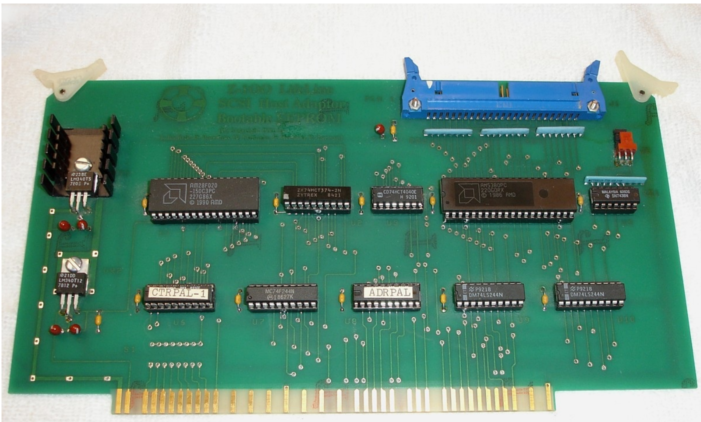
Z-100 LifeLine

Since 1989

March 2025

#WEB

~~~~~~



"Z-100 LifeLine" SCSI Host Adaptor & Bootable EEPROM

What is a Breakout Switch?

by Steven Vagts
Editor, "Z-100 LifeLine"

What is a Breakout Switch?

From the SCSI Host Adaptor
& Bootable EEPROM Board
Copyright (C)1992

When Zenith and Microsoft support moved on from supporting the Z-100 series computer to developing PC hardware and software in the late 1980s, it quickly became apparent that any additional work on the Z-100 would have to be done within the Z-100 community. This was the driving factor for Paul Herman to begin publishing the "*Z-100 LifeLine*" in 1989. The purpose of the "*LifeLine*" was (and still is) to provide a central point for dissemination of information and development between venders, research teams and the ultimate users.

One of the first projects was the development of a new "*Z-100 LifeLine*" SCSI Host Adapter, also referred to as the LLSCSI Controller Board, to replace the aging "Winchester" MFM hard drives.

Originally conceived at the 1990 Z-100 Get-Together in Norfolk, Virginia, the SCSI/EEPROM board was a product of 1-1/2 years of research and development. Under the auspices of "*Z-100 LifeLine*", a development team was selected in November of 1990, and production units were first delivered in March of 1992.

A full team of volunteers began work on this project in 1990:

Paul F. Herman	Project Coordinator, EEPROM programming, Marketing
Robert F. Hassard	Engineering design, Prototype development
Robert W. Donohue	MTR-100 ROM and BIOS programming
William E. Flanagin	SCSI programming
Travis J. Barfield	Parts acquisition & Manufacturing
Michael Zinkow	Z-DOS development
John Beyers	BIOS programming & Z-DOS utilities

The "*Z-100 LifeLine*" SCSI Host Adaptor/Bootable EEPROM Board, hereafter referred to as the LLSCSI/EEPROM Board, was a multifunction S-100 board designed for the Heath/Zenith Z-100 Series computer. It provided the following features:

- An industry standard **SCSI Host Adaptor.** This allowed you to connect fixed or removable media hard drives, tape backup units, CD-ROM drives, floptical drives, or any other device which included an imbedded SCSI controller, to your Z-100 computer.

- A **Bootable EEPROM Device.** This non-volatile memory device, based on the AM28F020 flash programmable EEPROM, could be programmed at any time without removing it from this board. Programming software was provided with the board. The EEPROM device was fully bootable
~~~~~~

and could contain up to 256Kb of user selectable programs or files.

- A **Hardware Breakout Switch.** The breakout switch circuitry worked by generating a non-maskable interrupt (NMI) on the S-100 bus. Firmware to support the breakout switch for debugging was provided in the MTR-100 Monitor ROM (aka ZROM), beginning with version 3.1.

