Table of Contents

─────────────────

1 Credits

2 Boot the scope with network access

3 Manufacturing Mode

4 Hardware Configuration

5 Debug Console

6 Linux shell access

7 Console logs

8 Example Console Log File

9 Interesting Commands

1 Credits

═════════

I learned much of this information from private conversations with

EEVBlog members. I'm just repeating the information.

For more details and to make contact with other curious people, visit

these threads: [MDO3000 Hacking] and [Someone has hacked MDO4000C?]

[MDO3000 Hacking]

https://www.eevblog.com/forum/testgear/mdo3000-hacking/

[Someone has hacked MDO4000C?]

https://www.eevblog.com/forum/testgear/someone-has-hacked-mdo4000c/

2 Boot the scope with network access

════════════════════════════════════

My scope comes up with a network address of 10.0.0.81

3 Manufacturing Mode

════════════════════

Before you start, use the scope's front panel to bring up the

Calibration page under the utility menu. You should see two other

menus along the bottom: "Signal Path" and "Factory".

To put the scope in manufacturing mode, connect to the SCPI shell at

port 4000. I'm on a recent mac (and they took telnet away), so I use

netcat in these examples.

┌────

│ $ nc -c 10.0.0.81 4000

│ Tektronix MDO4034B-3 Instrument Control Terminal Session

│ Control commands:

│ !t <timeout> : set the response timeout in milliseconds.

│ !d : send device clear to the instrument.

│ !r : read response from instrument.

│ !h : print this usage info.

│ Commands containing a ? are treated as queries and responses are read automatically.

│

│ Timeout is 10000 milliseconds

│ > :PASSW INTEKRITY

│ > :MFG:MOD 1

└────

As soon as you issue the :MFG:MOD command, you should see three more

menu items appear at the bottom: "Status Details", "RF IQ Vector" and

"Serial Console".

The easiest way to leave Manufacturing Mode is via SCPI:

┌────

│ > :MFG:MOD 0

└────

You can also turn it off via a new menu item in the GUI under Utility

> Self Test > Error Log.

4 Hardware Configuration

════════════════════════

While in Manufacturing Mode, goto Utility > Config > About > Hardware

Configuration

My MDO4034B-3 shows the following (Xs replace actual serial numbers):

┌────

│ Hardware Configuration

│

│ InstrumentSerial: XXXXXXX

│ Main Board Serial: XXXXXX

│ IO Board Serial: XXXXXX

│ Front Panel Serial: XXXXXX

│ Power Supply Serial: XXXXXXXX

│ Display Serial: XXXXXXXXXXXXX

│

│ Demux Info: 4 present, Version: 387D

│ Trigger Version: 2

│ Model (from resistors): 1 (4-channel, 350 MHz)

│ Sample Rate, Sample/sec: 5.000000e+09

│

│ Main Board Version: 11 (Rev: K)

│ Front Panel: Board: 16, FW: 53, CC: 6

│ Analog Front End: Board: RF\_3GHz\_BW\_1GHz, Rev: B, FW: 1032

│

│ CPU Version: B

│ U-Boot 2009.08 v1.07 Apr 19 2013

│ Linux 3.19.3 version 1.28

│

│ SDIA image: Version 141

│ View: bk\_cayman\_er\_rc1

│ Target: route66b\_scopeApp\_bk\_cayman\_er\_rc1.ppcex

└────

It's interesting to note that resistors are apparently used to

distinguish between different models and that my 350Mhz scope

(according to the sticker) has a 1GHz front end that's capable of

sampling at 5GS/s. Based on the hw info, it looks like a different

analog board is required to get a 6GHz spectrum analyzer.

5 Debug Console

═══════════════

These scopes have a "debug console" runs on port 1072. There's a

little bit of online help:

┌────

│ $ nc -c 10.0.0.81 1072

│ > ?

│ Commands available from the console are:

│ ? <command name>

│ DemoMgrStatus

│ acqBusTest [prt]:0

│ acqHsaUnitTest

│ acqInterpTest [prt]:0

│ acqMemLayout [bufType]:0 [recLen]:0

│ .

