

# MicroMonitor Disaster Recovery Using the BDI2000

## 1.1.1 Introduction

A “disaster” in this context is the embedded system being in a state that doesn’t allow it to successfully start up. In a MicroMonitor (uMon) based system, there are several phases of startup... processor boot, monrc execution and application autoboot. As a result, there are several different situations that can cause an unsuccessful startup...

- ❑ **Corrupt Boot Flash:** The boot flash, or the memory in which the uMon executable resides, has been modified (or corrupted). Depending on what was corrupted, the monitor may or may not be able to start up the target. This is typically caused by illegal accesses to the flash sectors that are dedicated to uMon’s executable image. If this flash is not protected, then it is susceptible to being overwritten by devious or buggy code (or commands). Yes, the flash write/erase algorithms make the memory space a little bit more difficult to modify, but depending on the type of device used, the algorithm can easily be run by mistake on the flash memory space.
- ❑ **Illegal Command or Executable in ‘monrc’ File:** Prior to the release of version 1.0 of uMon, executables (scripts or binary) could be run from the monrc file. Since the monrc file is automatically run by uMon *during* uMon startup, the user had to be very careful about what was put into monrc. As of release 1.0 of uMon, the monrc file cannot run any other executable, and certain commands (i.e. ethernet related) are also disabled during monrc execution.
- ❑ **Bad Non-Interruptible Autoboot File:** If an application is loaded onto the target (in TFS) and it is configured as non-query autoboot (‘b’ flag in TFS), then it better run successfully. If not, even though the startup is doing exactly what it is supposed to do, the target is rendered useless because the startup sequence is not abortable and the application is bad.

This document walks the user through the process of restoring a target as a result of a “disaster”. The following steps will walk the user through the process:

1. Connect the BDI2000 to the target hardware.
2. Initialize the target CPU and SDRAM using a BDI2000 script.
3. Using the BDI2000, download a version of MicroMonitor to run out of RAM on the target.
4. Interface to the RAM-based MicroMonitor using TFTP (or Xmodem) to program the boot flash with a new MicroMonitor image.

The procedure outlined in this text references the Cogent Single Board (CSB) series of embedded systems from Cogent Computer Systems (<http://www.cogcomp.com>); however, note that this procedure would be the same regardless of the target hardware. The target system is hereon referred to as “CSB” or “CSBXXX” where XXX is a board ID for one of the CSB series. We will assume that the target is dead (i.e. not responsive), and that it needs to be booted with an external JTAG type of tool. The BDI2000 development tool from Abatron (<http://www.abatron.ch>) is used throughout this text as the tool that steps the target through it’s initial boot sequence.

## 1.1.2 The Overall Setup

This procedure may be invoked from files that are created by the user or files that are provided to the user. If they are pre-built files, then just use them; however, if you are building these files from your own version of the MicroMonitor source tree (release 1.0 or above), then run the following “make” command under the target directory:

```
make clobber depend boot rundisable ramtst clean
```

Running make with this exact sequence will rebuild your target’s boot image (boot.bin), then it will rebuild the ram-based image (ramtst.elf) with TFS’s “run” capability disabled. This is preferable for disaster recovery so that the image loaded in ram will not automatically run any files (that may be corrupted).

The setup to be discussed requires a PC; a BDI2000, a CSB board and a few tools that come with the uMon package available from Microcross (<http://www.microcross.com>), tftp.exe, moncmd.exe and newmon.exe. Referring to Figure 1, the PC's ethernet and serial ports are each connected both to the BDI2000 and the CSB. Note that this configuration is ideal, the serial port connection between the BDI2000 and the PC is only needed for initial configuration of the BDI2000, so depending on the state of your BDI2000, this may not be necessary. Also, it isn't absolutely necessary to have the ethernet connection between the CSB and PC. This connection is only needed if the transfer between the CSB and the PC is done via TFTP (if Xmodem is used, then this is done via the serial port).

