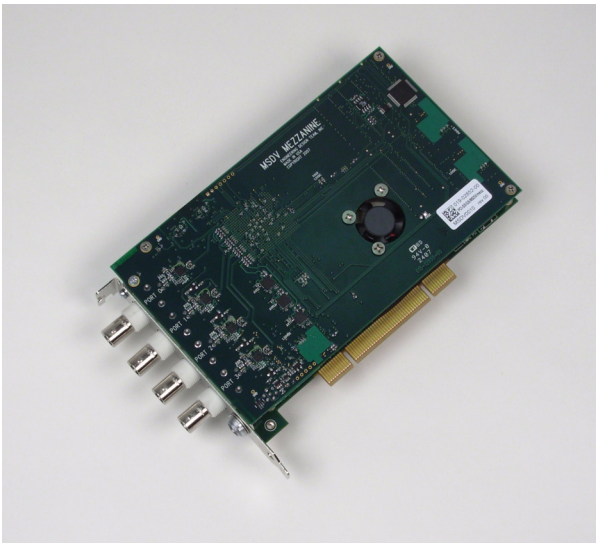
MSDV

Multirate serial digital video



A mezzanine board for use with a main board

(PCI SS, PCI GS, or PCIe8 LX)

September 17, 2008
008-02784-00

edt
a HEICO company

Contact

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich
Germany
www.skyblue.de
info@skyblue.de
Tel. +49 (0) 89 - 780297 0

Terms of Use Agreement

Definitions. This agreement, between Engineering Design Team, Inc. (“Seller”) and the user or distributor (“Buyer”), covers the use and distribution of the following items provided by Seller: a) the device driver binary software, as well as all source code for software libraries, utilities, and example applications (collectively, “Software”); b) the binary and all provided source code for any and all configurable or programmable devices (collectively, “Firmware”); and c) the computer boards and all other physical components (collectively, “Hardware”). Software, Firmware, and Hardware are collectively referred to as “Products.” This agreement also covers Seller’s published Limited Warranty (“Warranty”) and all other published manuals and product information in all forms, both physical and electronic (“Documentation”).

License. Seller grants Buyer the right to use or distribute Seller’s Software and Firmware Products solely to enable Seller’s Hardware Products. Seller’s Software and Firmware must be used on the same computer as Seller’s Hardware. Seller’s Products and Documentation are furnished under, and may be used only in accordance with, the terms of this agreement. By using or distributing Seller’s Products and Documentation, Buyer agrees to the terms of this agreement, as well as any additional agreements (such as a nondisclosure agreement) between Buyer and Seller.

Export Restrictions. Buyer will not permit Seller’s Software, Firmware, or Hardware to be sent to, or used in, any other country except in compliance with applicable U.S. laws and regulations. For clarification or advice on such laws and regulations, Buyer should contact: U.S. Department of Commerce, Export Division, Washington, D.C., 20230, U.S.A.

Limited Rights. Seller grants Buyer a royalty-free right to modify, reproduce, and distribute executable files using the Seller’s Software and Firmware, provided that: a) the source code and executable files will be used only with Seller’s Hardware; b) Buyer agrees to indemnify, hold harmless, and defend Seller from and against any claims or lawsuits, including attorneys’ fees, that arise or result from the use or distribution of Buyer’s products containing Seller’s Products. Seller’s Hardware may not be copied or recreated in any form or by any means without Seller’s express written consent.

No Liability for Consequential Damages. In no event will Seller, its directors, officers, employees, or agents be liable to Buyer for any consequential, incidental, or indirect damages (including damages for business interruptions, loss of business profits or information, and the like) arising out of the use or inability to use the Products, even if Seller has been advised of the possibility of such damages. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to Buyer. Seller’s liability to Buyer for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, product liability, tort including negligence, or otherwise) will be limited to fifty U.S. dollars ($50.00).

Limited Hardware Warranty. Seller warrants that the Hardware it manufactures and sells shall be free from defects in material and workmanship for a period of 12 months from date of shipment to the initial Buyer. This warranty does not apply to any products which are misused, abused, repaired, or otherwise modified by Buyer or others. Seller's sole obligation for breach of this warranty shall be to repair or replace (F.O.B. Sellers Plant, Beaverton, Oregon) any goods which are found to be non-conforming or defective as specified by Buyer within 30 days of discovery of any defect. All installation and transportation expenses, and all other incidental expenses and damages, shall be borne by the Buyer. *This warranty is expressly in lieu of all other warranties, express or implied, including but not limited to any warranties of merchantability or fitness for any particular purpose.*

Limitation of Liability. *In no event shall seller be liable for any type of special consequential, incidental, or penal damages, whether such damages arise out of or are a result of breach of contract, warranty, tort (including negligence), strict liability, or otherwise.* All references to damages herein shall include, but not be limited to, loss of profit or revenues, loss of use of the goods or associated equipment, costs of substitute goods, equipment, facilities, downtime costs, or claims for damages. Seller shall not be liable for any loss, claim, expense, or damage caused by, contributed to, or arising out of the acts or omissions of Buyer, whether negligent or otherwise.

No Other Warranties. Seller makes no other warranties, express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose, regarding Seller’s Products or Documentation. Seller does not warrant, guarantee, or make any representations regarding the use or the results of the use of the Products or Documentation or their correctness, accuracy, reliability, currentness, or otherwise. All risk related to the results and performance of the Products and Documentation is assumed by Buyer. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to Buyer.

Disclaimer. Seller’s Products and Documentation, including this document, are subject to change without notice. Documentation does not represent a commitment from Seller.

Contents

MSDV Mezzanine Board

The MSDV is a mezzanine board that connects to a main board (PCI SS, PCI GS, or PCIe8 LX). It supports the DVB ASI (digital video broadcast asynchronous serial interface) standard. It can receive and output DVB ASI streams on any of four channels. Each channel can be either input or output and maps to one of four DMA channels on the host.

The MSDV has four BNC connectors for each of the four channels. It also has a Lemo connector to input time code signals (IRIG-B or one-pulse-per-second) for timestamping data.

