

Control of the LBNL Digital Optical Module Test Boards using a PC-104 Single-Board Computer

John Jacobsen
Lawrence Berkeley National Laboratory

`jacobsen@rust.lbl.gov`

Last updated January 8, 2001

Table of Contents

OVERVIEW	1
SYNCSERVER	2
THE PERL INTERFACE TO SYNCSERVER	3
THE NETWORK INTERFACE TO SYNCSERVER	4
HOW TO COMPILE, INSTALL AND RUN THE SOFTWARE	7
NOTES ON CONFIGURATION OF THE PC-104/TEST BOARD SYSTEMS	8

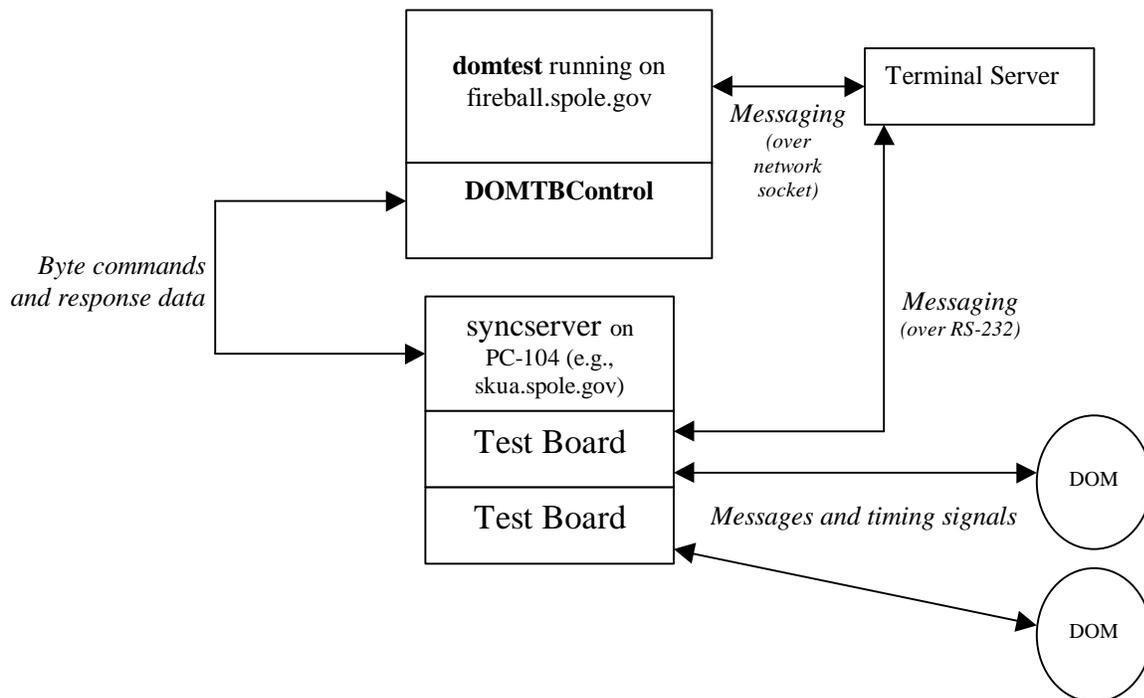
Overview

This document describes the software operation of the Digital Optical Modules (DOMs) via "Test Boards." The Test Boards will be introduced in the '00/'01 season at South Pole to supplement the existing communications functionality provided by the DESY data acquisition system (DAQ) and terminal servers. The most important new features provided by the Test Boards will be the demonstration of time calibration and of fast digital communications (up to about 60 kb/sec).

The Test Boards are controlled by a Linux PC-104¹ system. Up to four Test Boards sharing a PC-104 bus and a partitioned address space can be controlled by one PC-104 unit via that bus. A program called "syncserver" running on the PC-104 accepts commands over a network connection and translates these commands into memory-mapped I/O on one or more Test Boards. The Test Boards, in turn, control communications to the DOMs over the long cables. The Test Board API is described in some detail by Jerry Przybylski at http://rust.lbl.gov/~gtp/local/testboard_API.htm.

Most of the interaction between an application on the surface and the DOMs currently occurs via an RS-232 serial connection which the Test Board translates into tri-state differential encoding for transmission down the long cable. This process is largely transparent to syncserver - for normal communication, a process opens a socket to a terminal server which in turn connects via the RS-232 interface to the Test Board. However, for the transmission of time calibration pulses, normal RS-232 communications is suspended, and the Test Board must be told to generate a calibration pulse, and also can provide a digitization of the return pulse. This is the key functionality which syncserver must provide.