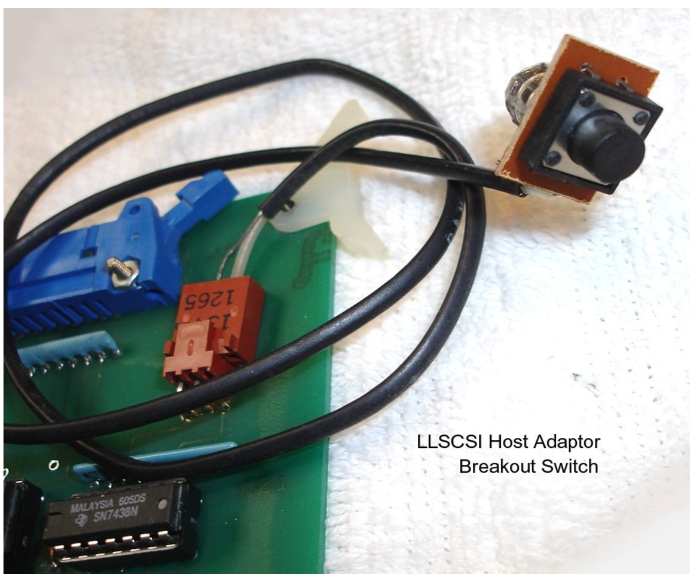
This was the first time that a Breakout Switch appeared on a LifeLine Project, but it was not the last... Several years later it also appeared on the "*Z-100 LifeLine*" IDE Controller Board.

Note: For more information on the LLSCSI Board and the LLIDE Controller Board, please refer to their respective articles on the "*Z-100 LifeLine*" Website.

But what was this fascination with a Breakout Switch, and more importantly, what does it do?

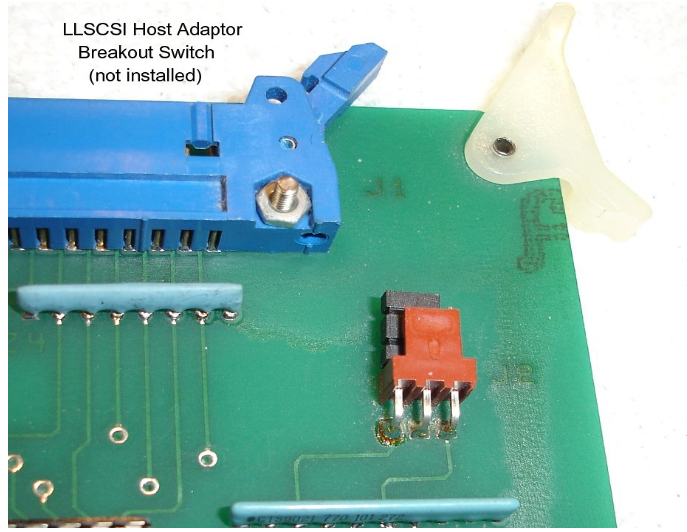
Hardware Installation:

The Breakout Switch connector is along the right side of the LLSCSI Controller Board.



Mount the Breakout Switch in a convenient location so that it can be accessed with the case closed. Back-panel mounting should be fine for occasional use. Programmers who expect to make use of the breakout switch may want to connect it to a long cable, where it can be brought to the front of the machine when in use. Connect the breakout switch cable to connector J2 on the SCSI/EEPROM Board.

IMPORTANT NOTE: If you will NOT be using the breakout switch feature, you MUST place a shorting jumper over the LEFT two pins of connector J2 at the right side of the LLSCSI board.



On the LLIDE Controller Board, the Breakout Switch is a push button mounted at the extreme upper right corner beside the mounting lever. If you wish to install a remote switch, you could solder a pair of wires across the push button terminals on the solder side of the controller board.

Testing the Hardware:

Use the following procedures to test your installation:

1. After mounting the LLSCSI Controller or the LLIDE Controller in any available S-100 bus slot, turn the power on. You should hear the usual two beeps and get a hand prompt. If not, check the ROM installation, the settings of J-101 and J-102, and your cable connections.

