EPROM EMULATOR

USER'S MANUAL

MODEL EE08

Version 1.1
6-18-93

INTRODUCTION:

Thank you for selecting a Technical S*olutions* product. We have made every attempt to provide a quality product at an affordable price. Our goal is to provide tools for Engineers and Technicians that are inexpensive, but fully functional. If you have any problems or comments, please don't hesitate to call or FAX us and let us know.

The EE08 uses SRAM or battery-backed SRAM to emulate EPROMS up to 4MBits in size. An IBM compatible computer (PC) is used to down-load object code into the SRAM.

CONFIGURATION:

1. CONFIGURE SRAM SIZE:
Set jumper J3 to indicate the size of SRAM being used. For SRAMs larger than 1Mbit, jumper pin 1 to 2 (closest to the edge of the board). For smaller SRAMS, jumper pin 2 to 3 . This jumper determines whether SRAM socket pin 30 receives +5V or A17. This information is written in the silk-screen inside the SRAM socket.

2. CONFIGURE TARGET SOCKET SIZE:
Set jumper J8 to indicate the size of the target socket. This jumper determines which target pin is used to supply power to the EE08. For 32 pin target sockets, jumper pin 1 to 2 (closest to the board edge). For 28 pin target sockets, jumper pin 2 to 3.

3. INSTALL SRAM :
Install the SRAM into the EE08. Insert the SRAM with the notch pointing the same direction as the notch shown on the silk-screen. If you are using a 28 pin SRAM, insert it as shown on the silk-screen, with socket pins 1,2,31 and 32 empty.

4. SET THE DIP SWITCHES:
The dip switches configure the EE08 for the EPROM size being emulated. These MUST be set correctly for the unit to function properly. These switches intercept the upper address lines from the target. SW1 (closest to the download cable) controls the highest address line (A18): The next one controls A17 and so forth. The last 2 switches are not connected. To emulate a 4Mbit EPROM (27040), all switches are closed. While emulating a 2Mbit EPROM, SW1 is opened. To emulate a 1Mbit EPROM, SW1 and SW2 are opened. This continues until all switches are opened for a 64Kbit EPROM. This is summarized below.

EPROM	**SW1**	**SW2**	**SW3**	**SW4**	**SW5**	**SW6**
27040	on	on	on	on	on	on
27020	off	on	on	on	on	on
27010	off	off	on	on	on	on

27512	off	off	off	on	on	on
27256	off	off	off	off	on	on
27128	off	off	off	off	off	on
2764	off	off	off	off	off	off

BASIC INSTALLATION:

1. Turn power off to the Target.

2. IF YOUR TARGET HAS IN-CIRCUIT EPROM OR FLASH PROGRAMMING CAPABILITY, DISABLE IT!

The EE08 is a 5 Volt ONLY device. Programming voltages will DAMAGE the device and will void the warranty. Even short surges during power-up or reset can be damaging.

3. Inset the EE08 into the target socket.

BE CAREFUL TO INSERT THE EE08 PROPERLY. INSERTING THE EE08 BACKWARDS CAN RESULT IN PERMANENT DAMAGE TO THE EE08 OR THE TARGET SYSTEM!

Pin 1 of the EE08 is identified by the small notch in the DIP plug, similar to an IC. Insert the EPROM Emulator so that this notch points the same direction as the notch on the target socket.

If you are inserting the EE08 into a 28 pin socket, insert it so that pins 1,2,31 and 32 hang over the end of the socket. These pins are designed to float un-terminated. When the EE08 is inserted properly, pin 3 of the EE08 will be placed in pin 1 of the socket. Be sure to configure J8 for a 28 pin Target socket.

4. Select an un-used printer port on your IBM compatible PC/XT/AT.

5. Plug the supplied DB-25 to RJ-45 converter into the selected port.

6. Plug one end of the supplied RJ-45 cable into the adapter.

7. Plug the other end of the RJ-45 cable into the EE08 jack.

8. (OPTIONAL) Connect a jumper between one of the reset pins (active high or active low) on the EE08 to the appropriate place on the target if automatic reset is desired during each download. RESET and /RESET are tri-stated TTL/CMOS compatible outputs, capable of sourcing or sinking 15 milliamps. They are active during downloads and tri-stated during emulation.

