SWITCH SETTINGS | |
| 8 - COMPARE DEVICE WITH BUFFER | U - MANUALLY SET PROGRAMMING VOLTAGE | |
| 9 - CHANGE DEVICE TYPE | Z - DEVICE OPTIONS | |
| COMMAND/DATA ENTRY | | ACTIVITY/STATUS |
| PROGRAMMING UNIT SWITCH ┌─── DIP SWITCH ──┐ 1 2 3 4 5 6 7 8 └───┴───┴───┴───┴───┴───┴───┴───┘ 1 1 1 1 1 1 1 1 | | REQUIRES ADAPTER (#ASEREE2 OR #ASERSM1) ┌─── DIP SWITCH ──┐ 1 2 3 4 └───┴───┴───┴───┘ 1 1 1 1 (DEVICE POSITION LEFT) |
| PRESS ANY KEY TO CONTINUE | | |

Press the "5" key. You are now viewing the buffer editor screen. The buffer editor is a separate operating mode of the EPROM+ software. It is intended to provide a tool kit of functions which allow you to work directly with data at the byte level.

ABOUT THE BUFFER

The buffer is a block of memory bytes used by the EPROM+ system software to hold data. Although vital to the operation of the EPROM+ system, the buffer is very easy to understand. You can think of the buffer as a group of post office boxes. Just as each post office box holds mail, each buffer location holds one byte of information. Just as each post office box represents a specific mail address each buffer location is also defined by a specific number called an address. Look at the buffer editor screen on your computer. The screen shows 256 bytes of data. This is the big block of FF's in the middle of the screen.

| BUFFER EDITOR | | |
|---------------------------------------------------|-------------------------------------------------|----------------------|
| EDITOR MODE = BYTE | BUFFER END = FFFFFFFF | DEVICE SIZE = 0 - FF |
| 00 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 10 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 20 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 30 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 40 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 50 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 60 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 70 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 80 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| 90 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| A0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| B0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| C0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| D0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| E0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| F0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | |
| ENTER EDITOR COMMAND ("?" FOR HELP/"ESC" TO EXIT) | | |

We are now going to jump right in and use the editor MODIFY command to illustrate the concept of addresses and bytes. First look at the lower left corner of the editor screen. It says ENTER EDITOR COMMAND. Press "M" and note that the lower corner now says MODIFY BUFFER (ESC TO EXIT). ESC is the "Escape" key usually labeled "ESC". Just to make the point press the "ESC" key and note that the ENTER EDITOR COMMAND prompt returns. This is important, pressing ESC always exits to the previous command level. Press "M" again then press "0" (zero not the letter O) then press the ENTER key. What you have done is invoked the "M"odify command. The modify command lets you modify the contents of the buffer. Since the buffer editor has to know which address in the data you wish to modify you followed the M with 0 which is buffer address 0. 0 is the very first address in the buffer. You then pressed ENTER which tells the editor to execute the command.

Notice that the editor screen has changed. In the upper left corner (edit window) there is a turquoise 0 followed by a white FF- and a cursor. The 0 is the address, the FF is the data byte contained at that address. The cursor is waiting for you to enter a new data byte which will replace the FF at buffer address 0. Now look at the first byte (FF) at the far left side of the first line. Notice that a block cursor appears over the FF. This first position in the first line is address 0 of the 256 byte screen block. The turquoise numbers in the far left column indicate the starting address of the first byte in that specific row. Press the down arrow key. Note that the block cursor moves down one line and the edit window address has changed from 0 to 10. Press the up arrow key to move the block cursor back to address 0. Now press the right arrow key. The block cursor moves one byte to the right and the address changes to 1. Using the arrow keys move the block cursor about in the 256 byte window to become familiar with block cursor movement and the byte addresses. Note that wherever you move the block cursor the edit window shows the current buffer address.

CHANGING DATA

Now that you are familiar with moving about in the buffer data using the arrow keys, move the block cursor to address 80. Press 55. Note that as you enter the 55 it appears in the edit window and replaces the FF after the second 5 is pressed. The block cursor then moves to address 81. Press AA. AA has replaced the FF in address 81. Now use the arrow keys and move the block cursor back to address 80. Note that the edit window now shows the 55. Now move the block cursor to address 82 and press C9. You have now replaced three bytes of buffer data which were initially FF and changed each to a different value (55, AA, C9). You also may have noticed that on the far right of the 256 byte block is a smaller block of 256 dots or periods. Note that there is a single dot for each byte of FF and there is also a single block cursor which moves with the main data block cursor. Also note that on the same line as address 80 where the data is 55 there is a corresponding "U" character in the smaller block. This is the ASCII display block and is covered on next page.