│ .

│ .

└────

The "utilConsoleRedirect" commands make sure you see the output of a

command. Without these, output probably goes to a UART somewhere.

┌────

│ > utilConsoleRedirect 1 1

│ StdOut is connected to debug

│ StdErr is connected to debug

│

│ > utilConsoleRedirect 2 1

│ StdOut is connected to debug

│ StdErr is connected to debug

│

│ >

└────

It's easy to leave the debug shell in a controlled way:

┌────

│ > quit

│ $

└────

6 Linux shell access

════════════════════

Use the "utilShell" console command to execute arbitrary shell

commands. UtilShell accepts a single argument and converts any

backslashes to spaces. I.e., to pass multiple words to a shell

command, use backslashes instead of spaces:

┌────

│ > utilShell uname\-a

│ string to shell: uname -a

│ Linux MDO4034B-3-XXXXXX.global.tektronix.net 3.19.3 V 1.28 Tektronix BK/CK Fri Sep 4 14:14:12 PDT 2015 ppc GNU/Linux

│ OK

│ >

└────

7 Console logs

══════════════

When the scope boots, it creates a log file. These are accessible in

the filesystem under /mnt/flash2/usr/local/nv:

┌────

│ > utilShell ls\-l\/mnt/flash2/usr/local/nv

│ string to shell: ls -l /mnt/flash2/usr/local/nv

│ -rw-r--r-- 1 root root 4187 Mar 31 11:44 consoleLog1.txt

│ -rw-r--r-- 1 root root 4188 Nov 29 10:53 consoleLog10.txt

│ -rw-r--r-- 1 root root 4188 Dec 3 13:22 consoleLog11.txt

│ -rw-r--r-- 1 root root 4188 Dec 18 21:14 consoleLog12.txt

│ -rw-r--r-- 1 root root 4188 Dec 25 11:57 consoleLog13.txt

│ -rw-r--r-- 1 root root 4188 Dec 26 11:21 consoleLog14.txt

│ .

│ .

│ .

│ -rwxr-xr-x 1 root root 60 Aug 28 2017 setup1.nfo

│ -rw-r--r-- 1 root root 97291 Aug 28 2017 setup1.set

│ -rw-r--r-- 1 root root 1646503 Aug 1 2017 userSaves.set

│ OK

│ >

└────

There are only 50 consoleLog\*.txt files at any particular time, so it

appears that the names wrap and files are re-used with each boot.

8 Example Console Log File

══════════════════════════

┌────

│ > utilShell cat\/usr/local/nv/consoleLog1.txt

│ string to shell: cat /usr/local/nv/consoleLog1.txt

│ errSetConsoleLogState() logging to /usr/local/nv/consoleLog1.txt

│ cfgInit

│ versionBuildFWVersionString(): TimestampString: 24-Oct-16 13:29

│ VersionFIRMWAREVERSIONversion: v3.22

│ Major ver num: 3 Minor ver num: 22

│ sysInit

│ execInit

│ hwInit

│

│ Main Board HW Rev: 0x0b

│

│ vertReprogramFeProc(): Platform Route66b fw 1032 filefw 1032

│ Front Panel Firmware update not needed

│ Current firmware 1032 >= 1032

│

│ Main Board HW ID: 0x07

│

│ AFE Board SW ID: 0x12

│ cfgGetRfHwInfo(): Contents of CfgRfHwInfo:

│ rfHwPresent = 1; rfFrontEndType = 2; rfAfeRev = 2

│ rfBw = 3e+09; rfLowBandStartFreq = 9000; rfAttenResolution = 1.000000

│ rfAcqMemSize = 2e+09

│

│ Main Board SW ID: 0x01

│

│ HFD144[0] ID\_REG = 0x00001440

│ HFD144[1] ID\_REG = 0x00001440

│ HFD144[2] ID\_REG = 0x00001440

│ HFD144[3] ID\_REG = 0x00001440

│ fanControlInit

│ Init ADT7476.