The MSDV can capture such data in any of six ways, discussed in MSDV Modes on page 7; two test modes also are provided. Data in raw mode — that is, with no manipulation — can be captured at approximately 27 MB per second. If the data needs to be manipulated in some way — for example, decoded — can be captured at approximately 4 MB per second or so, depending on the mode used.

The MSDV implements the DVB ASI interface using one field-programmable gate array (FPGA) on the mezzanine board, as well as the FPGA resources on the main board.

DVB ASI

DVB ASI data is a stream of packets representing broadcast video data, encoded with 8b/10b encoding. A complete discussion of DVB-ASI specifications is provided in EN 50083-9: Cabled Distribution Systems for Television, Sound and Interactive Multimedia Signals, available from the European Telecommunications Standards Institute website.

Related Manuals

Detailed documentation on EDT's C software library routines, helpful for writing your applications, is available on EDT's website in either HTML or PDF form. The *PCI SS/GS Main Board User's Guide* is available in PDF form.

Manual	URL
EDT DMA Software Library (HTML)	www.edt.com/api
EDT DMA Software Library (PDF)	www.edt.com/manuals/misc/api.pdf
PCI SS/GS Main Board User's Guide	www.edt.com/manuals/PCD/pciss_gs.pdf
PCI/GS or PCIe8 LX Time Distribution Board User's Guide	www.edt.com/manuals/PCD/time-dist.pdf

Installation

Install the PCI SS, PCI GS, or PCIe8 LX MSDV by fitting the connectors through the host back panel and then plugging into the PCI connector. The MSDV can be difficult to install in some host computers. The BNC connector is farther from the PCI Bus connector, so it can interfere with the host back panel and prevent the bottom of the MSDV back panel from being inserted smoothly. You can usually manipulate the board into position thus:

1. Place the edge connector between two PCI connectors on the motherboard.
2. Position the MSDV back panel.
3. Lift the board slightly to insert the edge connector into the host socket.

The BNC connector furthest from the PCI connector is channel 0, and the closest is channel 3; see Figure 2 for details.

You can load bitfiles and initialize the board in either of two ways:

- The example application `msdv_snap` loads the correct bitfiles and gets the board ready to acquire data, as explained in About msdv_snap on page 5.
- The EDT tool `initpcd` loads the bitfile specified in the command line, as explained in Configuring the MSDV on page 9.

Initializing the MSDV on page 11 explains in detail what happens at the hardware level during board initialization.

About the Software and Firmware

The distribution directory includes a subdirectory, `bitfiles/`, which in turn includes this subdirectory:

XC5VLX30T	Contains the bitfile for the Virtex 5 Platform Xilinx® FPGA, which is:
`dvb_asi.bit`	Configures the FPGA on the mezzanine board for DVB ASI operation.

NOTE The MSDV requires that the main board PCI FPGA be loaded with four-channel firmware, either `pciss4.bit` or `pcigs4.bit`.

The following MSDV-specific files are included in the distribution directory:

`msdv.bit`	Configures the user interface FPGA on the main board to communicate with the MSDV mezzanine board.
`mezzload`	An application that loads the bitfiles specified above into the MSDV FPGA and the user interface FPGA on the main board.
`mezzload.c`	The C source for the application above.
`msdv_snap`	Example application that captures data from the MSDV board and transfers it to disk for testing or verification.
`msdv_snap.c`	C source for `msdv_snap`.
`lib_msdv.c`	C library routines to use in your MSDV applications.
`edt_msdv.h`	Include file for the above C library routines.

Sample software initialization files for all board configurations are in the `pcd_config` subdirectory of the distribution directory, including:

`msdv.cfg`	Software initialization file for `initpcd` to use to configure the MSDV.

FPGA Configuration Files

The main board implements the DMA interface using two field-programmable gate arrays (FPGAs), referred to as the PCI FPGA and the UI (user interface) FPGA:

- The *PCI FPGA* communicates with the host computer over the PCI Bus. It implements the DMA engine, which transfers data between the board and the host computer, and loads its firmware on powerup from flash ROM located on the main board.
- The *UI FPGA* transfers data between the user device and the PCI FPGA; in some instances, it also sends the data to the mezzanine board. The UI FPGA or mezzanine board may also process the data in some manner, depending on the application.
- In addition, some mezzanine boards also have an FPGA which must be loaded with the correct firmware.

FPGA configuration files define the firmware required for the PCI FPGA and the UI FPGA. The PCI FPGA firmware files are in the `flash` subdirectory of the EDT top-level distribution directory. UI FPGA firmware files are in the `bitfiles` subdirectory of the EDT top-level distribution directory. mezzanine board configuration files, if any, are in the mezzanine board-specific subdirectory.

Each FPGA must be loaded with the firmware specific to the chosen interface, and the firmware in one FPGA must be compatible with the firmware in the other. By default, the correct FPGA configuration file for the PCI FPGA is loaded at the factory. However, you'll need to load the required FPGA configuration file for the UI FPGA yourself.

The firmware files specific to your MSDV are listed at the beginning of this section. Instructions for loading them are provided in Configuring the MSDV.

The PCD Device Driver

The PCD device driver is the software running on the host that allows the host operating system to communicate with the MSDV. The driver is loaded into the kernel upon installation, and thereafter runs as a kernel module. The driver name and subdirectory is specific to each supported operating system; the installation script handles those details for you, automatically installing the correct device driver in the correct operating system-specific manner.

Software Initialization Files

Software initialization files (having the extension `.cfg`) are editable text files that run like scripts to configure EDT boards so that they are ready to perform DMA. The commands in a software initialization file are defined in a C application named `initpcd`. When you invoke `initpcd`, you specify which software initialization file to use with the `-f` flag.

A typical software initialization file loads an FPGA configuration file into the UI FPGA and sets up various registers to prepare the board for DMA transfers. Some software initialization files may also load an FPGA configuration file into an FPGA residing on the mezzanine board.

A variety of software initialization files are included with the EDT software, at least one of which is customized for each main board or main board / mezzanine board combination — that is, each FPGA configuration file has a matching software initialization file. Software initialization files are located in the `pcd_config` subdirectory of the EDT top-level distribution directory. The software initialization files specific to your MSDV are listed at the beginning of this section. Instructions for their use are provided in Configuring the MSDV.