2. Boot the system with MS-DOS v3.1 or later. You can boot from a floppy or a Z-217 controlled hard drive. The DIP switch has been set so the default boot device is the 5" Floppy Drive, so you can just press {**B**} for Boot, and press {**RETURN**}. Or you can manually select the boot device by using one of the following command sequences (If the device is installed):

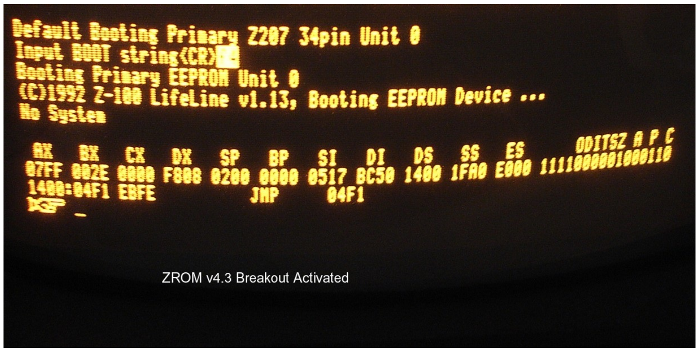
{**B**}oot {**F1**} boots from the 5-1/4 inch floppy
{**B**}oot {**F2**} boots from the 8 inch drive
{**B**}oot {**F3**} boots from the Z-217 hard drive
{**B**}oot {**F4**} boots from the SCSI EEPROM, or
{**B**}oot {**F4**}{**p**} boots from the SCSI EEPROM
{**B**}oot {**F4**}{**s**} boots from the IDE NVsRAM

Where:
{**P**} or {**p**} is the Primary EEPROM device
{**S**} or {**s**} is the Secondary NVsRAM device

3. You should now be at the DOS prompt.

4. If you installed the LLSCSI Breakout Switch, try activating the switch. On the LLIDE Controller, just push the button. In either case, you should get a display of the CPU register contents, along with an unassembled assembly language instruction, followed by the

MTR-100 hand prompt. If not, you may have installed the switch incorrectly.



5. Type the {**G**} monitor command. The word '*Go*' should be displayed on the screen. Now hit the {**RETURN**} key, and you should be back to the DOS prompt.

Great! But what is it used for? Is there any software for it?

The NMI Breakout Switch

The breakout switch portion of the LLSCSI Controller or the LLIDE Controller is a tool for programmers. It allows you to break out of any executing program, perform various debugging chores, and then continue execution. The switch works by generating a non-maskable interrupt (NMI) on the S-100 bus. The LLSCSI Breakout Switch schematic is in Figure 1.

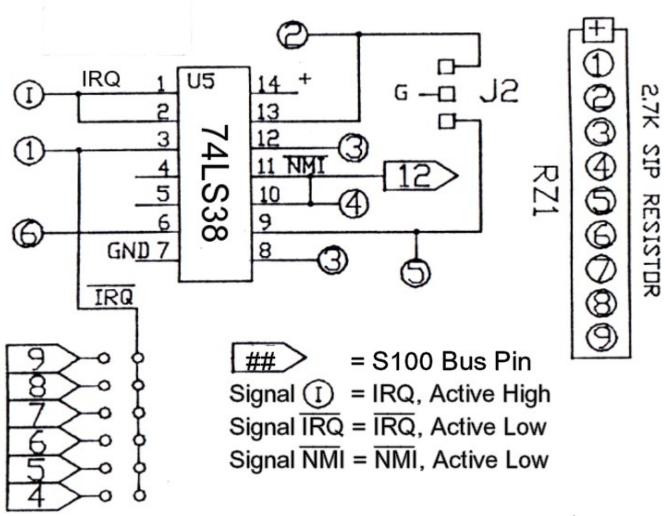


Figure 1.

In order to use the breakout switch for its intended purpose, you need special software to support it (a non-maskable interrupt routine, to be specific). The MTR-100 Monitor ROM v3.1 and later includes debugging capability which utilizes the NMI breakout switch. Additional instructions accompanied the MTR-100 with details about how the switch was used with the ROM.

Use of the breakout switch was not recommended unless you were a programmer who understood assembly language programming. If you do not fall into this category, you may not want to install the breakout switch.

IMPORTANT: If you do not install the breakout switch on the LLSCSI board, a shorting jumper must be placed over the two pins of J2 nearest the center of the board. Otherwise, your Z-100 may fail to operate correctly.