│ mitlInit

│ diagInit

│ diagRunEarlyPostDiags

│ ialInit

│ ialInit(): AFE id 0x12, rev 0x2, bI 8

│ calInit

│ Factory Checksum: Stored: 35540, Calculated: 35540 Spc CheckSum: stored: 27074 calculated: 27074 Factory RF Checksum: Stored: 30720, Calculated: 30720 Spc RF CheckSum: stored: 28115 calculated: 28115 Factory RF Vector Checksum: Stored: 33923, Calculated: 33923 dmxInit

│ Demux initialization

│

│ Dram Calibration results:

│ -------------------------

│ DramCal has PASSED on all Demuxs

│

│ Demux initialization complete

│ debugConsoleInit

│ diagRunPostDiags

│ Starting POST diags

│ Finished POST diags

│ dmxInitAfterDiagnostics

│ acqInit

│ autosetInit

│ clkInit

│ demoMgrInit

│ fpInit

│ extPwrInit

│ cmdInit

│ dsyInit

│ dsyMgrInit

│ dvmInit

│ hcInit

│ limInit

│ markInit

│ searchInit

│ cmdPeriodicSenderInit

│ maskInit

│ dspLibInit

│ mathInit

│ measInit

│ histogramInit

│ usbInit

│ fileUtilInit

│ measPackageInit

│ piInit

│ probesInit

│ refInit

│ rfInit

│ rfMeasInit

│ start\_rtlInit

│ trigInit

│ txtDsyInit

│ uiInit

│ utilInit

│ vertInit

│ vidPicInit

│ vxi11Init

│ waveAlertInit

│ wfmDsyInit

│ wfmMgrInit

│ busInit

│ calPowerupRun

│ mitlInit

│ acqInit

│ trigInit

│ afgInit

│ ---------------------- startScopeApp() running Start code ------------

│ execStart

│ mitlStart

│ acqStart

│ autosetStart

│ calStart

│ cfgStart

│ clkStart

│ cmdStart

│ diagStart

│ dsyStart

│ dvmStart

│ errStart

│ fpStart

│ extPwrStart

│ hcStart

│ hwStart

│ limStart

│ searchStart

│ cmdPeriodicSenderStart

│ maskStart

│ mathStart

│ measStart

│ measPackageStart

│ histogramStart

│ nvStart

│ piStart

│ probesStart

│ refStart

│ rfStart

│ rfMeasStart

│ start\_rtlStart

│ trigStart

│ start\_postTrigRtlStart

│ ialStart

│ txtDsyStart

│ uiStart

│ usbStart

│ fileUtilStart

│ utilStart

│ vertStart

│ vidPicStart

│ vxi11Start

│ waveAlertStart

│ wfmDsyStart

│ wfmMgrStart

│ busStart

│ afgStart

│ ---------------------- startScopeApp() running Run code --------------

│ execRun

│ autosetRun

│ calRun

│ cfgRun

│ clkRun

│ cmdRun

│ diagRun

│ dsyRunInitWfmPalette

│ dvmRun

│ errRun

│ fpRun

│ extPwrRun

│ hcRun

│ hwRun

│ limRun

│ wfmMgrRun

│ searchRun

│ cmdPeriodicSenderRun

│ maskRun

│ mathRun

│ measRun

│ histogramRun

│ measPackageRun

│ nvRun

│ piRun

│ probesRun

│ refRun

│ rfRun

│ rfMeasRun

│ start\_rtlRun

│ trigRun

│ ialRun

│ txtDsyRun

│ uiRun

│ usbRun

│ utilRun

│ enetRun

│ fileUtilRun

│ vertRun

│ vidPicRun

│ vxi11Run

│ waveAlertRun

│ wfmDsyRun

│ busRun

│ debugConsoleRun

│ mitlRun

│ afgRun

│ OK to connect by: telnet MDO4034B-3-XXXXXX.global.tektronix.net 1072

│ --------------------------------------------------------

│ startScopeApp() complete; duration = 15.348500 seconds

│ =================================================================

│ PID to Task info written to /tmp/threads.txt

│ Power Up Completed at 11:44:06

│ 11:44:06 Enter 'ctrl-\' to quit scopeApp

│

│ 11:44:06 SocketServerService: Socket server daemon started on port 4000.