ENTRY BASE = HEX - What does it mean?

If you already understand the hexadecimal number system, ASCII and binary you can skip this part of the tutorial. If not, this will provide the foundation for most of your future work. All computers, whether large or small, work with binary numbers. Binary means two so a binary digit (called a bit) can only have one of two values, 0 or 1. A group of eight bits is called a **byte** and a group of four bits is called a **nibble** while sixteen bits is called a **word**. Remember that each group of FF represents a single byte. That means that FF must represent eight bits. The eight bits represented by FF is 11111111 (eight 1's). So why is FF equal to 11111111? The reason is that FF is a different number base. Humans normally use the decimal (base 10) number system. This number system has ten unique characters which are used to represent numbers. The characters are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. There is no single character for ten so we use two characters to represent 10. The same applies to the hexadecimal number system (base 16). The only difference is that you need sixteen characters instead of ten. To address the extra characters requires we use the first six letters of the alphabet (A, B, C, D, E and F). Therefore base 16, which is also known as HEX, uses these sixteen characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The letters A, B, C, D, E and F represent the decimal numbers 10, 11, 12, 13, 14 and 15.

So what does this have to do with binary? Well if you want to show the number nine in decimal you only need to use one character; 9. However if you want to show the number 10 you will need two characters, 1 and 0. If you want to show the number 100 you will need three characters. Remember that each character position is a place holder. For 10 the 1 means one in the tens place and the 0 means 0 in the ones place. For 100 the 1 means 1 in the hundreds place, 0 in the tens place and 0 in the ones place. Each position is a place holder. Each decimal place holder is ten times greater than the preceding place holder. Binary works the same way except that each place holder is two times greater than the preceding place holder. Lets look at the binary number 1101. Just like a decimal number the digit on the far right is the least significant digit also known as the least significant bit or LSB. The digit on the far left is the most significant bit or MSB. Each character is a place holder but unlike decimal (base 10) where each place holder is a multiple of 10 (1000, 100, 10, etc.) with binary each place holder is a multiple of 2. Let's use the decimal example 245. This number represents 2×100 plus 4×10 plus 5×1 . Remember in the binary number system you only have two characters, 0 and 1 therefore the place holder values are smaller. For a four digit binary number beginning with the MSB the place holder values are 8, 4, 2 and 1. Therefore the binary number 1101 represents 1×8 plus 1×4 plus 0×2 plus 1. If you add up the result $(8+4+0+1)$ the decimal answer is 13 however the hexadecimal answer is the letter D. Remember in HEX we represent numbers greater than 9 using letters. So why do we use HEX? It is the easiest way to represent four bit binary numbers. In other words you can represent one binary byte (8 bits) with two hex characters. Here's an example. Assume you have an eight bit (one byte) binary number which is 11001001. If you speak the number in binary you have to say one, one, zero, zero, one, zero, zero, one. However if you speak the number in hex you simply say C9. Just to make it clear lets break it apart. The first four bits of the number are 1100. This is $8+4+0+0$ which is 12 decimal. In hex 12 decimal is the letter C. The second four bits of the number are 1001. This is $8+0+0+1$ which is 9 decimal and is the same in hex. Let's take a 16 bit binary number and convert it to hex. The number is 1011001100011111. First break it into groups of 4 bits. 1011 - 0011 - 0001 - 1111. Now use the binary bit value for each place holder of each group of four. 1011 is $8+0+2+1=11$ decimal or B in hex. 0011 is $0+0+2+1=3$ in decimal and hex. 0001 is $0+0+0+1=1$ decimal and hex. 1111 is $8+4+2+1=15$ decimal or F in hex. Now when you combine the hex characters the binary number 1011001100011111 is B31F in hex. Sixteen binary bits compressed to four hex characters.

At this point you should understand HEX and why it has become the standard way to represent binary number in a compact, human readable form. The next piece of information which is required to build your foundation is to understand ASCII (pronounced “ASK KEY”). ASCII is the abbreviation for “American Standard Code for Information Interchange”.