Theory of Operation

As we mentioned, the LifeLine SCSI/EEPROM Board serves three functions; a SCSI host adaptor, a bootable in-circuit programmable EEPROM, and a Breakout Switch. The IDE Controller is similar, but for IDE devices. In each case, the operation of the Breakout Switch is similar.

Perhaps the best way to describe the function of the Breakout Switch is to do an example of its use. While many programmers may find other uses for the switch, I have only used it in a few specific instances - usually troubleshooting a stalled Z-100.

While checking out the installation of the LLSCSI Host Adaptor Controller, I found that the LLSCSI EEPROM would boot just fine with the Monitor-ROM v3.2 and v4.24, but would stall right after the booting message with ZROM v4.3!

Instantly, I figured there was an undiscovered bug in the newer ZROM, but I needed to check it out - a perfect use for the Breakout capability.

I quickly put together a suitable cable with a momentary-closing push button switch and installed it on the board's connector.

I attempted to boot from the EEPROM using the command;
{**B**}oot {**F4**} {**CR**}

and the display showed:

"Input Boot String <CR> F4"
"Booting Primary EEPROM Unit 0"
"(C)1992 Z-100 LifeLine v1.13, Booting EEPROM Device ..."
"No System"

And then immediately stalled...

Pressing the Breakout Switch

Pressing the Breakout Switch at this point interrupts the presently running DOS program, displays the status of the registers and flags, and brings us back to the Hand Prompt:

AX BX CX DX SP BP
07FF 002E 0000 F808 0200 0000

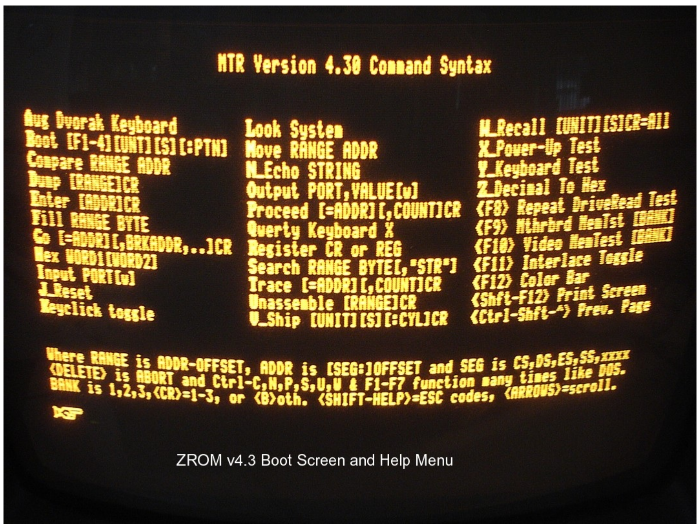
SI DI DS SS ES
0517 BC50 1400 1FA0 E000

ODITSZ A P C
1111000001000110

(All on one line)

1400:04F1 EBFE JMP 04F1

At this point (using ZROM v4.x) you can press the {HELP} key to see the commands available from the Monitor ROM (ZROM).



ZROM v4.3 Boot Screen and Help Menu

There are several commands that you can use, which are similar to those available in the DEBUG utility.

ZROM Function
Syntax:
{**C**}ompare RANGE ADDR
{**D**}ump [RANGE]CR
{**E**}nter [ADDR]CR
{**F**}ill RANGE BYTE
{**G**}o [=ADDR][,BRKADDR,..]CR
{**H**}ex WORD1[WORD2]
{**I**}nput PORT[w]
{**M**}ove RANGE ADDR
{**O**}utput PORT,VALUE[w]
{**P**}roceed [=ADDR][,COUNT]CR
{**R**}egister CR or REG
{**S**}earch RANGE BYTE[,"str"]
{**T**}race [=ADDR][,COUNT]CR
{**U**}nassemble [RANGE]CR

Where RANGE is ADDR-OFFSET, ADDR is [SEG:]OFFSET and SEG is CS, DS, ES, SS, xxxx.