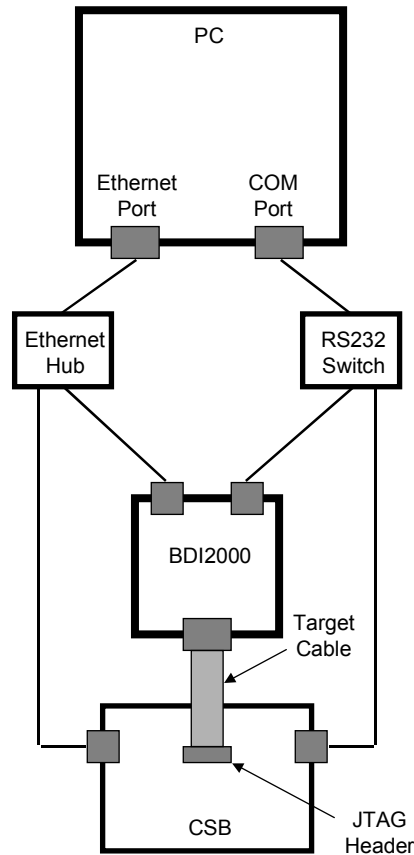


Figure 1: Complete Setup

### 1.1.3 Using the BDI2000 to restore a CSB's boot flash:

It is assumed that the user is familiar with the BDI2000 and its interface with the CSB. The following discussion is general with regard to a particular CSB model number. For each CSB port there is a directory and under that directory there are board-specific files used by this disaster recovery procedure:

- ❑ **bdi\_hookup.jpg**: A picture showing the connection between the BDI2000 and the CSB system.
- ❑ **bdi2000.cfg**: A script file run by the BDI2000 to prepare the CSB for download of a ram-based monitor image.
- ❑ **bdi2000.regs**: A register description file used by the BDI2000.
- ❑ **build\_CSBXXX\ramtst.elf**: The ELF file that contains the version of the monitor that will be loaded into RAM on the CSB.
- ❑ **build\_CSBXXX\boot.bin**: The binary image of the monitor that is to be loaded into the CSB's boot flash.

Note that the point of these files is to restore the target from the “disaster” state. The image that is actually burned into the target may not be the latest; however, the target will then be sane enough to be updated to the latest version using the ‘make newmon’ command in the target’s port directory (assuming you have source code for the port).

### 1.2 Configure the BDI2000 for the CSB’s microprocessor

The BDI2000 is capable of interfacing to several different processor types. The first step is to make sure that the BDI2000 is loaded with firmware and logic that interfaces to the CPU on the CSB that is to be restored. Since this procedure may differ with different revisions of the BDI2000 (and different CPUs), the user must refer to the BDI2000 user manual for this basic setup procedure.

The second phase of BDI2000 configuration is to prepare it for the network it will reside in. For the sake of this discussion, the BDI2000 is configured as shown in the diagram of Figure 2.