ASCII - A BRIEF HISTORY LESSON

In the early days of computers, late 1950’s to mid 1960’s, companies that manufactured computers also manufactured the printers and display terminals that operators used to interface to the machines. Although the display terminals and printers would display english letters, numbers and punctuation, you could not plug a display terminal manufactured by General Electric into a computer manufactured by UNIVAC. The machines and the peripherals all had to come from the same manufacturer in order for the computer system to work. Since all computers work in binary, the engineers at the computer company would decide which binary number would represent a corresponding english letter. For example let’s say that the General Electric engineer decided that the binary number 10010101 (95 hex) would represent an uppercase “A”. Whenever this number was received by a display terminal an A would appear on the screen. The UNIVAC engineer could choose any binary number he wanted to use when a UNIVAC terminal was sent the letter A. There was no standard for the codes that would be used to display english characters on a display terminal screen or printer. This was not a problem until computers began to be linked together. At that point it became clear that a standard format was needed to allow the exchange of information between machines as well as the peripherals. The result was the creation of the ASCII standard. ASCII defines which english characters are represented by which binary numbers. Here are a few examples of english characters and their corresponding ASCII codes. A=41 hex or 01000001 in binary. B=42 hex or 01000010 in binary. All english characters, both upper and lower case, numbers and punctuation are defined. Once the ASCII codes were defined and integrated into the computer hardware and software, any display terminal or printer could be used with any computer.

THE ASCII BLOCK

Back to the tutorial. At address 80 through 82 you have the hex codes 55, AA and C9. If you look at the lower right corner you will see “ENTRY BASE = HEX”. This tells you that the number base you are using to enter data into the buffer is HEX. Now press the TAB key. Note that the entry base changes to ASCII. Using the arrow keys move the block cursor to address 90. Now type these letters - ASCII. Note that in the main data block the hex codes 41, 53, 43, 49 and 49 appeared while in the ASCII block the letters ASCII appeared. This is because with the entry mode set to ASCII you can directly enter the ASCII character codes into the buffer simply by pressing the desired key. **NOTE:** The ASCII area will only display printable characters. If the hex code does not correspond to a printable character the location in the ASCII block will display a dot or period (.). This is why the entire ASCII block is filled with dots when the software starts. The system initializes the buffer data to FF and there is no printable character represented by that hex code.

At this point you should be familiar with the buffer, buffer data, moving about in the buffer plus entering data in both hex and ASCII. Press ESC to return to the editor command prompt then press ESC again to return to the main command list. Be sure that the 93C56 chip is securely attached. We will now read the contents of the 93C56 into the buffer. To do this we will use COMMAND 3 - READ DEVICE INTO BUFFER. Press “3” then “Y”. Note that the SOCKET POWER led lights while the data is read from the part. Now press “5” to view the buffer contents. The data in the buffer should now match the figure on the right. This data was programmed into the test chip specifically for use with this tutorial.

We will now use several buffer editor commands to learn more about the system. Press “C” then “1”. This is the buffer CLEAR command. The data from the chip has been erased from the buffer and the buffer contents have been reset to FF. So why read the chip and then clear the data? To show another way to read the chip while in the buffer editor. Press “R”. Note the response READ DEVICE INTO BUFFER and also ENTER BUFFER STARTING

ADDRESS->. You will usually read data from a part into the buffer starting at address 0, however with this command you may specify a different buffer starting address if you wish. Just to show you type 80 and press ENTER. Press “Y” to confirm that you indeed wish to read the part. Note that the data appears starting at address 80, not address 0. Press the down arrow key to see the rest of the data if you wish then press the HOME key to return to the buffer beginning. Now clear the buffer again (“C” then “1”). Now read the chip into the buffer using the “R” command. Press “R” then “0” then ENTER then “Y”. The 93C56 data again appears beginning at address 0.