Commands defined in `initpcd` and typically found in software initialization files allow for specific FPGA configuration files to be loaded (for example, `bitfile:`), write specified hexadecimal values to

specified registers (for example, `command_reg:`), enable or disable byte-swapping or short-swapping to accommodate different operating systems' requirements for bit ordering (for example, `byteswap:`), or invoke arbitrary commands (for example, `run_command:`). If the mezzanine board you're using has an FPGA, you can load the appropriate firmware on it by running the command `mezzload`, which determines the appropriate firmware for the specific mezzanine board in your system.

For example:

```
bitfile: ssd16io.bit
command_reg: 0x08
byteswap: 1
run_command: set_ss_vco -F 1000000 2
run_command: mezzload
```

For complete usage details, enter `initpcd --help`.

C source for `initpcd` is included so that you can add your own commands, if you wish. You can then edit your own software initialization file to use your new commands and specify that `initpcd` use your new file when configuring your board. If you would like us to include your new software initialization commands in subsequent releases of `initpcd`, send mail to `tech@edt.com`.

Sample Applications and Utilities

Along with the driver, the FPGA configuration files, and the software initialization files, the software CD includes a number of applications and utilities that you can use to initialize and configure the board, access registers, or test the board. For many of these applications and utiilities, C source is also provided, so that you can use them as starting points to write your own applications. The most commonly useful are described below; see the README file for the complete list.

NOTE Software is updated regularly; the latest versions are available on our website at www.edt.com/software.html. We encourage you to use the latest versions for new installations. For existing applications, upgrade only if you have a specific reason to do so.

Sample Applications

`rd16`	Performs simple multichannel ring buffer input.
`wr16`	Performs simple multichannel ring buffer output.
`simple_read`	Performs DMA input without using ring buffers. Data is therefore subject to interruptions, depending on system performance.
`simple_write`	Performs DMA output without using ring buffers. Data is therefore subject to interruptions, depending on system performance.
`simple_getdata`	Serves as an example of a variety of DMA-related operations, including reading the data from the connector interface and writing it to a file, as well as measuring input rate.
`simple_putdata`	Serves as an example of a variety of DMA-related operations, including reading data from a file and writing it out to the connector interface.
`test_timeout`	Under normal operation, timeouts cancel DMA transfers. This application exemplifies giving notification when a timeout occurs, without canceling DMA
`set_ss_vco`	A utility for programming the output clock or clocks on the MSDV to specific frequencies used by the UI FPGA for input and output.

Utility Files

`initpcd`	A utility for initializing and configuring the MSDV.
`pdb`	Utility application that enables interactive reading and writing of the PCI SS/GS UI FPGA registers.

Testing Files

A variety of files — C source, executables, and FPGA configuration files — are available to test the boards. Their uses are described in the documents listed under the heading Testing Procedures. They include at least:

`sslooptest`	Tests most PCI SS- and PCI GS-based boards. Determines the board model and selects the loopback test to run, then runs it.
`xtest`	Tests the PCI CD and CDa boards, and the single-channel DMA interface for the PCI SS and PCI GS main boards.

Building Applications

Executable and PCD source files are at the top level of the EDT PCD driver distribution directory. If you need to rebuild an application, therefore, run `make` in this directory.

Windows and Solaris users must install a C compiler. For Windows, we recommend the Microsoft Visual C compiler; for Solaris, the Sun WorkShop C compiler. Linux users can use the `gcc` compiler typically included with your Linux installation. If Solaris or Windows users wish to use `gcc`, contact `tech@edt.com`.

After you've built an application, use the `--help` command line option for a list of usage options and descriptions.

About msdv_snap

The application `msdv_snap` and its accompanying C source code provides an example of capturing data. This command-line application can be invoked with a number of options to customize its behavior.

For a Help message listing all usage options, invoke these applications with the flag `-h`. For example:

```
msdv_snap -h
```

The following example captures 100 MB of a video signal (the `-s` flag) from from channel 1 (the `-c` flag) of unit 0 (the `-u` flag, the first board in the system), specifying the data be captured using mode 3 (the `-m` flag) and placed in the specified output file (the `-o` flag):

```
msdv_snap -u 0 -c 1 -m 3 -s 100 -o output_file
```

This takes approximately thirty seconds.

The following brief discussion points out a few considerations not given in the Help message.

- Specify the operating mode with the `-m` flag, which takes an integer 0–7 as an argument. Each is described in MSDV Modes on page 7.
- `msdv_snap` allows you to specify that the output be formatted in hexadecimal chunks of 32, 16, or 8 bits , using the flags `-H`, `-Hw`, or `-Hb`, respectively. In all cases, the most significant bit is the first bit output (in time) and the leftmost bit of the chunk (in memory).
- The flag `-s` to `msdv_snap` specifies the final file size in megabytes. The application terminates when the specified size has been reached.

- `msdv_snap` allows you to change the default number and size of the ring buffers using the flags `-n` and `-b`. For performance reasons, the ring buffer size is always rounded to the nearest multiple of 4096. The application then checks to determine whether the requested size and number of ring buffers is reasonable for the line rate. If it is not, the application configures the ring buffers as requested, but emits a warning message. The default parameters have been chosen to work with the DVB ASI line rate.
- By default, `msdv_snap` initializes the board, including initializing all the channels. However, you can suppress this initialization by invoking the second process with the flag `-I` (to skip initialization). This can be useful if you wish to run more than one instance of the application at a time, with the different processes accessing data from different channels. In that case, if a subsequent process initialized all the channels, it would interfere with the first process.
- To determine the status of a specified channel without running the application, you can invoke `msdv_snap` with the flag `--status`. For example, to see the status of unit (board) 0, channel 2, enter:

```
msdv_snap -u 0 -c 2 --status
```