As with all the boot screen commands, you only press the first letter of the command and type the arguments as presented above. {**CR**} is the {**RETURN**} key.

As this is not an article on DEBUG, we are only using a few of these commands today. For a more thorough discussion of all the DEBUG commands, please refer to the Microsoft MS-DOS Manuals.

So, back to our stalling situation, even without doing anything else, we can see that we are stuck in an endless Jump-to-itself loop. So this was intentional.

If we press {**T**} for trace, we get the same line displayed for each press of {**T**}. Let's see what is going on...

The line;
1400:04F1 EBFE JMP 04F1

shows us that we are in SEGment 1400:. This will usually vary considerably with different computers and will most certainly not be the same as what you may see on your computer. It is the area in memory that the computer decided to use. Different computer programs will end up in different areas (Segments) of memory.

However, in this case, the 1400 SEGment was purposely set and used within the program to create the Boot code for the EEPROM.

The 04F1 is the OFFSET in this particular program, which generally begins at offset 0100 in hex. This will generally NOT change, and is like a line number in a BASIC program, except unlike a line number, this OFFSET is tracking the number of bytes used to this point in the program. Every byte of code and data is being counted.

The EBFE are the two bytes representing the JMP command, where EB#h is JMP (next address +#). If the # is less than 80, then the jump address is the next address +#. But if the # is greater than 7F, then the jump address is the next address - (FF-#). In this case, next address minus 2; it is jumping back to itself!

The JMP 04F1 is telling us in plain language that it is jumping to itself!

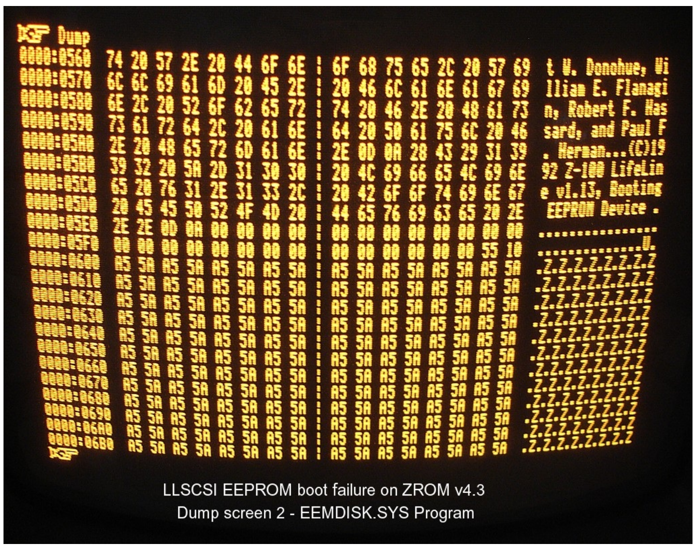
Now, let us check out why?

In cases like this, before I Unassemble a section of code, I like to Dump the area around this section because the Dump command will show us any areas that are actually not code, but ASCII text, and this particular program has much of it.

So press **{D}**ump and enter the desired OFFSET, **0400**. You can enter the entire address by including the SEGment, 1400:, but the current segment is assumed. This displays the screen: (Sorry, but my pictures show SEGment 0000:)

LLSCSI EEPROM boot failure on ZROM v4.3
Dump screen 1 - EEMDISK.SYS Program

And, pressing just **{D}**ump again gives us the next screen:



LLSCSI EEPROM boot failure on ZROM v4.3
Dump screen 2 - EEMDISK.SYS Program

The areas that did not hold text is where we need to use the {**U**}nassemble command to actually list and make sense of the code.

Since we are stuck at the 1400:04F1 line, we will try looking at the code before that line. I included this last picture to give you an idea what the Unassemble screen actually looks like. The code is listed in columns of addresses, code bytes, and program ststements, but there wasn't much of interest found.