│ Protocol: Raw

│ 11:44:08 SDIA version 0x141, Trinity, running calSdiaRunClkSync()

│ 11:44:08 calSdiaRunClkSync PASSED

│ 11:44:16 Power up deskew - PASSED

│ 11:44:18 Power up deskew RF - PASSED

│ 11:44:18 Mss Trigger Cal

│ 11:44:18 Mss Trigger Cal - PASSED

│ 11:44:18 Last Cal'd facVersion: 5, current facVersion 5

│ 11:44:18 Last Cal'd SpcVersion: 5, current SpcVersion 5

│ 11:44:18 Last Cal'd facRfVersion: 4, current facRfVersion 4

│ 11:44:18 Last Cal'd SpcRfVersion: 1, current SpcRfVersion 1

│ 11:44:18 Last Cal'd facRfVectorVersion: 1

│ OK

└────

9 Interesting Commands

══════════════════════

My scope had only been powered up once before when I received it. At

that point, the "consoleLog1.txt" file probably contained whatever

happened at the factory to configure the scope. As part of my

exploration, I saved a few lines at the beginning of the file:

┌────

│ 00:40:30 errSetConsoleLogState() logging to /usr/local/nv/consoleLog1.txt

│ 00:40:54 cfgSetSerialNumber(): setting serial number to XXXXXXX

│ 00:40:54 cfgSetUBootEnvVariable - setting serial# = XXXXXXX

│ 23:57:14 cfgSetUBootEnvVariable - setting bboboard# = XXXXXX

│ 23:57:15 cfgSetUBootEnvVariable - setting hostname =

└────

The "cfgSetUBootEnvVariable" command is not listed in the command

help. Perhaps it's actually there (I haven't tried to run it) and

simply hidden, or maybe it's enabled by some other mode. I wish I'd

saved the entire file, but I didn't realize at that point that it

would be overwritten.

Table of Contents

─────────────────

1 Credits

2 Boot the scope with network access

3 Manufacturing Mode

4 Hardware Configuration

5 Debug Console

6 Linux shell access

7 Console logs

8 Example Console Log File

9 Interesting Commands

1 Credits

═════════

I learned much of this information from private conversations with

EEVBlog members. I'm just repeating the information.

For more details and to make contact with other curious people, visit

these threads: [MDO3000 Hacking] and [Someone has hacked MDO4000C?]

[MDO3000 Hacking]

https://www.eevblog.com/forum/testgear/mdo3000-hacking/

[Someone has hacked MDO4000C?]

https://www.eevblog.com/forum/testgear/someone-has-hacked-mdo4000c/

2 Boot the scope with network access

════════════════════════════════════

My scope comes up with a network address of 10.0.0.81

3 Manufacturing Mode

════════════════════

Before you start, use the scope's front panel to bring up the

Calibration page under the utility menu. You should see two other

menus along the bottom: "Signal Path" and "Factory".

To put the scope in manufacturing mode, connect to the SCPI shell at

port 4000. I'm on a recent mac (and they took telnet away), so I use

netcat in these examples.

┌────

│ $ nc -c 10.0.0.81 4000

│ Tektronix MDO4034B-3 Instrument Control Terminal Session

│ Control commands:

│ !t <timeout> : set the response timeout in milliseconds.

│ !d : send device clear to the instrument.

│ !r : read response from instrument.

│ !h : print this usage info.

│ Commands containing a ? are treated as queries and responses are read automatically.

│

│ Timeout is 10000 milliseconds

│ > :PASSW INTEKRITY

│ > :MFG:MOD 1

└────

As soon as you issue the :MFG:MOD command, you should see three more

menu items appear at the bottom: "Status Details", "RF IQ Vector" and

"Serial Console".

The easiest way to leave Manufacturing Mode is via SCPI:

┌────

│ > :MFG:MOD 0

└────

You can also turn it off via a new menu item in the GUI under Utility

> Self Test > Error Log.