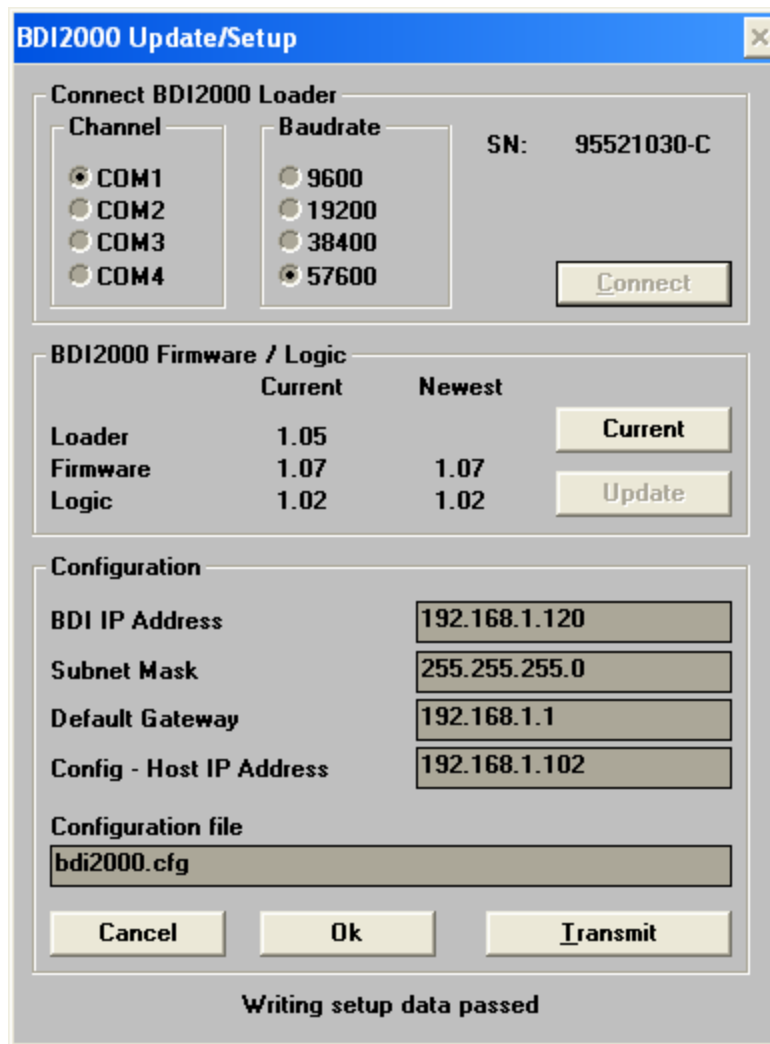


Figure 2: BDI2000 Setup

The BDI2000 is configured to ask for the file “bdi2000.cfg” which is located in the same directory that the tftp server will be run in. The following steps assume that the BDI2000 has been properly configured for the correct CPU and your network.

### 1.3 Physically connect the BDI2000 to the CSB

Using the BDI2000-to-target cable specified for the CPU that is on the CSB, connect the BDI2000 to the CSB as shown in the hookup.jpg picture for the particular CSB. For example, if your target system is the CSB350, then refer to csb350/bdi\_hookup.jpg. Note that in many of the pictures, the CSBXXX will be attached to some carrier board or port expansion board optionally available with many of the CSB

target systems. As always, be aware of static discharge, and make sure that power is off on both the BDI2000 and the CSB when connecting the cable.

#### 1.4 Physically connect the PC's COM port to the CSB

Using a terminal emulator (e.g. HyperTerminal), connect to the CSB's serial port. For the majority of CSB targets, the uMon console port is configured for 38400, no parity, 1 stop bit and 8-bit characters. A null-modem is also usually required. With this connection established, note that since the target is dead at this point, there will be no communication. This link is established here so that after the ram-based monitor image is downloaded and started, you will then see the uMon console prompt and possibly the startup information.

#### 1.5 Prepare for the BDI2000's TFTP Requests

Once the physical connection between the BDI2000 and the CSB has been established, both devices can be powered up. When the BDI2000 is turned on, it will generate a TFTP request to the host that it was configured for. In our case, that host is our PC at IP address 192.168.1.102 (see Figure 2). This means you need to start up a TFTP server that will respond to the BDI2000's request. The BDI2000 comes with a basic TFTP server (tftpsrv.exe); however, we will use the server that comes with the MicroMonitor toolset (tftftp.exe). Start the server up in a console window at the directory level that has the configuration file needed to interface with the CSB (bdi2000.cfg). This file is part of the CSB's MicroMonitor port, so it will be in that port's directory. Note that this file contains a hard-coded IP address of the PC that it will communicate with. Adjust this address in your bdi2000.cfg file to match the IP address of your PC...

```
[HOST]
IP      192.168.1.102      <- Modify this address to match your system.
FILE    build_CSBXXX/ramtst.elf  <- 'XXX' is the CSB board ID
```

#### Listing 1: Snippet from the bdi2000.cfg File

#### 1.6 Startup the BDI2000 and Note the Initial File Transfers

Start up the TFTP server "tftftp srvr" in the console window. Next powerup the BDI2000 and the CSB. As soon as the BDI2000 starts up it will ask for the bdi2000.cfg file by issuing a TFTP request to the PC. The transaction will be logged to the console window as follows:

```
PC> tftftp srvr
TFTP transferring: bdi2000.cfg (octet)
TFTP bdi2000.cfg transfer completed
PC>
```

#### Listing 2: Output generated by the TFTP Server After BDI2000 Startup

At this point the BDI2000 has pulled down the configuration file from the PC, and its time to connect to the BDI2000 via telnet<sup>1</sup>. Referring once again to Figure 2, the BDI2000 is configured to use IP address 192.168.1.120<sup>2</sup>; hence, the command "telnet 192.168.1.120" should connect your PC to the BDI2000's telnet-based command interface. Once connected, the BDI> prompt will be present and the following commands can be executed:

```
BDI>load
Loading build_CSBXXX/ramtst.elf , please wait ....
Loading program file passed
BDI>go
BDI>
```

#### Listing 3: Commands (and Responses) at the BDI2000 Telnet Interface

Following the 'load' command, the window in which the TFTP server is running will also indicate the transfer of the build\_CSBxxx/ramtst.elf file. Note that depending on the server used, the message

<sup>1</sup> Note that the TFTP server should be left running and the telnet connection should be established in a different window on the PC.