11. Apply power to the target system.

12. Run the loader program. We wrote the loader as a command line driven routine rather than an interactive one. This allows you to run the program from a batch file without intervention. It can be added directly to your Compile-Link-Locate batch file. Enter "BLD08" without parameters to see the parameter syntax.

All parameters are optional and can be listed in any order. The Printer Port is specified by base address rather than LPTx. This allows the use of multi-port parallel cards that do not map directly into LPT1-LPT3. Most BIOS/DOS compatible printer ports are located at one of the following I/O addresses: 378, 3BC, or 278.

ADDITIONAL NOTES

1. Most linker/locators will generate Binary files quicker than Intel Hex or Motorola 'S' files. Binary files are more compact (if your EPROM is over 40% full) and therefor download quicker. With that in mind, we wrote the BLD08 program to use Binary files. If your Linker or Locator will only generate HEX files, you will need to convert them to Binary before downloading them. The UTILITY directory on the DISK contains a utility that will do this conversion for you. HEX2BIN.EXE will accept INTEL and MOTOROLA HEX files in 8 16 and 32 bit addressing formats and convert them to binary. Type HEX2BIN (without parameters) to view the parameter syntax.

2. The RESET and /RESET outputs are driven by TRI-STATED devices. They are active during the download only.

3. The Emulator (and therefor the target) must be powered before the object code can be down-loaded into it. If you use a battery-backed-SRAM, it will retain its information when power is removed. However, power MUST be applied to the EE08 before new data can be downloaded into it.

4. The pin header on the EE08 is documented below:

PIN NUMBER	FUNCTION	NOTES
1	RESET	pin closest to LED

2	/RESET	
3	/WRITE	from TARGET
4	/WAIT	to TARGET
5	/REQUEST	to TARGET
6	/GRANT	from TARGET
7	n/c	
8	n/c	pin closest to SRAM

TROUBLESHOOTING TIPS

The BLD08 program does error checking before and (optionally) after thm download. The possible error messages and possible solutions are listed below.

1. "A PRINTER PORT WAS NOT FOUND AT xxxx". At the beginning of the download, the program verifies that the device at the selected port address appears to be a printer port. This error is printed if there is no device at this address or the device does not respond like a printer port. Use the command line parameter "/P:xxx" to configure BLD08 for the base address of a valid printer port. The most common port addresses are 3BC, 278 and 378.

2. "ERROR - THE EE08 IS NOT RESPONDING ....". After the printer port is verified, BLD08 check to see if an EE08 is connected and communicating properly. This message is generated if the EE08 fails to respond properly. If you receive this message, check the following items:

   1. Verify that the EE08 is plugged into the printer port that was specified.
   2. Verify that the cable is connected properly
   3. Verify that the target is powered-up during the down-load
   4. Verify that the EE08 is plugged in properly
   5. Verify that J8 is jumpered properly for the target socket size.
   6. If you extended the download cable, try removing it.
   7 If you do buy a longer modular cable, be sure to purchase one that is wired the same as the one provided (8 conductor, pin 1 to pin 1). Note that modular cables can be purchased in straight through or cross-over configurations.
   8. Try a different printer port.

3. Once the port is verified and the EE08 responds properly, the download is completed. If the "/V" flag is used on the command line, the contents of the EE08 is then compared to the contents of the file that was just down-loaded. If a single byte is different, BLD08 will stop and report a "VERIFY ERROR AT xxxx". If you receive this error message, check the following:

   1. Verify that J3 is jumpered properly for the size of SRAM installed.
   2. Verify that the download file is NOT larger than the SRAM.
   3. Have your SRAM checked, or try another one.
   4. If you extended the download cable, try removing it.
   5. If you do buy a longer modular cable, be sure to purchase one that is wired the same as the one provided (8 conductor, pin 1 to pin 1). Note that modular cables can be purchased in straight through or cross-over configurations.
   6. Try a different printer port.