4 Hardware Configuration

════════════════════════

While in Manufacturing Mode, goto Utility > Config > About > Hardware

Configuration

My MDO4034B-3 shows the following (Xs replace actual serial numbers):

┌────

│ Hardware Configuration

│

│ InstrumentSerial: XXXXXXX

│ Main Board Serial: XXXXXX

│ IO Board Serial: XXXXXX

│ Front Panel Serial: XXXXXX

│ Power Supply Serial: XXXXXXXX

│ Display Serial: XXXXXXXXXXXXX

│

│ Demux Info: 4 present, Version: 387D

│ Trigger Version: 2

│ Model (from resistors): 1 (4-channel, 350 MHz)

│ Sample Rate, Sample/sec: 5.000000e+09

│

│ Main Board Version: 11 (Rev: K)

│ Front Panel: Board: 16, FW: 53, CC: 6

│ Analog Front End: Board: RF\_3GHz\_BW\_1GHz, Rev: B, FW: 1032

│

│ CPU Version: B

│ U-Boot 2009.08 v1.07 Apr 19 2013

│ Linux 3.19.3 version 1.28

│

│ SDIA image: Version 141

│ View: bk\_cayman\_er\_rc1

│ Target: route66b\_scopeApp\_bk\_cayman\_er\_rc1.ppcex

└────

It's interesting to note that resistors are apparently used to

distinguish between different models and that my 350Mhz scope

(according to the sticker) has a 1GHz front end that's capable of

sampling at 5GS/s. Based on the hw info, it looks like a different

analog board is required to get a 6GHz spectrum analyzer.

5 Debug Console

═══════════════

These scopes have a "debug console" runs on port 1072. There's a

little bit of online help:

┌────

│ $ nc -c 10.0.0.81 1072

│ > ?

│ Commands available from the console are:

│ ? <command name>

│ DemoMgrStatus

│ acqBusTest [prt]:0

│ acqHsaUnitTest

│ acqInterpTest [prt]:0

│ acqMemLayout [bufType]:0 [recLen]:0

│ .

│ .

│ .

└────

The "utilConsoleRedirect" commands make sure you see the output of a

command. Without these, output probably goes to a UART somewhere.

┌────

│ > utilConsoleRedirect 1 1

│ StdOut is connected to debug

│ StdErr is connected to debug

│

│ > utilConsoleRedirect 2 1

│ StdOut is connected to debug

│ StdErr is connected to debug

│

│ >

└────

It's easy to leave the debug shell in a controlled way:

┌────

│ > quit

│ $

└────

6 Linux shell access

════════════════════

Use the "utilShell" console command to execute arbitrary shell

commands. UtilShell accepts a single argument and converts any

backslashes to spaces. I.e., to pass multiple words to a shell

command, use backslashes instead of spaces:

┌────

│ > utilShell uname\-a

│ string to shell: uname -a

│ Linux MDO4034B-3-XXXXXX.global.tektronix.net 3.19.3 V 1.28 Tektronix BK/CK Fri Sep 4 14:14:12 PDT 2015 ppc GNU/Linux

│ OK

│ >

└────

7 Console logs

══════════════

When the scope boots, it creates a log file. These are accessible in

the filesystem under /mnt/flash2/usr/local/nv:

┌────

│ > utilShell ls\-l\/mnt/flash2/usr/local/nv

│ string to shell: ls -l /mnt/flash2/usr/local/nv

│ -rw-r--r-- 1 root root 4187 Mar 31 11:44 consoleLog1.txt

│ -rw-r--r-- 1 root root 4188 Nov 29 10:53 consoleLog10.txt

│ -rw-r--r-- 1 root root 4188 Dec 3 13:22 consoleLog11.txt

│ -rw-r--r-- 1 root root 4188 Dec 18 21:14 consoleLog12.txt

│ -rw-r--r-- 1 root root 4188 Dec 25 11:57 consoleLog13.txt

│ -rw-r--r-- 1 root root 4188 Dec 26 11:21 consoleLog14.txt

│ .

│ .