<sup>2</sup> Obviously the IP address used will depend on your network.

“Incoming Error NNN: <>” can be ignored. The important thing to note in the telnet connection is that the transfer succeeded.

At this point, the console port of the CSB should have generated at least a prompt, but most likely it also displays the uMon startup information similar (but not identical) to that shown in Listing 4.

```
MAC store abort: etheraddr[] (0x200020) in RAM
TFS Scanning //A/...
TFS Scanning //B/...
monrc: tfs facility not available
MICRO MONITOR
CPU: AMD Au1100
Platform: CSB350
Release 1.0.1, Built: Dec 29 2004 @ 17:42:30
Monitor RAM: 0xa18392f0-0xa1854ab8
Application RAM Base: 0xa0000000
MAC: 00:00:00:00:00:00
IP: 0.0.0.0
uMON>
```

#### Listing 4: Output at CSB’s Console after BDI2000 Load Command

The initial “MAC store abort” line may or may not be present. If present, it occurs because the monitor is RAM based; hence, the step that would normally program the MAC address to flash is aborted. This is expected. An additional unexpected line may be output, depending on the state of flash when the new monitor is put in RAM... “monrc: tfs facility not available”. This line is generated if the RAM based monitor scans flash and finds a valid monrc file in TFS. If the ram-based monitor was built with “rundisable”, then TFS cannot run any scripts or executables. This may not be necessary for your target, but it does satisfy the case where the autobootable file is corrupted; hence, it should not be automatically run. If you don’t want this, then rebuild the ramtst image by running “make clean ramtst”, without previously running “make rundisable” as described in section 1.1.2 above.

The remaining lines of Listing 4, are the typical MicroMonitor reset output. Depending on how the monitor was configured and the version, this header content may vary or may not even be seen. At a minimum, the uMON> prompt will be generated, and the CSB should now respond to commands at the console.

### 1.7 Unlock all flash sectors

Depending on the state of the flash at the time of the corruption, it may be necessary to unlock sectors so that they can be erased and re-written. To do this we must determine the sector range that the flash banks occupy and then we issue a flash unlock command to that range. To determine the sector range of the flash banks, issue the command “flash info”, then note the first and last sector and then issued the flash unlock command for that entire sector range...

```
uMON>flash info
Device = INTEL-28F640
Base addr   : 0xbf800000
Sectors     : 64
Bank width  : 2
Sector      Begin          End          Size        SWProt?    Erased?
0           0xbf800000 0xbf81ffff 0x020000    no         no
1           0xbf820000 0xbf83ffff 0x020000    no         yes
2           0xbf840000 0xbf85ffff 0x020000    no         yes
3           0xbf860000 0xbf87ffff 0x020000    no         yes
4           0xbf880000 0xbf89ffff 0x020000    no         yes
5           0xbf8a0000 0xbf8bffff 0x020000    no         yes

... (output for sectors 6-58 are omitted here)

59          0xbff60000 0xbff7ffff 0x020000    no         yes
60          0xbff80000 0xbff9ffff 0x020000    no         yes
61          0xbffa0000 0xbffbffff 0x020000    no         yes
62          0xbffc0000 0xbffdffff 0x020000    no         yes
```

```
63 0xbffe0000 0xbfffffff 0x020000 no yes
uMON>
uMON>flash unlock 0-63
uMON>
```

### Listing 5: Flash Unlock Procedure

Referring to Listing 5, the output of the “flash info” command shows the sector range starting with 0 and ending with 63. Note that this information is very target dependent, so don’t assume that this output is the same for all CSB targets. The next command “flash unlock 0-63” actually unlocks this entire sector range. This unlock step may take a few seconds, and should be accompanied by a small “ticker” indicating that progress is being made. If the ticker doesn’t move, this indicates that for some reason the unlock algorithm is hanging (this shouldn’t happen).

Almost done! The final step is to take the flash based version of MicroMonitor and install it. This can be done via ethernet using TFTP or RS-232 using Xmodem...