4. If the target does not respond as expected after the download, and you DID NOT receive any error messages, check the following:

   1. Verify that the DIP switches are set properly.
   2. Verify that J3 and J8 are set properly.
   3. Verify that the download file will fit in the EPROM being emulated.

4. Verify that the EE08 is plugged into the target properly.
5. Verify that the target is supplying a full clean +5Volts to the EE08.
6. Verify that the target has adequate power supply by-passing (particularly if it is a 2 layer or wire-wrapped board). The EE08 requires about twice as much current as the EPROM it is emulating.
7. Verify that the download file is a BINARY file. Use HEX2BIN to convert it if necessary.
8. If you used a HEX to BINARY conversion program, verify that you specified the REAL physical starting address of the EPROM for the conversion.
9. If you used the "/O" parameter for the download, verify that you really needed this option. It is rarely used.
10. If you are using a target socket extension cable, try removing it. Cables tend to add reflections, crosstalk and additional capacitive loading to the target.

If you need additional technical assistance, we can be reached at:

Technical *Solutions*
P.O. BOX 462101
Garland TX. 75046-2101
Voice: (214) 272-9392 FAX: (214) 494-5814

Please be prepared with the following information:

- EPROM being emulated
- DIP switch and jumper settings
- SRAM being used.
- Command line parameters being used
- EXACT (if any) error messages generated
- Target information (CPU,SPEED,strange memory maps...etc.)

ADVANCED INSTALLATIONS:

This section documents the various arbitration schemes supported by the EE08 and the proper installation to use them. WE HIGHLY RECOMMEND VERIFYING THE BASIC INSTALLATION FIRST. There are 6 different basic methods supported(with numerous variations or combinations). Each are discussed below.

ABSOLUTE :
If no arbitration is selected, the EE08 will ALWAYS arbitrate for the HOST. This is the method used by the download program. If the target and HOST collide, the HOST will complete a valid cycle and the target will retrieve garbage. The RESET lines can be used to hold the target in reset during the operation. if desired. This is the fastest method of accessing the EE08 because it does not have to arbitrate for each access. If your application will need to be reset after the access (like in a code change) or if an occasional miss -read of the data is acceptable (as in some look-up tables) then this is the fastest and simplest method, requiring no extra connections.

CONNECTIONS:
- /REQUEST jumpered to /GRANT
- /WAIT - n/c
- /RESET or RESET connected to TARGET reset (OPTIONAL)
- /WRITE connected to TARGET /write (OPTIONAL)

SEMAPHORE BYTE - AVOIDANCE:
The EE08 contains a fully dual-ported byte of memory. This byte can be accessed by the TARGET and the HOST at any time without fear of a collision. A command-line flag allows you place this byte at offset 0 in the EE08 memory space or at offset 0x7ffff. The downloader moves it high by default (possibly out of reach by your target). This SEMAPHORE byte can be used in several ways. The diagram below documents the contents of this byte.

SEMAPHORE BYTE DEFINITION

B7	B6	B5	B4	B3	B2	B1	B0
T1	T0	H2	H1	H0	C2	C1	C0

C0..C2	:	COUNTER (READ ONLY)
H0..H2	:	HOST CONTROL (Only the HOST can write, both can read)
T0,T1	:	TARGET CONTROL (Only the TARGET can write, both can read)

In simple avoidance, the HOST accesses the EE08 at will. The target watches the counter bits to determine when it is safe to access the memory, thereby avoiding a collision. The counter bits are actually the bit counter for the serial download from the host. The HOST will actually READ or WRITE to the memory IMMEDIATELY AFTER THE COUNTER BITS ADVANCE TO xxxxx111B. If the target NEVER accesses the memory space when the counter bits are set, it will never collide with the HOST. This method does not require additional connections to the target. Naturally, this will NOT work if the target is executing out of this same memory space because the code fetches would not watch the counter bits.

```
CONNECTIONS:
        /REQUEST jumpered to /GRANT
        /WAIT - n/c
        /RESET or RESET connected to TARGET reset (OPTIONAL)
        /WRITE connected to TARGET /write (OPTIONAL)
```