The following diagram shows some of the relationships at play:



¹ PC-104 is an embedded computing standard based on IBM-PC compatible, single board computers with a compact form factor and a shared, compact ISA bus (also known as the PC-104 bus).

The software itself can be divided into the following components:

1. The **syncserver** PC-104 control software (written in C), which talks to the FPGA in the test board using memory-mapped registers in port I/O space.
2. A low-level C-language interface (**portio.c / portio.h**) which syncserver uses to talk to the test boards. It also includes functionality to talk to a PC-104 relay switch system, used to turn on or off the power to the test boards and other hardware.
3. A **Jamplayer** package, from Altera, Inc., ported to the PC-104 system. This package allows syncserver to program and deprogram the FPGA.
4. A Perl package, called **DOMTBControl.pm**, which provides the interface whereby a Perl application on the surface can issue commands to syncserver.
5. The Perl application **domtest** or the skeleton test program **client_test.pl** which uses DOMTBControl.pm.

All these software components are in the CVS² tree on rust.lbl.gov in the directories domsoft/src/domio and domsoft/src/portio. The former directory is the location of the Perl test and data logging programs (domtest, domtalk, DOMLogger). The latter is the location of the PC-104-specific software (syncserver, portio, Jamplayer) and the DOMTBControl.pm Perl interface to syncserver.

The following example will highlight the most important features of this design. Assume a Perl application (e.g. domtest) on the surface wants to send a timing pulse down the cable. The application must first create a DOMTBControl object, which opens a socket to syncserver running on the PC-104. When a timing pulse is desired, the application calls a method of the DOMTBControl object which sends a single byte command on the socket. Syncserver reads this byte, interprets it as an instruction to send a timing signal, and uses functions implemented in portio.c to write the appropriate registers in I/O space. The FPGA in the Test Board generates the timing pulse, and provides a 16-byte integer timestamp of the timing pulse, which is then read out by syncserver. If all goes well, syncserver then sends a confirmation response byte on the socket, followed by the time stamp data, and the DOMTBControl method will return a "status ok" value so that the application knows that everything's fine; the method also returns the time stamp data.

syncserver

As explained above, syncserver runs on the PC-104 and listens for network commands from a client application. When a command is given, syncserver reads and writes the appropriate FPGA registers on the test board, and sends response data back on the socket to the client.

The Test Board FPGA registers are accessed in memory-mapped I/O space using the inb() and outb() functions. A layer of routines, defined in portio.c, encapsulate this functionality. The details of which bits in which registers have which functions are specified in Jerry's Test Board API document.

² CVS - Concurrent Versions System. This software allows for version control for code development by multiple developers running on separate development machines. All the DOM software except for the DOM Application and FPGA designs are in the domsoft CVS archive on rust.lbl.gov. For more information, see <http://rust.lbl.gov/~jacobsen/cvs>.

Syncserver runs setuid root (its file permissions should be 04755). The source code is located in the portio directory. For compilation and installation instructions, see **How to Compile, Install and Run the software**, below. Currently, installation and execution of syncserver has to be done by hand, although this could probably be automated and improved.

The Perl interface to syncserver

The Perl package DOMTBControl.pm provides all the functionality one needs to talk to syncserver, and therefore to the Test Board.

When you create a new DOMTBControl object, the "new" method opens a socket to the PC-104 system (syncserver listens on port 3666). Calls to that object's methods cause data to be written to syncserver over the socket. This data consists of a control byte along with zero or more data bytes, with the number of data bytes fixed by the value of the control byte. Response data consists of a status byte followed by zero or more message data bytes. The details of the interface are described below, in **The Network Interface to syncserver**.

If a DOMTBControl method returns the string "STATUS_OK" in the first argument, that indicates that the command completed successfully. Otherwise, the error string will contain information about what went wrong.

As an example, the following code sends a time tick down the cable to the DOM:

```
use DOMTBControl;

my $tb = DOMTBControl::new("skua.spole.gov"); # Open connection
die "Can't connect!\n" unless defined $tb;

# Send the time pulse:
my ($err, $timestamp0, $timestamp1) = $tb->sendTimePulse(0);

if($err eq "STATUS_OK") {
    print "Time pulse was acknowledged by syncserver.\n";
    print "Timestamp: $timestamp0 $timestamp1.\n";
} else {
    print "Time pulse request generated an error : $err.\n";
}

$tb = undef;
# When the test board object goes out of scope
# or is no longer referred to by any
# scalar, the socket connection to the PC-104 is closed.
```

For a real time calibration, the example would have to include DOMSet³ messages to the DOM which tell it to "go quiet" (stop communications using the RS-232 interface), digitize the time tick, send a time tick in response, and DOMTBControl functions to read out the received time tick in the test board.