Since we began dumping at 1400:0400, let us use the command:

{**U**}nassemble **1400:0400**

1400:0400 xxxx JMP 0421

However, as the code between the bytes 0402 and 0421 did not make sense, they are probably just data in the form of data bytes and data words. So, we just press {**U**}nassemble again, and concentrate on the instructions beginning at:

1400:0421 xxxx JMP 0448

We really do not need to worry about the code bytes represented by xxxx, nor most of the address numbers, so for readability, we will ignore these in the following code statements, as we continue to press {**U**}nassemble:

```
1400:0448 NOP
     0449 PUSH  CS
          POP   DS
          MOV   AX,1400 ; SEGment 1400
          MOV   ES,AX
          MOV   CX,3C00 ; Block size
          MOV   SI,0400 ; Starting Offset
          MOV   DI,SI
          CLD
          REPZ
          MOVSB
          JMP 1400:0460

1400:0460 MOV   DI,0404
          MOV   SI,040B
          MOV   AL,[SI+0A]
          MOV   CL,0A
          REPZ
          MOVSB
          MOV   SI,0436
          MOV   CL,12
          REPZ
          MOVSB
          PUSH  CS
          POP   DS
          MOV   [0600],AL
          MOV   AL,08
          OUT   FC,AL
          MOV   SI,05AB
```

Note: 05AB Points to the start of the opening string!

```
          MOV   BP,0039
```

And 39h = 57d, which gives the length of the opening string.

```
1400:0483 CALL  04F3 - Print string routine
          PUSH  CS
          POP   ES
          REPZ
1400:049D CMPSB
          JNZ   04E8
```

Note: 04E8 Jumps to the routine that prints the "No System" message!

```
          MOV   AX,[BX+041C]
          ADD   AX,[0404]
          DEC   AX
          MOV   CL,[040E]
1400:04AD SHR   AX,CL
          XCHG  BP,AX
          MOV   AX,[BX+041A]
          DEC   AX
          DEC   AX
          MOV   CL,[0412]
          SHL   AX,CL
          ADD   AX,[0410]
          XCHG  BX,AX
          MOV   AX,0040
          MOV   ES,AX
          XOR   DI,DI
          MOV   DX,00CE
1400:04CB MOV   AX,BX
          OUT   DX,AX
1400:04CE CALL  04FE
          MOV   CX,0400
1400:04D4 IN    AX,DX
          STOSB
          LOOP  04D4
          INC   BX
          DEC   BP
          JNZ   04CB
          MOV   AX,CS
          ADD   AX,0040
1400:04E1 MOV   DS,AX
          JMP   0040:0000    Start of EEPROM?

1400:04E8 MOV   SI,050C
```

Note: 050C is the start of the "No System" Message!

```
          MOV   BP,000B
          CALL  04F3         Print Msg
1400:04F1 JMP   04F1         JMPs to self!

1400:04F3 LOBSB
          PUSH  SI
          CALL  FE01:0019
```

So, it is just like we were in DEBUG, developing a program. It works the same way, and you can make out a lot about what is happening.

The key to our problem seems to be the CMPSB statement at 1400:049D! Fortunately, we have most of the source code for these LLSCSI files, if we can identify the routine affected...

Initially, I thought this code was from the LLSCSI.SYS device driver, but when I used the command;

DEBUG LLSCSI.SYS

and tried dumping the file, the code did not match at all.

I suspected IO.SYS next, but the command;

DEBUG IO.SYS

again showed that was not the file, either.

After some head scratching and searching with DEBUG to list the contents of other routines, I finally found that it actually matched the last portion of the EEMDISK.SYS file!

While I had thought this file was limited to programming the EEPROM, it is also used to boot the EEPROM programming!

Now we had the fully documented file to explain the original intent of the code, the reason for the test, what was tested - all from the Breakout Switch!

As it turns out, while searching for the source code for EEMDISK, I also found a later version, but it did not work any better.

After much more experimenting and testing, I have come to the conclusion that it is indeed a problem in ZROM v4.3 and will require further troubleshooting.

For now, we will just need to accept the fact that the LLSCSI board will NOT work with the newer ZROM v4.3.

I hope this explains at least one of the useful functions of the Breakout Switch. It is certainly another tool to help programmers in troubleshooting their routines.
I hope you find it useful.

If you have any questions or comments, please email me at:
z100lifeline@swvagts.com

Cheers,

Steven W. Vagts