## 1.8 Install the Flash-Based Version of MicroMonitor Using Ethernet

To use UDP to transfer the file to the target, the target must be configured to communicate over the network. This is done by configuring the monitor’s IP information and using TFTP. Issue the following commands at the CSB’s console:

```
set IPADD 192.168.1.110
set GIPADD 192.168.1.1
set NETMASK 255.255.255.0
ether on
```

### Listing 6 : Network Configuration Commands

Obviously use the appropriate addresses and mask for your network. Now, assuming you’ve got the CSB connected to the network, you should be able to issue a ping to and from the board. At the console (HyperTerminal) connection of the CSB, issue the command “icmp echo \$GIPADD”. This command is essentially the same as a ping (an ICMP echo). If the CSB is correctly configured, then the immediate response will indicate that the device at the IP address stored in the GIPADD shell variable is alive...

```
uMON> icmp echo $GIPADD
192.168.1.1 is alive
uMON>
```

### Listing 7: ICMP Echo Transaction Initiated at Target

Also, at the console window of your PC issue the command “ping 192.168.1.110”. The interaction should be similar to the text of Listing 8.

```
PC> ping 192.168.1.110

Pinging 192.168.1.110 with 32 bytes of data:

Reply from 192.168.1.110: bytes=32 time<1ms TTL=60
Reply from 192.168.1.110: bytes=32 time<1ms TTL=60
Reply from 192.168.1.110: bytes=32 time<1ms TTL=60
Reply from 192.168.1.110: bytes=32 time<1ms TTL=60

Ping statistics for 192.168.1.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
PC>
```

### Listing 8: Ping Transaction Initiated at PC



If the MAC address displayed is not the one you want, then backspace over it and re-enter your MAC address. Then hit return<sup>3</sup>.

```
Configuring '00:30:23:72:80:01' as MAC address, ok?
```

#### **Listing 11: Confirmation of MAC Address to Burn to Flash**

Respond with 'y' (and return) if the MAC address is correct, and that will allow the monitor to completely start up...

```
MAC address burned in at 0xffff80020
MICRO MONITOR
CPU: PowerPC 405GP
Platform: CSB472
Built: Oct_25,2004@20:15:09
Monitor RAM: 0x003000-0x01c568
Application RAM Base: 0x01d000
MAC: 00:30:23:72:80:01
IP: 192.168.0.214
uMON>
```

#### **Listing 12: MicroMonitor Reset Message from Flash**

The actual details in the header will vary, but the important thing is that now the BDI2000 can be removed and the CSB target will boot up standalone. All done!

### **1.9 Install the Flash-Based Version of MicroMonitor Using RS-232**

As an alternative to the above Ethernet-based procedure for burning the image, the Xmodem protocol can also be used to transfer and burn in the image. The steps are slightly different, but the end result is the same: the boot flash is reprogrammed and the CSB can once again run standalone. This Xmodem-based procedure does not require any of the ethernet configuration stuff described in section 1.8 above; however, since it is transferring the data over RS-232, the transfer rate is much slower.

With HyperTerminal running and the ram-based monitor started, the target simply needs to be told where the base of the boot flash is. The shell variable BOOTROMBASE holds this information; however, since the image running on the target is now in RAM, the value stored in the variable may reflect the monitor's starting point in RAM, so this needs to be correctly established. To determine the value to use, refer to the bdi2000.cfg file. *Hopefully*, at the top of this file there is a line of text that indicates the target's BOOTROMBASE setting, so assuming the value is 0xffff80000, then the command "set BOOTROMBASE 0xffff80000" will establish this. Having done that, the next step is to start up the Xmodem protocol on the target side, so issue the command "xmodem -B" at the console of the CSB (e.g. HyperTerminal). Depending on the terminal emulator used, once the xmodem -B command is issued, there may be some "weird" character output about once a second. This is the Xmodem protocol on the target side waiting for the Xmodem protocol to start on the host side. So, at the HyperTerminal user interface, initiate an Xmodem file transfer of the file build\_CSBXXX/boot.bin. This should stop the "weird" characters and now the transfer should be started. Upon completion of the transfer, the CSB will ask if it is ok to install the image, type 'y' to continue. A few seconds later the monitor should restart and the header indicating that it has rebooted will appear at the console. The actual details in the header will vary, but the important thing is that now the BDI2000 can be removed and the CSB target will boot up standalone. All done!

---

<sup>3</sup> Note that depending on the CSB target configuration, the MAC address burn step may not be part of the startup.