³ DOMSet is the Perl interface to the DOMs via the terminal server. See "Perl Classes for Interacting with the DOM," by J. Jacobsen, http://rust.lbl.gov/amanda/dom/domsoft/docs/serial_communications.html

Current methods supported by DOMTBControl.pm:

- new() -- Creates a DOMTBControl object and opens a connection to syncserver running on the PC-104.
- echoTest() -- just a test to make sure things are working, by sending a byte to syncserver and making sure syncserver sends the same byte back.
- FPGARegisterTest() -- syncserver writes byte patterns to FPGA Control Register 0 & makes sure they can be read back ok
- sendTimePulse() -- causes syncserver to tell the FPGA on the Test Board to send a timing pulse down the cable.
- readTBTimeTickData() -- get timestamp and digitized waveform of DOM timetick received by test board. (Returns error string, time stamp, and reference to an array of waveform values).
- FPGAUnload() -- unload the current program in the FPGA on the PC/104 system
- FPGAListFiles() -- list currently available FPGA files on the PC/104 system
- ForceDOMCommunicationsEnable() -- force the test board FPGA to reenable communications with the DOM over the RS-232/terminal server connection.
- ReadFPGARegister() -- read a test board FPGA register directly
- WriteFPGARegister() -- write a test board FPGA register directly

The Network Interface to syncserver

This section consists of messages to syncserver and their response values. In other words, this is the API which DOMTBControl.pm uses to communicate with syncserver. Messages are all call-and-response, with the call consisting of a command byte with zero or more data bytes, and the response consisting of a status byte with zero or more data bytes. Please note - all multi-byte integer values are sent big-endian (network byte order).

Messages to syncserver

Message: **Echoback**

Byte value: 0x02

Associated DOMTBControl method: echoTest

Number of arguments: 1

Return value: the argument

Code status: implemented

Message: **FPGA Test**

Byte value: 0x09

Associated DOMTBControl method: FPGARegisterTest

Number of arguments: 1 byte (test board ID)

Return value: status byte

Code status: implemented

Message: **Issue Timing Pulse**

Byte value: 0x03

Associated DOMTBControl method: sendTimePulse

Number of arguments: 1 byte (test board ID)

Return values:

Status byte

If status is STATUS_OK, two four-byte integer timestamps of timing pulse.

Code status: implemented

Message: **Read Received Timetick**

Byte value: 0x08

Associated DOMTBControl method: readTBTimetickData

Number of arguments: 1 byte (test board ID)

Return values:

Status byte

If status is STATUS_OK, two four-byte integer timestamps of the received pulse, as well as 256 bytes containing the digitized waveform.

Code status: implemented

Message: **FPGA load command**

Byte value: 0x04

Associated DOMTBControl method: loadFPGAFile

Number of arguments: 1 byte (test board ID) followed by variable number of bytes (name of jam file, terminated with \n)

Return value: status byte (see syncserver.h).

Code status: implemented

Message: **FPGA list available designs**

Byte value: 0x05

Associated DOMTBControl method: listFPGAFiles

Return message:

Status byte

If status byte is STATUS_OK, rest of the bytes are a string containing file names, separated by commas, terminated by a newline.

Code status: implemented

Message: **FPGA unload command**

Byte value: 0x06

Associated DOMTBControl method: unloadFPGA

Number of arguments: 1 byte (test board ID)

Return message: Command status byte

Code status: not implemented at lowest level (needs API info from Jerry); otherwise implemented

Message: **FPGA status command**

Byte value: 0x07

Associated DOMTBControl method: getFPGAStatus

Number of arguments: 1 byte (test board ID)

Return message:

Command status byte

FPGA status : 1 (loaded) or 0 (unloaded)

Code status: not implemented (needs API info)

Message: **Force DOM Communications Enable**

Byte value: 0x0A

Associated DOMTBControl method: ForceDOMCommunicationsEnable

Number of arguments: 1 byte (test board ID)

Return message: command status byte

Code status: implemented.