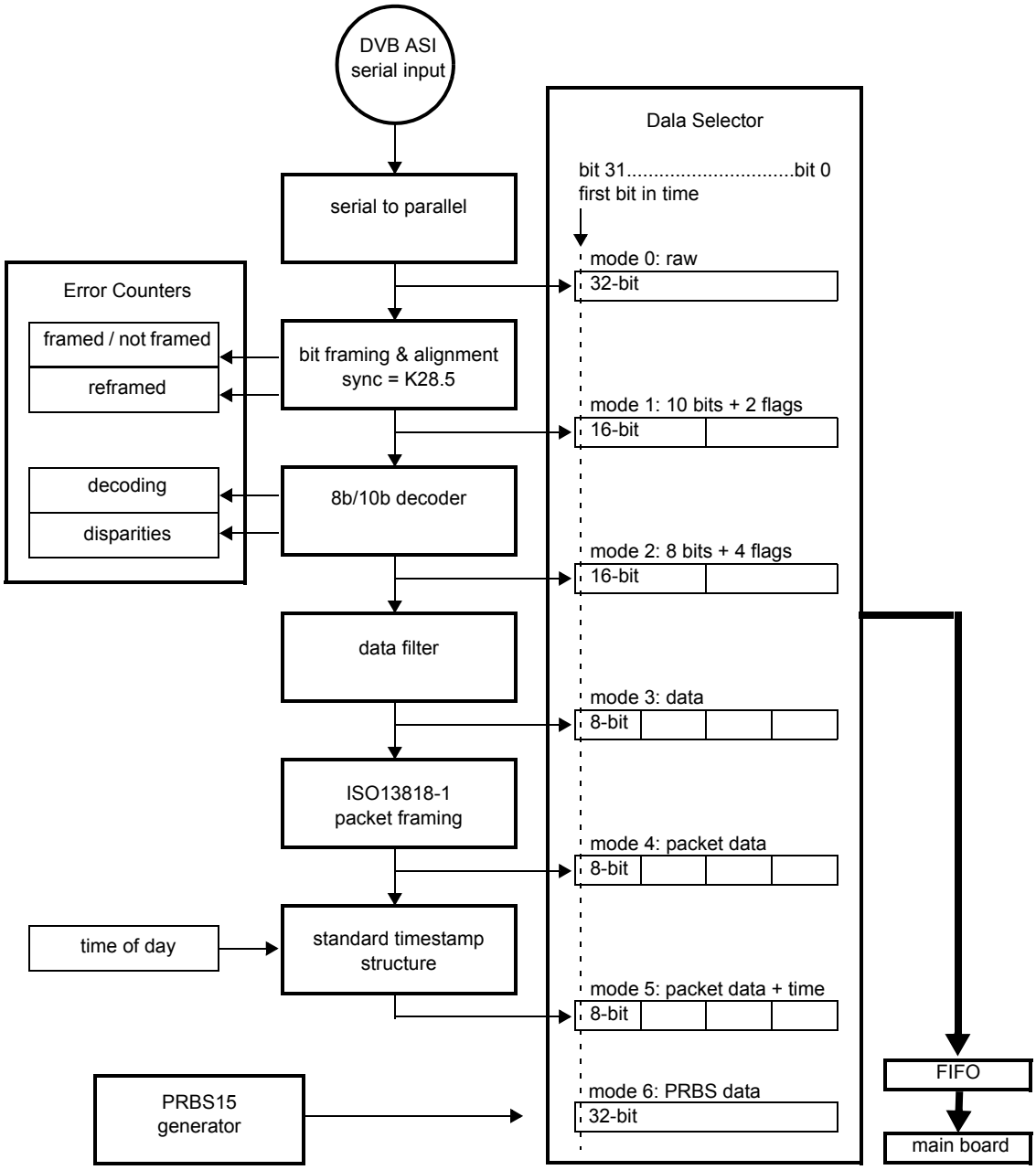
An example of the resulting status message is given below.

```
   MSDV Status

   Channel Enable = off         off          on          on

   Port Config
   direction     = input        input        input       input
   definition    = hidef        hidef        hidef       hidef
   mute          = off          off          off         off
   bypass        = off          off          off         off
   error count   = enabled      enabled      enabled     enabled

   Data Config
   mode          = TS frames    TS frames    TS frames   TS frames
   invert prbs   = on           on           on          on
   filter idle   = off          off          off         off

   Port Status
   carrier det    = on          on           on          on
   reset          = off         off          off         off
   lock           = on          on           on          on
   bit align      = off         off          off         off
   TS frame       = off         off          off         off
   TS 206         = off         off          off         off

Reframe Count    =  204         204          204         204
Coder Errors     =  204         204          204         204
Disparity Errors =  204         204          204         204
```

MSDV Modes

The MSDV board can capture data in six ways, plus two modes for testing. These modes are illustrated in Figure 1:

Figure 1. MSDV Modes



Operating Modes

Mode 0 — Raw All bits captured are output to the DMA channel in precisely the same order.

Mode 1 — Aligned

Each ten bits are padded to 16 bits. The additional six bits are the most significant bits: the first output (in time) and the leftmost bits in memory. These bits are as shown in Table 1:

Table 1. Structure of 16-bit Word in Mode 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	ten-bit aligned flag	new align-ment flag	ten-bit data									
byte 1								byte 0							

Mode 2 — 8b/10b

Mode two is for a signal encoded in 8b/10b. Each ten bits is changed to the corresponding 8-bit value, according to 8b/10b decoding. An additional eight bits are added, to make a 16-bit value. The additional eight bits are the most significant bits: the first output (in time) and the leftmost bits in memory. These bits are as shown in Table 2:

Table 2. Structure of 16-bit Word in Mode 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	code valid flag	dispar-ity error flag	code error flag	control or data word?	8-bit decoded data							
byte 1								byte 0							

Mode 3 — Data Mode three is also for a signal encoded in 8b/10b. Each ten bits is changed to the corresponding 8-bit value, according to 8b/10b decoding. Only the error-free data words are transmitted; control and error characters are deleted.

Mode 4 — Transport Stream Frames

Transport stream frames are output. If bit 4, FILTER_IDLE, is set in the Data Configuration Registers, idle packets are deleted from the output.

Mode 5 — Timestamped Transport Stream Frames

Transport stream frames are output. Idle packets are deleted from the output and timestamps are added according to the following scheme: TBD

Testing Modes

Mode 6 — PRBS15

In order to test the board, the onboard PRBS15 code generator generates PRBS15 code, which the MSDV outputs to the DMA channel.