We are now going to change the data in the buffer, but not in the device (93C56). You use the “M” command as before except now we will change the hex codes which represent the word BYTES to BIKES. Press “M” then “50” then ENTER. Note the position of the block cursors, both HEX and ASCII. Using the right arrow key move the ASCII block cursor until it is over the Y character. This is address 59 (look in the edit window - top left). Press TAB to change the data entry mode to ASCII. Now simply press I and K. You have changed the word from BYTES to BIKES. Press ESC to exit the M command. The data has been changed in the buffer but not in the 93C56 chip. It still has the codes for BYTES. Since this is the case the buffer and the chip data do not match.

| EDITOR MODE = BYTE | | BUFFER END = FFFFFF | DEVICE SIZE = 0 - FF |
|---------------------------------------------------|-------------------------------------------------------|-------------------------------------------------------|----------------------|
| 0 | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | ***** |
| 10 | 2A 20 20 20 20 41 4E 44 52 4F 4D 45 44 41 20 20 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | * ADRRHEA * |
| 20 | 2A 20 20 20 20 52 45 53 45 41 52 43 48 20 20 20 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | * RESEARCH * |
| 30 | 2A 20 20 20 20 54 45 53 54 20 46 49 4C 45 20 20 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | * TEST FILE * |
| 40 | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | ***** |
| 50 | 20 20 20 20 32 35 36 20 42 59 54 45 53 20 20 20 | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | 256 BYTES |
| 60 | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A | ***** |
| 70 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| 80 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| 90 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| AA | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| BB | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| CC | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| DD | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| EE | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 AA 55 | .U.U.U.U.U.U.U.U |
| FF | 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF | 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF | |
| ENTER EDITOR COMMAND ("?" FOR HELP/"ESC" TO EXIT) | | | |

The system has a command (Command 8) which allows you to compare the data in the buffer to the data which is in the chip. Press ESC to return to the command list. Press "8" (compare device with buffer). Command 8 defaults to beginning the comparison at buffer address 0 so just confirm that you wish to compare by pressing "Y". Note the result on the right. You should see COMPARISON COMPLETE:000002 ERROR(S). There are two errors because you changed two bytes in the buffer and they no longer match the bytes in the chip. Now examine which bytes don't match. At the EXAMINE ERRORS?-> prompt press "Y". The system shows you the data at buffer address 000059 which is now 49 plus the device data at the memory chip address 000059. Note that the buffer address is what you changed (49) while the device address is the original value (59) which is still in the chip. Press "C" to continue. Note the next comparison error is at address 00005A with the buffer data 4B and device data 54. Now let's change the data at buffer address 00005A back to 48. Press ESC then "5" (buffer editor). Now press "M" then "5A" then ENTER. Note that the block cursor is already at address 5A and shows that the data is 4B. To change the data back to match the data in the chip press "54". The data in the buffer has now been changed back (also note the ASCII window where the K has changed to a T). Now the number of errors should be 1. Let's check. Press ESC, ESC, "8" then "Y". As expected, one error. Press "Y" to examine the error. Note that only one byte in the buffer (address 000059) does not match the same byte in the chip. Press ESC to terminate.

This is how you locate data differences between two chips or a chip and any data in the buffer. Let's say that you have a chip which contains a specific count value. You want to know at what address the count value is stored. First read the chip into the buffer. Now take another chip with a different count value or cause the data to change in the same (original) chip. Now using Command 8 locate the addresses where the data in the chip no longer matches the data in the buffer. This is one location where the count value data is stored. **Note:** Most times the numbers stored will occupy more than one byte although usually in the same address area.