SEMAPHORE - HANDSHAKING
The control bits within the semaphore byte can be used to implement a handshake protocol between the HOST and TARGET. For example, The HOST could set H0 to indicate that he wants control of the memory space and likewise the target could set T0 to indicate his need of the space. If each side respects the other's, request no collisions will occur. Since the hardware cannot guarantee atomic read-write operations, the algorithms must. If both sides implement the following pseudo-code, safe access will be guaranteed.

```
ACCESS()
    {
      DO
         { clear SUCCESS                    // assume failure
            if ( target bit not set)
          { set my bit
                 if  (target bit still not set)
                    do read(s) and/or writes(s)
                    set SUCCESS
             else
                    clear my bit ;
                    clear SUCCESS
          }
         }  until SUCCESS
    }
```

This methods requires no additional connections to the target. The target can execute out of this memory space during the arbitration and during the time the target owns the space. However, once the HOST gain control of the space, the target must execute out of other memory space to avoid fetching garbage in the event of a collision. Since the TARGET does not know when the HOST will win arbitration, this method is not too useful if the TARGET must execute out of this space. A better approach is presented below:

ALGORITHM 2:
PARK the arbitration on the TARGET port, make the HOST ask for access. In this approach, The HOST requests access to the memory space by setting his bit and waiting for the TARGET to grant access by setting his bit. The TARGET will only do so when it is safe for him (he is executing out of some other memory space) for the duration of the HOST access. This slows down the HOST accesses but gives the TARGET full control.

```
HOST:
        read_08_byte( address)
                        set host request bit
                        wait for target granted bit
                        data = read(address)
                        clear host request bit
                        return(data)

        TARGET:
                POLL:
                        if (his request bit set)  and (I feel generous)
                                call GRANT


                GRANT:     // this routine MUST be outside of the memory space
                        set my granted bit
                        wait for his request bit to be cleared
                        clear my granted bit
                        return
```

CONNECTIONS:

- /REQUEST jumpered to /GRANT
- /WAIT - n/c
- /RESET or RESET connected to TARGET reset (OPTIONAL)
- /WRITE connected to TARGET /write (OPTIONAL)

SEMAPHORE - HANDSHAKE with interrupts

This approach is similar to the last one. The main difference is that the EE08 /REQUEST line is used to interrupt the TARGET when access is needed. The SEMAPHORE bits are then used to indicate when access is granted.   This eliminates the need for the TARGET to constantly poll for requests.

CONNECTIONS:

- /REQUEST connected to TARGET interrupt
- /GRANT - n/c
- /WAIT - n/c
- /RESET or RESET connected to TARGET reset (OPTIONAL)
- /WRITE connected to TARGET /write (OPTIONAL)

SOFTWARE UART:
Use the semaphore bits to implement a software UART. In this approach, the HOST never accesses the memory space directly (except during the initial download). Commands are sent serially through the semaphore byte to the TARGET. The target responds serially back through the semaphore byte. The routines could use 1 bit for data and another bit for handshake (data ready/data received). This method is particularly suited for micro controllers that can not execute out of other memory space, do not have ready/wait lines and do not have bus request/grant lines.

Since the memory space is not being accessed directly by the HOST, there is no need for arbitration. The EE08 /request and /grant lines are free for other uses. The UTIL08.LIB contains routines to use these lines directly. The /request line could be used to interrupt the TARGET when a valid data bit is ready from the HOST. Likewise the TARGET can assert the /grant line when it has valid data ready. The /grant line can be polled by the HOST. It is also connected to the interrupt line on the printer port card (if enabled). The HOST can take advantage of this if an Interrupt Service Routine is implemented. If these lines are used, the UART routines could transfer 2 data bits at a time in interrupt mode.

> **NOTE: We intend to provide source and executables of examples of a software UART and interrupt routines in the near future. If these have been distributed since the printing of this manual, the code and documentation will be on the distribution diskette.