Mode 7 — reserved

Mode 7 is reserved for internal EDT testing.

Framing

The MSDV has framing capability. ISO/IEC 13818-1 specifies a 188-byte packet; the first byte always has a value of 0x47, followed by three header bytes. No other regularly occurring byte within the packet should be 0x47.

The additional three header bytes are, from the most significant position to the least:

- the transport error indicator bit,
- the one-bit payload unit start indicator bit,
- 13 bits of program ID,
- a two-bit transport scrambling control flag,
- a two-bit adaptation field control flag, and
- a four-bit continuity counter (0–16 within each program ID, to determine if a packet was dropped).

EN 300 429 specification V 1.2.1 specifies an alternative 204-byte transport stream packet with additional error detection and correction data (possible scrambled).

For additional details, see the specificatiosn in References on page 23.

Configuring the MSDV

Configuring the MSDV mezzanine board requires:

- loading the correct firmware in the PCI FPGA on the mezzanine board, and
- loading the correct firmware in the PCI FPGA on the main board.

The following section explains these basic steps, which are sufficient to conduct testing . However, these steps alone are not sufficient to begin data acquisition, which requires additional board initialization. For this procedure, see Initializing the MSDV on page 11.

If you run `msdv_snap`, described in About msdv_snap on page 5, this is done for you.

In the instructions below, placeholders appear in italics; replace these with the values required in your own case.

To configure the mezzanine board's FPGA with the correct firmware, enter:

```
mezzload -u unit_number
```

To configure the board by unit number and to specify a different firmware file:

```
mezzload -u unit_number filename
```

The `mezzload` program detects and loads the main board user interface FPGA with `msdv.bit` if it is not already loaded. If it is already loaded and you want to reload it, use the `bitload` utility:

```
bitload -u unit_number msdv
```

Automatic Configuration

The utility `initpcd` takes, as an argument, a configuration file, and then automatically runs the pertinent command (or commands) of those discussed above. This utility loads the bitfiles, programs

the registers, sets the clocks (if necessary), and gets the MSDV mezzanine board ready to perform DMA.

If you use `initpcd` to configure the MSDV, your application can concern itself solely with performing DMA and other application-specific operations; it will therefore omit MSDV-specific operations and be portable to other EDT boards that perform DMA.

To configure the MSDV, enter:

```
initpcd -u unit_number -f msdv.cfg
```

replacing `unit_number` with the number of the board (by default, 0).

NOTE The configuration files are editable text files. If the files provided don't meet your needs, copy and modify the one that's closest to your required configuration, then run `initpcd` with your new configuration file.

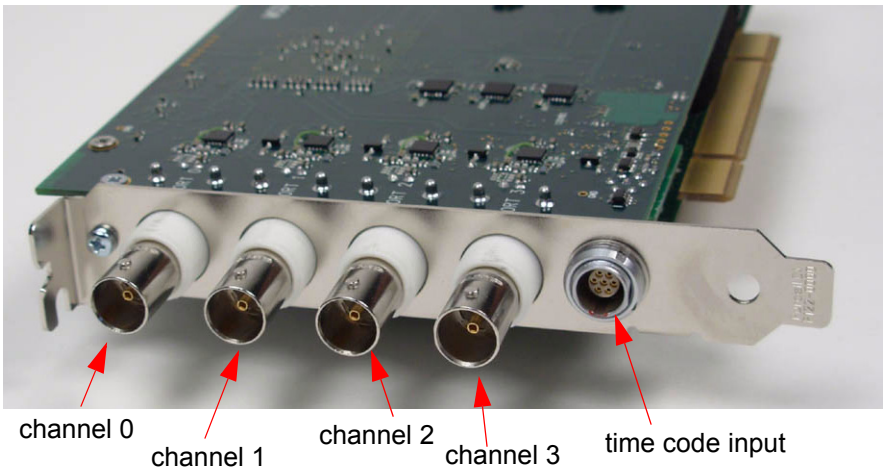
Loading the PCI FPGA Firmware

For the MSDV to operate correctly, the PCI FPGA PROM needs to be loaded with either the `pciss4` or `pcigs4` FPGA configuration file. For the procedure to check or load the correct FPGA configuration file, see the PCI SS, PCI GS, or PCIe8 LX User's Guide, EDT document number 008-01826.

Connector Pinout

The BNC connectors connect to the channels as shown in Figure 2. Each channel can be either input or output.

Figure 2. Connector Pinout



Time Code Inputs

The MSDV includes a Lemo connector and circuitry to accept time code inputs from an IRIG-B or one-pulse-per-second signal source, which you can use to timestamp transport stream frames.

The connector pinout and associated circuitry is described in the PCI/GS or PCIe8 LX Time Distribution Board User's Guide.

Initializing the MSDV

The example application `msdv_snap` initializes the MSDV by synchronizing the digital PLLs that produce the clock signals and enabling the channels. if you're using `msdv_snap`, this initialization is done for you. If, however, you are writing your own application for the MSDV, the correct initialization sequence is given below.

The C library routine names referenced in this procedure are prefixed with `edt_msdv`. These routines can all be found on your software distribution CD, in the main directory, in the files `lib_msdv.c` and its header file `edt_msdv.h`. They are documented in the EDT DMA Software Library; for additional references, see Related Manuals on page 1.

NOTE In the following procedure, as well as in the register descriptions, if a bit is *set*, then its value is one; if it is *clear*, then its value is 0.

Initializing the Clock Signals and Logic Circuits

This procedure initializes basic board operations and communication between the main and mezzanine boards. The MSDV mezzanine board has an FPGA which must communicate with the UI FPGA on the main board. All communication between the boards is synchronized to the 100 MHz reference clock signal. This clock signal is initialized automatically upon main board startup, but the clock signal must then be sent to the mezzanine board and distributed to its FPGA by means of a low-skew clock buffer on the mezzanine board. Until both FPGAs are synchronized to the reference clock signal, the main board cannot read or write any registers on the mezzanine board (addresses 0x80 – 0xF0).

The C library routine `edt_msdv_base_init` determines the kind of board installed in your system, loads the main and mezzanine FPGA configuration files if necessary, and then performs the steps below. Optionally, you can include an argument to set the operating mode: these are described in MSDV Modes on page 7.