Message: **Read FPGA Register**

Byte value: 0x0B

Associated DOMTBControl method: ReadFPGARegister

Arguments: 1 byte test board ID, 1 byte register number

Return message: status byte, register contents byte

Code status: implemented

Message: **Write FPGA Register**

Byte value: 0x0C

Associated DOMTBControl method: WriteFPGARegister

Arguments: 1 byte test board ID, 1 byte register number, 1 byte value to write

Return message: status byte

Code status: implemented

Message: **Turn Power Relay On**

Byte value: 0x0D

Associated DOMTBControl method: PowerRelayOn

Arguments: 1 byte relay number

Return message: status byte

Code status: implemented

Message: **Turn Power Relay Off**

Byte value: 0x0E

Associated DOMTBControl method: PowerRelayOff

Arguments: 1 byte relay number

Return message: status byte

Code status: implemented

A response of UNKNOWN_COMMAND (byte value 2) is given if the command byte is not one of the above commands.

Message Status Response Byte Definitions

Status: **STATUS_OK**

Byte value: 0x01

Meaning: It worked.

Status: **UNKOWN_COMMAND**

Byte value: 0x02

Meaning: I don't know what you're talking about

Status: **FUNC_NOT_IMPL**

Byte value: 0x04

Meaning: I know what you're talking about, but I can't do it yet

Status: **FPGA_FILE_NOT_FOUND**

Byte value: 0x03

Meaning: You wanted me to load an FPGA file that I can't find in the usual place on the PC-104.

Status: **FILENAME_TOO_LONG**

Byte value: 0x05

Meaning: The name of the FPGA file you asked me to load is too long.

Status: **JAM_LOAD_FAILED**

Byte value: 0x06

Meaning: The Jamplayer software couldn't load the FPGA file for some reason.

Status: **SHORT_WF_READ**

Byte value: 0x07

Meaning: I didn't get the expected number of bytes from the digitized waveform.

Status: **CANT_READ_WF**

Byte value: 0x08

Meaning: I can't read the waveform data from the COM ADC.

Status: **FPGA_TEST_FAILED**

Byte value: 0x09

Meaning: Read/write test failed

Status: **REGI_OUT_OF_RANGE**

Byte value: 0x0A

Meaning: The FPGA register number was too large

How to Compile, Install and Run the software

To build syncserver and install DOMTBControl.pm:

1. change to domsoft/src/portio
2. cvs update your code
3. make
4. make install (this copies DOMTBControl.pm to /usr/local/dom/lib)

To install syncserver on the PC-104 system:

1. in the portio directory, run update_syncserver. This copies syncserver to the PC-104 system in the directory /tmp and gives it the correct file permissions

To run syncserver:

1. Telnet to the PC-104 system
2. Verify, using "ps ax | grep syncserver" that syncserver isn't running
3. If it is, kill it with "kill -9 <pid>" where <pid> is the process ID of the currently running syncserver.
4. Start syncserver with "/tmp/syncserver".

5. On the client machine (e.g., fireball.spole.gov or rust.lbl.gov) run `client_test.pl` or `domtest`. If running `domtest`, choose "Test Board Functions" followed by "connect to test board/PC-104 system." If it connects successfully, `syncserver` is running and accepting connections.

To install an FPGA design on the PC-104 system:

1. FTP to the PC-104 system with username "dom"
2. `cd /tmp`
3. put the FPGA design file
4. telnet to the PC-104 system
5. `su to root`
6. make the root filesystem writable (see General Notes in **Notes on Configuration of the PC-104/Test Board Systems**)
7. move the FPGA design file from `/tmp` to `/home/dom/fpga`
8. make the root filesystem read-only

If `syncserver` and `domtest` are running, you can test that the FPGA design has made it by selecting "Load a new FPGA file on PC-104 system" from the Test Board Functions menu, and loading the new design.

Notes on Configuration of the PC-104/Test Board Systems

As of this writing, there is one working PC/104 system (`skua`), with two others which have been received and are being configured (`petrel` and `fulmar`)⁴. The systems are made by Emac Inc. (www.emacinc.com) and are configured as follows:

- PCM-3346 Single board PC-104 computer
- 64MB 144p EDO SODIMM (RAM)
- 3.5" HD Floppy Drive (currently on back-order)
- 3.5" IDE Hard Disk
- 48MB flash disk
- Linux installation on IDE drive and Flash disk
- Perl installation on IDE disk
- Small SMTP client for sending mail installed on both disks
- Realtime extensions installed on both disks
- Apache Web server installed on both disks

Emac has fairly extensive hardware and configuration documentation for this system:

The PCM-3364 User's Manual, http://rust.lbl.gov/~jacobsen/docs/dom/3346_fnt.pdf is a description of the hardware and low-level configuration and

The Servier in a Box Manual, http://rust.lbl.gov/~jacobsen/docs/dom/sib_20_manual.pdf is a description of the Linux system and software configuration.