CONNECTIONS:

- /REQUEST connected to TARGET interrupt (OPTIONAL)
- /GRANT - connected to TARGET control bit (OPTIONAL)
- /WAIT - n/c
- /RESET or RESET connected to TARGET reset (OPTIONAL)
- /WRITE connected to TARGET /write (OPTIONAL)

WAIT-STATES:
If the TARGET has wait-state circuitry, this can be used to effect arbitration. The /WAIT line on the EE08 is connected to the /WAIT line on the TARGET. The EE08 will wait for any TARGET access to complete before starting its access. If the TARGET then starts another access before the HOST access is complete, it will be held off with wait-states until the HOST access is complete. At that time, the /WAIT line is released and the TARGET is allowed to complete its access.

The /WAIT line is ACTIVE LOW, OPEN COLLECTOR.

CONNECTIONS:

- /REQUEST jumpered to /GRANT
- /WAIT - connected to TARGET /wait or ready
- /RESET or RESET connected to TARGET reset (OPTIONAL)
- /WRITE connected to TARGET /write (OPTIONAL)

REQUEST-GRANT:
If the TARGET has bus mastering capabilities, this can be used for arbitration. The /REQUEST line from the EE08 can be connected to a bus request line on the TARGET. When the EE08 needs to access the EE08 memory space, it asserts /REQUEST. This tells the TARGET that a master wants control of the system. The TARGET arbitration circuitry holds off the local processor and generates a /GRANT signal. This signal is connected to the EE08 /GRANT pin. Since the processor is being held-off, it can not access the EE08 memory space and will not collide with the HOST access. When the HOST has completed its access, it will release the /GRANT line, relinquishing control to the TARGET processor.

/REQUEST and /GRANT are an ACTIVE LOW, TTL compatible signals.

CONNECTIONS:

- /REQUEST connected to TARGET bus request
- /GRANT connected to TARGET bus grant

/WAIT - n/c
/RESET or RESET connected to TARGET reset (OPTIONAL)
/WRITE connected to TARGET /write (OPTIONAL)

SUMMARY:

The SOFTWARE UART implementation provides hardware independent communications between the HOST and TARGET. It requires NO TARGET connections and will work with any TARGET. However, it requires software on both ends of the link.

The /REQUEST-/GRANT or WAIT-STATE implementations are completely transparent to the software on both ends and allow both ends completely random access to the entire memory space at will. However, both methods depend on hardware support from the TARGET.

Between these two extremes are the various SEMAPHORE implementations. These rely on minimal software, but require that the TARGET can execute from code space outside of the memory space occupied by the EE08 during the HOST accesses.

All targets should be able to implement one of the listed methods, or a variation on one. If you have a unique situation that precludes the use of ANY of these methods, we would like to hear about it. We may be able to suggest an alternative solution.

ADDITIONAL NOTES:

The active levels and drive types (TTL/OC) of /REQUEST, /GRANT and /WAIT were selected for common applications. We realize that MURPHY'S law still rules and regardless of how we configure them, your needs will be different. We used a programmable device to implement these signals and socketed it so that the EE08 could be customized by the end-user. The distribution diskette contains several JEDEC files for different configurations. Read the JEDEC.DOC file for descriptions. If you require a different configuration, we would be happy to compile a custom JEDEC file for you. Of course you can always use external logic to convert any of the signals to what you need.

LIBRARY REFERENCE:

The routines in the UTIL08.LIB librarys were compiled under the small model with BORLANDC++ 3.1 and MICROSOFT VISUAL C++1.0. However, they do not use any of the C++ features (just plain old C).