THE INSPECT COMMAND - A handy way to view data

The Inspect command has two functions. First it allow you to inspect the contents of a chip without reading it into the buffer. Second it lets you easily see areas of a chip where data does not match the buffer. Let's look at how it works. If you are not already in the buffer editor press "5". Clear the buffer ("C" then "1"). Now press "I". INSPECT DEVICE CONTENTS appears. The command is waiting for you to specify which area of the chip you wish to inspect. Press "0" then ENTER. Note that the data from the chip appears on the screen. Also note that in the top center window the **BUFFER END =** message has been replaced with **DEVICE CONTENTS**. Press any key. The buffer editor display reappears. Note that the INSPECT command displayed an entire screen of data (256 bytes) from the chip. This is a fixed value and cannot be changed. Even if you are working with a chip which is smaller than 256 bytes, an entire screen will always be displayed. Now let's look at the compare mode. Remember that the buffer has been cleared to FF. Press "I" then "0" then <SP> then "0" then ENTER (**Note:**<SP> is the space bar). The entire screen with the exception of eight bytes at the very bottom (line F0) has changed to reverse video (hi-lite) with the chip data (DEVICE CONTENTS) displayed. Notice what the eight byte values are, that's right, FF. The same as the data in the buffer. The inspect with compare function shows the device contents and at the same time indicates with reverse video which bytes in the buffer don't match the same bytes in the chip. In this case, only eight bytes match. Press any key to return to the buffer editor screen. Now lets make it a little less overwhelming. Read the chip into the buffer. Press "R" then "0" then ENTER then "Y". The data from the chip appears on the screen. Now, as before change the word BYTES to BIKES. Press "M" then "59" then ENTER. This time we're going to work in HEX so press "49" (HEX code for I) then "4B" (HEX code for K). Now, as before, two bytes in the buffer don't match the data in the chip. Press ESC to exit the Modify command. Now use the Inspect command to show the differences. Press "I" then "0" then <SP> then "0" then ENTER. Note the byte addresses which you changed are shown in reverse video. The displayed data is from the device but the underlying buffer data is different. This mode of the inspect command (inspect with compare) requires two addresses. The first is the starting address in the chip. The second is the starting address in the buffer which is to be compared with the data in the chip. Now lets wrap up the Inspect command with two more examples. Press ESC to return to the editor screen. Press "R" then "0" then ENTER then "Y". This again reads the chip into the buffer. Now press "I" then "0" then <SP> then "0" then ENTER. Note no reverse video blocks since the buffer and chip now match. Press any key to return to the editor screen. Now press "I" then "0" then <SP> then "100" then ENTER. Note the same display as before when the buffer was set to all FF. Why is this? Because this time the buffer address used to compare to the chip was 100 not 0. The buffer data beginning at address 100 is all FF. Lets check. Press any key and note that the buffer address now starts at 100. The inspect command will set the displayed starting address to whatever value was used for the comparison. Press the HOME key to return to the buffer start.

ON TO PROGRAMMING

Up to this point you have learned to work with the data in the buffer, read the contents of the chip and use the compare (command 8) and buffer editor Inspect command. It is now time to actually program new data into the 93C56. Since the 93C56 already has data programmed, let's start by saving a copy of the data in the chip at a buffer address other than 0. This feature of the buffer editor allows you to use the buffer as a scratch-pad for temporary data storage. The chip data has already been read into the buffer beginning at address 0. Since the size of the 93C56 chip is 256 bytes, the data starts at 0 and ends at FF. If you look at the upper right hand window you will see that the device size is displayed (DEVICE SIZE = 0 - FF). Press the DOWN ARROW key. Note that the bottom line start address is 100. This is the same 100 used in the previous inspect example. Press the DOWN ARROW key until address line 100 is the first line in the display. You see the chip data move off the screen as you move forward in the buffer address space. Now press the PgDn key. Note that the first line address is now 200. Page Down (PgDn) advances the buffer display 256 bytes or one screen. We will now read the chip again except the data will be placed into the buffer starting at address 200. Press "R" then "200" then ENTER then "Y". The chip data appears on the display. Now press the PgUp key twice. The first press moves you back in the buffer 256 bytes (one page) then one more page back to the buffer start. We will now change the buffer data and program the chip. Press "M" then "32" then ENTER. Now press TAB (changes entry mode to ASCII) and type REFLASH FILE. Note that it replaced TEST FILE in the buffer. Press ESC, ESC. This returns you to the command list. Press "2" then "Y". The system programs the changed buffer data into the part and confirms proper programming. You see "PROGRAMMING COMPLETE" - "DATA VERIFICATION IS CORRECT". Press "N" since you will not be programming another part. Now let's see if the data is indeed in the chip. Use the Inspect command. Press "5" then "I" then "0" then ENTER. The chip data with the changes you made appears. Press any key. Now let's change it back. Press the PgDn key twice. There is the original data from the chip. Now we will use an editor command to program the original data back into the part. The original data is located in the buffer beginning at address 200. We will use the PROGRAM DEVICE FROM BUFFER command. Press "P" then "200" then ENTER then "2FF" then ENTER then "P". The system programs the data from buffer address 200 to buffer address 2FF and then confirms proper programming. Let's confirm the programming again with the Inspect command. Press "N" to return to the editor prompt then "I" then "0" then <SP> then "0" then ENTER. This is actually Inspect with compare. Note that TEST FILE is shown in reverse video indicating that it does not match what is in the buffer at address 0. Press any key to terminate the Inspect command and display the buffer where REFLASH FILE remains.