`edt_msdv_base_init` uses a structure — `cfg` — to set SSWAP and BSWAP (bits 0 and 3 in the Configuration Register at address 0x0F) as necessary for your host, as well as to set other fields. For details on filling in the fields in this structure, see the EDT DMA Software Library documentation; for an example of its use, see the example application `msdv_snap.c`.

To initialize the clock signals and logic circuits:

1. Load the appropriate firmware as described in Framing on page 9. Among other actions, this initializes the PLL on the main board that generates the 100 MHz reference clock.

   If the firmware is already loaded and you don't wish to reload it, you can instead clear the Command Register at address 0x00 and the MSDV System Clock Enable Register at address 0x24. Write all zeroes to these registers instead of reloading the firmware.

2. To initialize the logic circuits in the main board UI FPGA, set the CMD_EN bit (bit 3) in the Command Register (address 0x00).

3. All channels on the mezzanine board use the same FPGA that receives the 100 MHz reference clock signal. To synchronize incoming data, it must be synchronized to the main board UI FPGA reference clock. To do so, set the SYS_EN bits (bit 2) in the MSDV System Clock Enable Register (address 0x24).

4. To ascertain that the mezzanine board FPGA has been synchronized, check the SYS_LOCK bit (bit 5) of both MSDV System Clock Enable Register (0x24) to ensure that it is set.
5. To ascertain that the main board UI FPGA has been synchronized, check the LOCAL_SYS_LOCK bit (bit 0) of the Status Register at address 0x03 to ensure that it is set.

   Include a timeout in your application at this point. (The library routine `edt_msdv_base_init` does so.) The bits in steps 4 and 5 should be set within approximately 10 ms of setting the bits in steps 2 and 3. If they are not, this probably indicates a fault in the board. In such a case, without a timeout, your application would hang. A wait of approximately half a second should be more than enough time.

   If the bits in steps 4 and 5 are still not set, do not continue with the initialization procedure. Instead, contact EDT.

The main board is now set up correctly. Basic initialization is now been complete and does not need to be done again until the host computer is power-cycled.

(The C library routine `edt_msdv_base_init` demonstrates initializing the clock signals and logic circuits. Steps 2–5 are accomplished by its calling `edt_msdv_lock_clocks`.)

Other Application Setup

If you have not called `edt_msdv_base_init` with an argument to set the operating mode, then at this point, your application must set the operating mode: these are described in MSDV Modes on page 7. The `edt_msdv_base_init` defaults have been chosen to operate correctly with a DVB ASI signal, but if you wish, you can also set or bypass equalization and set or clear muting, using the routines `edt_msdv_set_bypass` and `edt_msdv_set_mute`. (Corresponding `_get_` routines return the status of these parameters.)

NOTE If muting is set, input is disabled.

The channels can be set up independently; you can configure one while another is acquiring data.

The library routines and example application are listed in About the Software and Firmware on page 2.

Enabling and Verifying the Input Signal

The mezzanine board FPGA has a digital phase-locked loop (PLL) that must be locked to the incoming data. If no data is incoming, or if it has been interrupted, the PLL must be reset.

Determine that the receiver has detected a carrier signal by checking bit 0 (CARRIER_DETECT) of the Port Status Registers.

Determine that the reference clock has been set correctly and is locked by checking bit 2 (PLL_LOCK) of the Port Status Registers.

Initializing and Enabling the Data Paths

1. Configure DMA as required, using the library routine `edt_configure_ring_buffers`.
2. Start DMA using the library routine `edt_start_ring_buffers`.
3. Set up DMA and enable the channels using the C library routine `edt_msdv_channel_start`.

Registers

The following legacy registers are implemented but not used:

- Data Path
- Function
- Channel Direction
- Channel Edge
- Channel Framing Status
- In addition, addresses 0x80 and 0x81 are reserved.

Command Register

Size	8-bit
I/O	read-write
Address	0x00
Access	PCD_CMD

Bit	Name	Description
7–4		not used
3	CMD_EN	Set this bit, and enable the required channels in the Channel Enable Register, for DMA to occur. When clear, resets all channels, flushes the FIFOs, and clears all under- and overflow bits.
2–0		not used

Status Register

Size	8-bit
I/O	read-write
Address	0x03
Access	PCD_STAT

Bit	Name	Description
7–1		not used
0	LOCAL_SYS_LOCK	The 100 MHz clock used for data transfers between the main and mezzanine boards is locked in the main board's UI FPGA to the main board reference clock.

Main Board FPGA Configuration File Organization Register

Size 8-bit

I/O read only

Address 0x05

Access PCD_ORG

Bit	Description
7–0	A byte specifying the organization that created the FPGA configuration file currently loaded in the UI FPGA on the main board. An FPGA configuration file produced by EDT returns the value 0xFF.

Configuration Register

Size 8-bit

I/O read-write

Address 0x0F

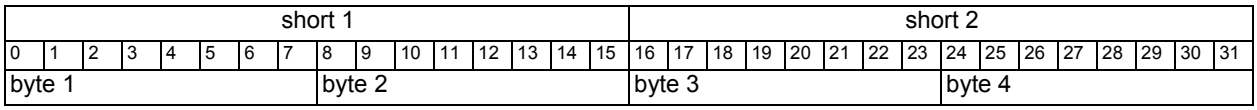
Access PCD_CONFIG

Bit	Name	Description
7–4		not used
3	SSWAP	Swaps the position of the two 16-bit short words in one 32-bit data word, so that *short 2* is transferred before *short 1*. Does not change the order of the bits within each short. See Figure 3 for details of the data word structure.
2–1		not used
0	BSWAP	Swaps the position of bytes 0 and 1, and also bytes 3 and 4, in a 32-bit data word, so that the bytes are positioned 1, 0, 3, 2. Does not change the position of the bits within each byte. See Figure 3 for details of the data word structure.

NOTE The Least Significant Bit First Register can also affect the way in which data is ordered.

Figure 3 shows the structure of a 32-bit data word, with no swapping in effect. With SHORTSWAP set, short 0 appears before short 1. With BYTESWAP set, byte 2 appears before byte 3, and byte 0 before byte 1. With both set, byte 0 appears first, followed by byte 1, byte 2, and finally byte 3.