A description of each procedure follows.

```
//////////////////////////////////////////////////////////////////
//
// init08 - initializes and verifies the printer port. Also sets the
//        semaphore byte high or low, as requested.
//
// RETURNS:     0 -> no errors
//              1 -> printer port not found
//              2 -> EE08 not found
//
//NOTES: should be the first routine called
//////////////////////////////////////////////////////////////////
unsigned char init08(unsigned int port_addr,unsigned char sym_addr)


//////////////////////////////////////////////////////////////////
//
// TARGET_RESET - Asserts or De-asserts reset to the target.
//          options are : 0 -> release reset, non zero -> assert reset
//
//RETURNS: nothing
//
//////////////////////////////////////////////////////////////////
void target_reset(unsigned char reset_flag)


//////////////////////////////////////////////////////////////////
//
// ARB_METHOD - Sets the arbitration method to be used. Options are
//           AUTOMATIC (1) or MANUAL (0).
//
// RETURNS: nothing
//
// NOTES: AUTOMATIC should ONLY be used if /REQUEST - /GRANT or
//           /WAIT is connected.  MANUAL should be used if software is being used
//           for avoidance or if manual control of this line is desired.
//////////////////////////////////////////////////////////////////
void arb_method(unsigned char arb_method)


//////////////////////////////////////////////////////////////////
//
// BUS_REQUEST - Manually asserts (1) or de-asserts(0)  the bus request bit
//            for manual arbitration or for other uses of this
//            control pin.
//
// RETURNS: nothing
//
//////////////////////////////////////////////////////////////////
void bus_request(unsigned char rq_flag)
```

```
//////////////////////////////////////////////////////////////////
//
// BUS_GRANTED - Returns true if the target has granted access to the
//              bus to the HOST.  Used for manual arbitration or other
//              uses of this input pin.
//
//////////////////////////////////////////////////////////////////
unsigned char bus_granted(void)


//////////////////////////////////////////////////////////////////
//
// READ_BYTE - Reads a byte from the requested offset in the EE08.
//             Arbitration is used if AUTO Arbitration is in effect or if MANUAL
//            arbitration is being used and BUS_REQUEST is active
//
//////////////////////////////////////////////////////////////////
unsigned char read_byte(unsigned long int offset)


//////////////////////////////////////////////////////////////////
//
// WRITE_BYTE - Writes the given byte to the given offset in the EE08
//              Arbitration is used if AUTO Arbitration is in effect or if MANUAL
//            arbitration is being used and BUS_REQUEST is active
//
// RETURNS: nothing
//
//////////////////////////////////////////////////////////////////
void write_byte(unsigned char wdata,unsigned long int offset)


//////////////////////////////////////////////////////////////////
//
// WRITE_NEXT - Writes the given byte into the next location in the
//              EE08.  This command is only valid following a "write_
//              byte" or another "write_next" command.
//             Arbitration is used if AUTO Arbitration is in effect or if MANUAL
//            arbitration is being used and BUS_REQUEST is active
//
//RETURNS: nothing
//
//////////////////////////////////////////////////////////////////
void write_next(unsigned char wdata)


//////////////////////////////////////////////////////////////////
//
// READ_NEXT - Reads the next byte from the EE08.  This is only valid
//             following a "READ_BYTE" or another "READ_NEXT" command.
//             Arbitration is used if AUTO Arbitration is in effect or if MANUAL
//            arbitration is being used and BUS_REQUEST is active
//
//////////////////////////////////////////////////////////////////
unsigned char read_next(void)



//////////////////////////////////////////////////////////////////
//
```

```
// READ_SYM - reads and return{ the current value of the semaphore
//
//////////////////////////////////////////////////////////////////
unsigned char read_sym(void)


//////////////////////////////////////////////////////////////////
//
// WRITE_SYM - Writes the given byte to the semaphore byte.
//
//RETURNS: the value of the semaphore byte after the write
//
//////////////////////////////////////////////////////////////////
unsigned char write_sym(unsigned char wdata)


//////////////////////////////////////////////////////////////////
//
// HEX2BIN - Translates the (1 to 8 character) HEX string pointed
//                        to by the parameter into its binary equivalent.
//
//NOTES: 1. *hex MUST be a valid NULL terminated string.
//            2. Only the last 8 characters of a string are valid
//            3. Non-HEX characters get converted to 0
//
//////////////////////////////////////////////////////////////////
unsigned long int hex2bin(unsigned char *hex)
```