│ .

│ -rwxr-xr-x 1 root root 60 Aug 28 2017 setup1.nfo

│ -rw-r--r-- 1 root root 97291 Aug 28 2017 setup1.set

│ -rw-r--r-- 1 root root 1646503 Aug 1 2017 userSaves.set

│ OK

│ >

└────

There are only 50 consoleLog\*.txt files at any particular time, so it

appears that the names wrap and files are re-used with each boot.

8 Example Console Log File

══════════════════════════

┌────

│ > utilShell cat\/usr/local/nv/consoleLog1.txt

│ string to shell: cat /usr/local/nv/consoleLog1.txt

│ errSetConsoleLogState() logging to /usr/local/nv/consoleLog1.txt

│ cfgInit

│ versionBuildFWVersionString(): TimestampString: 24-Oct-16 13:29

│ VersionFIRMWAREVERSIONversion: v3.22

│ Major ver num: 3 Minor ver num: 22

│ sysInit

│ execInit

│ hwInit

│

│ Main Board HW Rev: 0x0b

│

│ vertReprogramFeProc(): Platform Route66b fw 1032 filefw 1032

│ Front Panel Firmware update not needed

│ Current firmware 1032 >= 1032

│

│ Main Board HW ID: 0x07

│

│ AFE Board SW ID: 0x12

│ cfgGetRfHwInfo(): Contents of CfgRfHwInfo:

│ rfHwPresent = 1; rfFrontEndType = 2; rfAfeRev = 2

│ rfBw = 3e+09; rfLowBandStartFreq = 9000; rfAttenResolution = 1.000000

│ rfAcqMemSize = 2e+09

│

│ Main Board SW ID: 0x01

│

│ HFD144[0] ID\_REG = 0x00001440

│ HFD144[1] ID\_REG = 0x00001440

│ HFD144[2] ID\_REG = 0x00001440

│ HFD144[3] ID\_REG = 0x00001440

│ fanControlInit

│ Init ADT7476.

│ mitlInit

│ diagInit

│ diagRunEarlyPostDiags

│ ialInit

│ ialInit(): AFE id 0x12, rev 0x2, bI 8

│ calInit

│ Factory Checksum: Stored: 35540, Calculated: 35540 Spc CheckSum: stored: 27074 calculated: 27074 Factory RF Checksum: Stored: 30720, Calculated: 30720 Spc RF CheckSum: stored: 28115 calculated: 28115 Factory RF Vector Checksum: Stored: 33923, Calculated: 33923 dmxInit

│ Demux initialization

│

│ Dram Calibration results:

│ -------------------------

│ DramCal has PASSED on all Demuxs

│

│ Demux initialization complete

│ debugConsoleInit

│ diagRunPostDiags

│ Starting POST diags

│ Finished POST diags

│ dmxInitAfterDiagnostics

│ acqInit

│ autosetInit

│ clkInit

│ demoMgrInit

│ fpInit

│ extPwrInit

│ cmdInit

│ dsyInit

│ dsyMgrInit

│ dvmInit

│ hcInit

│ limInit

│ markInit

│ searchInit

│ cmdPeriodicSenderInit

│ maskInit

│ dspLibInit

│ mathInit

│ measInit

│ histogramInit

│ usbInit

│ fileUtilInit

│ measPackageInit

│ piInit

│ probesInit

│ refInit

│ rfInit

│ rfMeasInit

│ start\_rtlInit

│ trigInit

│ txtDsyInit

│ uiInit

│ utilInit

│ vertInit

│ vidPicInit

│ vxi11Init

│ waveAlertInit

│ wfmDsyInit

│ wfmMgrInit

│ busInit

│ calPowerupRun

│ mitlInit

│ acqInit

│ trigInit

│ afgInit

│ ---------------------- startScopeApp() running Start code ------------

│ execStart

│ mitlStart

│ acqStart

│ autosetStart

│ calStart

│ cfgStart

│ clkStart

│ cmdStart

│ diagStart

│ dsyStart

│ dvmStart

│ errStart

│ fpStart

│ extPwrStart

│ hcStart

│ hwStart

│ limStart

│ searchStart

│ cmdPeriodicSenderStart

│ maskStart

│ mathStart

│ measStart

│ measPackageStart

│ histogramStart

│ nvStart

│ piStart

│ probesStart

│ refStart

│ rfStart

│ rfMeasStart

│ start\_rtlStart

│ trigStart

│ start\_postTrigRtlStart

│ ialStart

│ txtDsyStart

│ uiStart

│ usbStart

│ fileUtilStart

│ utilStart

│ vertStart

│ vidPicStart

│ vxi11Start

│ waveAlertStart

│ wfmDsyStart

│ wfmMgrStart

│ busStart

│ afgStart

│ ---------------------- startScopeApp() running Run code --------------

│ execRun

│ autosetRun

│ calRun

│ cfgRun

│ clkRun

│ cmdRun

│ diagRun

│ dsyRunInitWfmPalette

│ dvmRun

│ errRun

│ fpRun

│ extPwrRun

│ hcRun

│ hwRun

│ limRun

│ wfmMgrRun

│ searchRun

│ cmdPeriodicSenderRun

│ maskRun

│ mathRun

│ measRun

│ histogramRun

│ measPackageRun

│ nvRun

│ piRun

│ probesRun

│ refRun

│ rfRun

│ rfMeasRun

│ start\_rtlRun

│ trigRun

│ ialRun

│ txtDsyRun

│ uiRun

│ usbRun

│ utilRun

│ enetRun

│ fileUtilRun

│ vertRun

│ vidPicRun

│ vxi11Run

│ waveAlertRun

│ wfmDsyRun

│ busRun

│ debugConsoleRun

│ mitlRun

│ afgRun

│ OK to connect by: telnet MDO4034B-3-XXXXXX.global.tektronix.net 1072

│ --------------------------------------------------------

│ startScopeApp() complete; duration = 15.348500 seconds

│ =================================================================

│ PID to Task info written to /tmp/threads.txt

│ Power Up Completed at 11:44:06

│ 11:44:06 Enter 'ctrl-\' to quit scopeApp

│

│ 11:44:06 SocketServerService: Socket server daemon started on port 4000.

│ Protocol: Raw

│ 11:44:08 SDIA version 0x141, Trinity, running calSdiaRunClkSync()

│ 11:44:08 calSdiaRunClkSync PASSED

│ 11:44:16 Power up deskew - PASSED

│ 11:44:18 Power up deskew RF - PASSED

│ 11:44:18 Mss Trigger Cal

│ 11:44:18 Mss Trigger Cal - PASSED

│ 11:44:18 Last Cal'd facVersion: 5, current facVersion 5

│ 11:44:18 Last Cal'd SpcVersion: 5, current SpcVersion 5

│ 11:44:18 Last Cal'd facRfVersion: 4, current facRfVersion 4

│ 11:44:18 Last Cal'd SpcRfVersion: 1, current SpcRfVersion 1

│ 11:44:18 Last Cal'd facRfVectorVersion: 1

│ OK

└────

9 Interesting Commands

══════════════════════

My scope had only been powered up once before when I received it. At

that point, the "consoleLog1.txt" file probably contained whatever

happened at the factory to configure the scope. As part of my

exploration, I saved a few lines at the beginning of the file:

┌────

│ 00:40:30 errSetConsoleLogState() logging to /usr/local/nv/consoleLog1.txt

│ 00:40:54 cfgSetSerialNumber(): setting serial number to XXXXXXX

│ 00:40:54 cfgSetUBootEnvVariable - setting serial# = XXXXXXX

│ 23:57:14 cfgSetUBootEnvVariable - setting bboboard# = XXXXXX

│ 23:57:15 cfgSetUBootEnvVariable - setting hostname =

└────

The "cfgSetUBootEnvVariable" command is not listed in the command

help. Perhaps it's actually there (I haven't tried to run it) and

simply hidden, or maybe it's enabled by some other mode. I wish I'd

saved the entire file, but I didn't realize at that point that it

would be overwritten.