Figure 3. Data Word Structure Without Swapping



Channel Enable Register

Size 16-bit

I/O read-write

Address 0x10

Access SSD16_CHEN

Bit	Name	Description
7–4		not used
3–0	CH_ENABLE[3–0]	Set to 1 to enable the corresponding I/O channel. The I/O channels are assigned as follows: • Bit 0 enables received data from channel 0. • Bit 1 enables received data from channel 1. • Bit 2 enables transmit data for channel 0. • Bit 3 enables transmit data for channel 1. Clear a bit to reset the respective channel.

Least Significant Bit First Register

Size 16-bit

I/O read-write

Address 0x16

Access SSD16_LSB

Bit	Name	Description
7–2		not used
1–0	LSB_FIRST[1–0]	When set, the least significant bit of the 32-bit data word is the first bit, and the most significant bit is the last. When clear for a channel, the most significant bit of a 32-bit word is the first bit.

NOTE Byte Swap and Short Swap in the Configuration Register can also affect the order of bits in a 32-bit word.

Underflow Register

Size 8-bit

I/O read only

Address 0x18

Access SSD16_UNDER

Bit	Name	Description
7–4		not used
3–0	UNDERFLOW[3–0]	A value of 1 in a bit indicates that the corresponding channel's internal FIFO has underflowed since the previous CMD_EN or CHANNEL_ENABLE. Underflow causes the corresponding channel to transmit the last valid byte repeatedly until it receives new DMA data. Reset by first disabling, then re-enabling, the channel (see the Channel Enable Register).

Overflow Register

Size 8-bit

I/O read only

Address 0x1A

Access SSD16_OVER

Bit	Name	Description
7–4		not used
3–0	OVERFLOW[3–0]	A value of 1 in a bit indicates that the corresponding channel's internal FIFO has overflowed since the previous CMD_EN or CHANNEL_ENABLE. Data received while the FIFO is in overflow is discarded. Reset by first disabling, then re-enabling, the channel (see the Channel Enable Register).

MSDV System Clock Enable Register

Size 8-bit

I/O read-write

Address 0x24

Access MSDV_SYS_CLOCK_ENABLE

Comment Used to synchronize the clocks on the main board's UI FPGA and the mezzanine board's FPGA.

Bit	Name	Description
7–6		Reserved; reads zero.
5	SYS_LOCKED	Set when system clock phase-locked loop is locked.
4–3		Reserved; reads back what was last written.
2	SYS_EN	When set, enables the phase-locked loop of the 100 MHz system clock used to transfer data from the main to the mezzanine board.
1–0		Reserved; reads back what was last written.

MSDV FPGA Load Registers

Size four 8-bit registers

I/O read-write

Address 0x40 – 0x43

Comment Used by `mezzload` to configure the FPGA parts on the MSDV Mezzanine board. Do not write these registers.

Main Board FPGA Configuration File Design ID Register

Size 16-bit

I/O read only

Address 0x7C, 0x7D

Access PCD_DESIGN_ID

Bit	Description
15–0	A sixteen-bit number assigned by the organization that produced the FPGA configuration file loaded in the main board UI FPGA. (EDT uses the top eight bits only.) The design ID for `msdv.bit` is 0x0800.

Main Board Configuration File Version String Register

Use this register to read the FPGA configuration file version string from ROM. Write the ROM address to the register and read the ASCII data from the same register. The version string is a maximum of 64 bytes long, so only the first six bits of the address are significant.

Size 8-bit

I/O read-write

Address 0x7E

Access MAIN_BITFILE_VERSION

Bit	Name	Description
7–0	ID_ADD_DATA	Write an address to read ROM contents. Result is *mainBoard_mezzBoard_bitfileName version.revision mm/dd/yyyy (number of DMA channels used, number of DMA channels required by the PCI FPGA).* The date given is the date the FPGA configuration file was created. Placeholders in italics are replaced by actual values — for example, `gs4_msdv_msdv 1.0 12/01/2007 (4,4).`

Board ID Register

Size 8-bit

I/O read-write

Address 0x7F

Access EDT_BOARDID

Comment Returns a unique four-bit code corresponding to the mezzanine board installed. A value of 0x02 indicates an extended board ID. To read an extended board ID code, use the application `extbdid` or the EDT DMA library routine `edt_get_boardID`.

Board IDs 0x0–0xF (except 0xA) are specified using the bottom four bits of the board ID register at address 0x7F. Board IDs greater than 0xF (and 0xA), and all new boards, use the extended board ID — a value hard-wired into a nonvolatile complex programmable logic device (CPLD). EDT boards read this device using the application `extbdid` (which can read any board ID, extended or not). The application `extbdid` first seeks the identifier in the board ID register; if it finds a value of 0x2, then it seeks the extended board ID from the device instead.

The board IDs listed in the tables below have been assigned by EDT. The first table specifies board IDs residing in the board ID register at address 0x7F.

Table 3. Mezzanine Board Identifiers in the Board ID Register

Bits in register 0x7F				Identifier	Description
3	**2**	**1**	**0**	**(hex)**	
0	0	0	0	0	RS-422 I/O
0	0	0	1	1	LVDS I/O
0	0	1	0	2	reserved — indicates the board uses an extended board ID
0	0	1	1	3	reserved
0	1	0	0	4	reserved
0	1	0	1	5	HRC for E4, STM-1, OC3
0	1	1	0	6	OCM
0	1	1	1	7	Combo II I/O, LVDS
1	0	0	0	8	ECL I/O
1	0	0	1	9	TLK1501 I/O
1	0	1	0	A	reserved
1	0	1	1	B	Combo III I/O, RS-422
1	1	0	0	C	Combo III I/O, LVDS
1	1	0	1	D	Combo III I/O, ECL
1	1	1	0	E	Combo II I/O, RS-422
1	1	1	1	F	Combo I/O, ECL

The following table specifies board identifiers hard-wired into the complex programmable logic device.