THE WRITE COMMAND - Before we move on, here is a command which you will find useful for small data changes. It's the WRITE command. The Write command allows you to write (program) small amounts of data into a chip without programming the entire part. To use the Write command you first specify the address range in the device where you want the data from the buffer placed, and then the starting address in the buffer where the data will start. Here's how it works. At this time the buffer has REFLASH FILE between addresses 32 through 3D. The 93C56 chip has TEST FILE stored at addresses 32 through 3D. Press "I" then "0" then ENTER to quickly view TEST FILE in the 93C56. See it? Press <SP> to terminate. Now we are going to write REFLASH FILE into the 93C56 overwriting TEST FILE. Press "W" then "32" then <SP> then "3D" then <SP> then "32" then ENTER. Watch as the Write command works. The data is written and then correct programming is verified. The message remains briefly then terminates. Let's review what you did. The three address values you entered after you pressed "W" instructed the system which device (93C56) addresses to use to define the start and end range (32 to 3D). The third value informs the system at which buffer address you wish to start (where the data will come from that you are putting into the chip). In this case it was buffer address 32. Let's check the chip. Press "I" then "0" then ENTER. TEST FILE has been replaced by REFLASH FILE. Press ESC. Now let's replace REFLASH FILE with TEST FILE. Press the PgDn key twice. There is the original data from the chip at address 200. TEST FILE begins at address 232 so press "W" then "32" <SP> "3D" <SP> "232" then ENTER. The system writes the data beginning at buffer address 232 into the 93C56 beginning at address 32 and ending at address 3D. Let's check it. Press "I" then "0" then ENTER. TEST FILE has replaced REFLASH FILE. The Write command gives you the capability to place any data you wish from the buffer into any locations in your chip.

WORKING WITH FILES

The last part of this tutorial describes working with files. You will learn to locate a file on the drive of your computer, program a chip from the data in the file plus save the contents of a chip to a file. You will also learn to save data in the buffer to a file. Let's begin. The first concept you need to understand is how the system organizes drives, directories (folders) and files. Since you are running this tutorial on your computer, we obviously don't know the configuration. You may be running under Windows or from the bootable CD. The first step is to return to the command list. Press ESC until the command list is displayed. Now press "P". This is the PATH command. It allows you to view directories and locate drives on your computer. When the PATH command opens it displays the directories (Microsoft calls them folders) on the current drive. Remember drives are specified by letters followed by a colon (C:). The lowest level directory on a drive is called the ROOT. We like to keep things simple so we suggest that if you are going to save files on a drive, use or make a new directory in the root. If you install our software from the CD using the install program, it will create a directory called EPROM and then copy the files from the CD. The path would look like this C:\EPROM. Since we don't know what your PATH command displays, let's scan for drives. Look at the lower left part of the display. Note the second line from the bottom (F2=VIEW FILES | F5=SCAN FOR DRIVES | F1=HELP). Press F5 (function key F5). The system will scan your computer for drives. When the scan completes you will see AVAILABLE DRIVES = followed by

the drive letters. (Note: If you are running from the bootable CD the drive letters displayed may not be the same drive letters assigned by Windows. Usually when you boot from the CD the CD drive is A:.) Choose a drive letter you wish to use. Be sure it is a drive that will allow you to save files, not a CD or DVD.

In order to proceed with the tutorial, we are going to assume drive C and choose the EPROM directory. From the AVAILABLE DRIVES list press "C" (chooses drive C:) then use the arrow keys to place the lite bar over the EPROM directory. Now let's look at the files in the EPROM directory. Press F2. Any files in the EPROM directory are displayed. If the directory is empty you will see NO FILES. In the lower left corner of the display you will see ENTER FILENAME TO RECALL ->. Type REFLASH.BIN and press ENTER. The RECALL FILENAME is a name you can use in the future for any file operation; both reading and programming. The system returns to the command list.