The following notes should fill in some of the missing details on what extra steps had to be taken to get the PC-104 systems to run at LBNL and the Pole.

A few general notes.

⁴ The names refer to Antarctic birds

- Our systems are set up to run off the flash disks, with the hard disks as backups in case problems arise with the flash disks.
- The flash disks are configured read-only to help ensure longevity for the flash filesystems, with directories such as /tmp implemented by RAM disks. **To make the root filesystem writable**, issue the following as root:

```
mount -o remount,rw /dev/tffs1 /
```

 To make it read-only again, substitute "ro" for "rw".

The installation / configuration sequence for the PC-104 systems is as follows:

- 1) If not already done, cable the PC-104 unit. The cables on petrel and fulmar are labeled, but you may need to refer to the connector diagram on the PCM-3364 User's Manual, pp. 9-10. If connector orientation is ambiguous, match the triangle on the connector with the arrow on the PCB.
 The crucial things to cable here are the power supply, network cable, and, if interactivity is desired, keyboard and VGA monitor.
- 2) See if the unit powers on ok and passes the self-tests. Examine the BIOS settings before it boots the operating system. Unit is configured to boot from the flash disk (Disk on a Chip or DOC) by default.
- 3) Make sure Linux boots.
- 4) Log in as root, with emac_inc as the password.
- 5) Quit the configuration menu. Mount / with read/write access: "mount -o remount,rw /dev/tffs1 /". Use the "ae" editor to comment out the line in /root/.profile invoking the SIBconfig.sh script. The script can be invoked by hand later with "/root/SIBconfig.sh". Getting rid of this line avoids the automatic running of the script when root logs in.
- 6) Change the password with the "passwd" command. Create a "dom" account by editing /etc/passwd (again, use the "ae" editor, and set the UID to 400) and create a home directory /home/dom.
- 7) Run the configuration script /root/SIBconfig.sh
- 8) Change the host name (e.g., skua.spole.gov)
- 9) Change the ethernet settings. Use ethernet device zero. Enter IP address, gateway, netmask, network address, broadcast address.
- 10) Change the default gateway by selecting "3) Routing tables" followed by "3) Set default gateway". Use the same number here as the network address in the previous step.
- 11) Change the DNS nameserver addresses (menu item 5) - remove old nameservers and static host address, add new nameservers and static host address (use the same name as the hostname you entered in item 8). Also change the domain and search names by editing /etc/resolv.conf with the "ae" editor. You may have to add the nameserver entries by hand when editing this file.
- 12) Create FPGA JAM files directory /home/dom/fpga (see instructions, above)
- 13) Reboot the machine using "shutdown -r now", make sure it boots and can be talked to on the network. Install and run syncserver (see instructions, above).

Further instructions for configuring the PC-104 system to boot from the hard disk:

- 14) Cable the hard disk. The IDE cable goes from the PC-104 system to the disk. There should also be a red, yellow and black four-wire power cable from the power supply to the disk.
- 15) Power up and hit the "DEL" key as soon as you see the first BIOS screen. Select "Advanced BIOS Features." Set the primary boot disk to HDD0 (hard disk), secondary boot disk to

HDD1 (flash disk), third boot disk to floppy. Escape to main menu and then select "Save and Exit Setup."

- 16) At the LILO prompt type Linux, or just hit return, or just wait.
- 17) Wait for the system to boot.
- 18) Log in as root, typing default password.
- 19) Change root password w/ passwd command.
- 20) If not already present, add a user "dom" with the following command: "useradd -u 400 -s /bin/tcsh dom"
- 21) Change the password on the dom account with the passwd command.
- 22) Log in as dom using "su - dom". Create the "fpga" directory with "mkdir fpga". Log out of the dom account with "logout"
- 23) Edit the following files:
 - /etc/HOSTNAME -- change host name
 - /etc/hosts -- change host name and IP address. Leave localhost line alone.
 - /etc/resolv.conf -- change domain name and add nameserver info.
 - /etc/sysconfig/network -- change info for HOSTNAME, DOMAINNAME, GATEWAY.
 - /etc/sysconfig/network-scripts/ifcfg-eth0 -- change info for IPADDR, NETMASK, NETWORK (same as GATEWAY, above), BROADCAST.
- 24) Finally, reboot the machine ("shutdown -r now"), verify that you can log in and use the ping command locally and remotely to test the new network settings.

In this configuration, with both hard disks and flash disks ready to boot, unplugging the power to the hard disk will result in automatically booting from the flash disk. As of this writing, this is the default behaviour.