Table 4. Mezzanine Board Identifiers in the CPLD

Identifier	Description
A	SRXL (with Graychips)
10	16TE3
11	OC192
12	3x3G
13	MSDV
14	SRXL2

Port Configuration Registers

Size 8-bit

I/O read-write

Address Ch 0: 0x90
Ch 1: 0x98
Ch 2: 0xB0
Ch 3: 0xB8

Access MSDV_CH*x*_PORT_CONFIG (*x* is an integer 0–3

Bit	MSDV_	Description
7	ERR_COUNT_EN	Set to enable the error counters.
6–5		not used
4	BYPASS	Set to bypass cable equalization, possibly useful for a very short cable.
3	MUTE_INPUT	Set to disable input.
2	XMIT_STDDEF	Set to specify 270 MB output (the standard definition); clear otherwise.
1		not used
0	OUT_ENABLE	Set to enable output; clear for input.

Data Configuration Registers

Size 8-bit

I/O read-write

Address Ch 0: 0x91
Ch 1: 0x99
Ch 2: 0xB1
Ch 3: 0xB9

Access MSDV_CH*x*_DATA_CONFIG (*x* is an integer 0–3)

Bit	MSDV_	Description
7–5		not used
4	FILTER_IDLE	Set to delete idle packets from output. Applies to Mode 4 data capture only.
3	INVERT_PRBS	Set to invert the PRBS15 test data.Applies to Mode 6 data capture only.
2–0	MODE_*string*	Set for the desired operating mode, replacing *string* with one of the following symbols: 000 RAW 001 BIT_ALIGNED 010 DECODED 011 DATA 100 TS_FRAMES 101 TIMESTAMP 110 PRBS15 111 TEST These correspond to the modes described in MSDV Modes on page 7.

Reframe Error Count Registers

Size	8-bit
I/O	read only
Address	Ch 0: 0x92 Ch 1: 0x9A Ch 2: 0xB2 Ch 3: 0xBA
Access	MSDV_CH*x*_REFRAME_COUNT (*x* is an integer 0–3)

Bit	Description
7–0	The number of reframe errors since the error counters were last enabled by setting bit 7 of the Port Configuration Registers.

Disparity Error Count Registers

Size	8-bit
I/O	read only
Address	Ch 0: 0x93 Ch 1: 0x9B Ch 2: 0xB3 Ch 3: 0xBB
Access	MSDV_CH*x*_DISP_ERR_COUNT (*x* is an integer 0–3)

Bit	Description
7–0	The number of disparity errors since the error counters were last enabled by setting bit 7 of the Port Configuration Registers.

Decoder Error Count Registers

Size	8-bit
I/O	read only
Address	Ch 0: 0x94 Ch 1: 0x9C Ch 2: 0xB4 Ch 3: 0xBC
Access	MSDV_CH*x*_CODE_ERR_COUNT (*x* is an integer 0–3)

Bit	Description
7–0	The number of decoder errors since the error counters were last enabled by setting bit 7 of the Port Configuration Registers.

Port Status Registers

Size 8-bit

I/O read only

Address Ch 0: 0x95
Ch 1: 0x9D
Ch 2: 0xB5
Ch 3: 0xBD

Access MSDV_CH*x*_PORT_STATUS (*x* is an integer 0–3)

Bit	MSDV_	Description
7–6		not used
5	TS_204	Set if the detected transport stream frames are 204 bytes; clear for 188 bytes.
4	TS_FRAME	Set if transport stream framing is detected (a value of 0x47 at the start of every 188- or 204-byte packet).
3	BIT_ALIGN	Set if the signal is the 8b/10b sync control character (K28.5) has been detected twice in five characters.
2	PLL_LOCK	Set if the internal reference clock is within the proper range for receiving digital video data. If this bit is clear, and the channel reset bit is clear, the reference clock has not been set, or the board is defective.
1	CHANNEL_RESET	Set if the channel has been reset. This bit is the logical NOR of bit 3 in the Command Register and the corresponding channel enable bit in the Channel Enable Register.
0	CARRIER_DETECT	Set if the digital video receiver has detected a carrier signal.

Mezzanine Configuration File Version String Register

Use this register to read the FPGA configuration file version string from ROM. Write the ROM address to the register and read the ASCII data from the same register. The version string is a maximum of 64 bytes long, so only the first six bits of the address are significant.

Size 8-bit

I/O read-write

Address 0xA0

Access MEZZ_BITFILE_VERSION

Bit	Name	Description
7–0	ID_ADD_DATA	Write an address to read ROM contents. Result is *bitfileName version.revision mm/dd/yyyy*. The date given is the date the FPGA configuration file was created. Placeholders in italics are replaced by actual values — for example, `dvb_asi 1.0 12/01/2007`.

Mezzanine FPGA Configuration File Organization Register

Size	8-bit
I/O	read only
Address	0xA1
Access	PCD_MEZZ_ORG

Bit	Description
7–0	A byte specifying the organization that created the FPGA configuration file currently loaded in the FPGA for the specified channel on the mezzanine board. An FPGA configuration file produced by EDT returns the value 0xFF.

Mezzanine FPGA Configuration File Design ID Register

Size	16-bit
I/O	read only
Address	0xA2, 0xA3
Access	MEZZ_DESIGN_ID

Bit	Description
15–0	A sixteen-bit number assigned by the organization that produced the FPGA configuration file loaded in the specified channel of the mezzanine board FPGA. (EDT uses the top eight bits only.) The design ID for `dvb_asi.bit` is 0x0900.

References

ISO/IEC 13818-1 specification. *Information technology — Generic coding of moving pictures and associated audio information: Systems.* Reference number ISO/IEC 13818-1:2007. Available from: www.iso.org/iso/catalogue_detail?csnumber=44169

EN 50083-9 specification. *Cable networks for television signals, sound signals, and interactive services. Part 9: Interfaces for CATV/SMATV headends and similar professional equipment for DVB/MPEG-2 streams.* Reference number BS EN 50083-9:2002. Available from: www.standardsdirect.org/standards/standards2/StandardsCatalogue24_view_14196.html

EN 300 429 specification V 1.2.1. Digital Video broadcasting; Framing structure, channel coding, and modulation for cable systems. Reference number ETS 300 429. Available from: www.ebu.ch/en/technical/publications/euro_standards/index.php