SAVE BUFFER TO DISK FILE - We will now save the first 256 bytes of the buffer to a file using COMMAND A (SAVE BUFFER TO DISK FILE). Press "A". At the lower left corner of the display you see ENTER FILENAME ->. Press the Ins (Insert) key. The REFLASH.BIN filename appears. Press ENTER. Now enter the buffer starting address where the save is to start. Press "0" and press ENTER. Now enter the buffer ending address (the last byte to save). Press "FF" (remember device size is 0 to FF) and press ENTER. Now press "Y" to confirm you wish to save. The buffer data is saved under the filename REFLASH.BIN and you are returned to the command list. Lets be sure the file is there. Press "D". You should see the filename (REFLASH.BIN) on the screen. The "D" (Directory of files) command shows you the files in the directory. Remember it because you will use it in the future. Press ESC to return to the command list.

SAVE DEVICE TO DISK FILE - In the previous exercise you saved 256 bytes of data from the buffer to a file. The system also allows you to save data directly from the part (93C56). Before we begin press "5". The buffer shows the REFLASH FILE message. Press "I" then "0" then ENTER. The device (93C56) is programmed with the test file message. Press ESC, ESC to return to the command list. Press "D". At the ENTER FILENAME TO RECALL -> prompt type TESTFILE.C56 and press ENTER. Now press "B" (SAVE DEVICE TO DISK FILE). Press the Ins (Insert) key. The filename TESTFILE.C56 appears. Press ENTER. Now press "Y" to confirm the save. You will see * DEVICE SAVE OPERATION COMPLETE * once the data has been read from the part and saved to the file. Let's check. Press any key then press "D". You should see the file TESTFILE.C56. Remember if you want to check the contents of a chip before you save it, use the Inspect command to get a quick look at the contents. If the chip data looks good then use Command B to perform the save. Press ESC to return to the command list.

PROGRAMMING FROM A DISK FILE

You now know two ways to save data to a disk file; Command A and Command B. Programming a part from a disk file is also very easy. Before we begin lets clear the buffer and program the 93C56 with FF. This will erase the part. Press "5" then "C" then "1". The buffer is set to all FF. Press ESC to return to the command list. Press "2" then "Y". The 93C56 is now filled with FF. Press "N" to terminate the command. Let's be sure the part is blank. Press "6" (VERIFY DEVICE IS ERASED) then "Y". You will see *** DEVICE IS ERASED ***. Press "N" to terminate. (Note: The ERASE test confirms that a part is completely programmed with the FF value.)

Now let's reprogram or reflash the 93C56. This is the same operation that you will use to restore an airbag module or immobilizer. First find the file that we wish to program into the 93C56. Press "D" and locate the REFLASH.BIN filename on the display. Use the arrow keys to position the lite bar over the name. Now look in the lower right corner of the display. You will see (FILENAME: F3 TO PROGRAM | F4 TO LOAD). Press F3. The system will clear the entire buffer and then load the file. With the file loaded you may PROGRAM or SKIP. Press "P". The system programs the chip and confirms correct programming. Note: If you only wanted to load the file into the buffer you would press F4. Press "N" to terminate. Now let's check the part. Press "6" then "Y". You will see - DEVICE NOT ERASED - so there is data other than FF in the 93C56. Press "N" to terminate then press "5" then "I" then "0" then ENTER. You see that the 93C56 has been programmed with the REFLASH.BIN file.

We're almost done with the tutorial. As a last exercise press ESC, ESC to return to the command list. Press "D" then locate the TESTFILE.C56 filename and place the lite bar over the name. Press F4. The system loads the file and shows a summary of where the file data was loaded. Press ESC then "5". The TESTFILE.C56 has been loaded into the buffer. Now use the inspect command with compare to confirm the differences between the chip and the buffer. Press "I" then "0" then <SP> then "0" then ENTER. The REFLASH FILE addresses are shown in reverse video indicating they don't match the buffer. Press ESC, ESC. Now press "2" then "Y". The system programs the TESTFILE.C56 data into the 93C56 which restores the chip to its original condition. Press "N". You're now at the command list and finished with the tutorial. Use what you've learned to proceed with your work.

CONTINUE TO LEARN - This tutorial covers the basic operation of the EPROM+ system when used with serial eeproms and is intended to help you get started while learning the fundamentals. The EPROM+ system manual describes each software command and function. We strongly suggest that you take time to fully explore the many features and capabilities of your system.