



DNBFC S12 PCIe Software Manual

Copyright (c) 2011 Dini Group, La Jolla Inc.

Last Updated 2011-10-07 I. Yulaev

Dini Group

7469 Draper Ave

La Jolla, CA 92037

United States

Phone: (858) 454-3419

Contact:

Support: support@dinigroup.com

Sales: sales@dinigroup.com

Web: <http://www.dinigroup.com>

Table of Contents

1. Introduction	6
1.1. References	6
1.2. Terminology and Conventions	6
2. Quick Start Guide	7
2.1. Setting Up Your Board.....	7
2.2. Load the Driver.....	7
2.2.1. Linux.....	7
2.2.2. Microsoft Windows.....	7
2.3. Run AETEST	7
2.3.1. Linux.....	7
2.3.2. Microsoft Windows.....	7
3. Communicating with the board	8
3.1. PCIE bus.....	8
3.2. Target access.....	8
3.2.1. Ease of use	8
3.2.2. Bandwidth	8
3.2.3. Latency	8
3.2.4. Functions.....	8
3.2.5. Timeout issues	8
3.2.6. BAR1, BAR2, BAR4.....	9
3.2.7. Multiple windows in one BAR.....	9
3.3. DMA	9
3.3.1. Why use DMA	9
3.3.2. Bandwidth	9
3.3.3. Functions.....	9
3.3.4. Timeout issues	9
3.3.5. To be moved to the driver	10
3.4. Daughtercard header	10
3.4.1. GPIO pins.....	10

3.4.2.	GTP	10
3.4.3.	Aurora	10
3.5.	DDR3 SDRAM chips	10
3.6.	Semaphores for exclusive access	10
3.7.	Memory map.....	11
3.7.1.	Amount of available memory space per FPGA	11
3.7.2.	Reference Design Address Space Mapping.....	11
3.8.	Interrupts	12
3.8.1.	DMA interrupts	12
3.8.2.	User FPGA interrupts	12
4.	AETEST.....	13
4.1.	Structure of AETEST	13
4.2.	Command line arguments.....	13
4.2.1.	-d=[#]	13
4.2.2.	-selftest	13
4.2.3.	-osc=[100].....	14
4.2.4.	-low.....	14
4.2.5.	-reseth=[0/1]	14
4.2.6.	-gfpga=<filename.bit>	14
4.2.7.	-dfpga=<filename.bit>.....	14
4.2.8.	-erase.....	14
4.2.9.	-mhz=[32 to 700]	15
4.2.10.	-pcie=<filename.bit>	15
4.2.11.	-temp.....	15
4.2.12.	-fan=[duty cycle percentage]	15
4.2.13.	-user.....	15
4.2.14.	-mutex	15
4.2.15.	-diotemp.....	15
1.1.	sample_customer_application()	17
1.2.	Menus mode of AETEST	17
1.2.1.	Power Waste	17
1.3.	Misc functions.....	18

1.3.1.	Target read/write.....	18
1.3.2.	DMA	19
1.3.3.	Eeprom.....	19
1.3.4.	temperature sensors.....	19
1.3.5.	programming fpgas.....	19
1.3.6.	setting oscillator frequencies.....	20
1.3.7.	reset	20
2.	Board Runtime Monitoring	21
3.	Board Firmware Upgrade.....	22
4.	Erasing All Non-Volatile Storage	23
4.1.	Flash for PCIE FPGA configuration	23
4.2.	EEPROM for board info	23
4.3.	Battery backed encryption key	23
5.	Additional Software Features	24

1. Introduction

Congratulations on your purchase of a DNBFC_S12_PClE Algorithm Acceleration board!

Please read the Hardware Manual before reading this Software Manual.

This document describes the software interface to the DNBFC_S12_PClE.

1.1. References

This document makes some reference to other pieces of documentation. These include:

- Hardware Manual: DNBFC_S12_PClE Hardware Manual.pdf
- The board schematic. It is titled “503-0197-0000_02_customer.pdf” and should be found on your user CD.
- The board netlist. It is titled “503-0197-0000_02_customer.net” and should be found on your user CD.

1.2. Terminology and Conventions

There is a variety of board-specific terminology that will be used through this document. Much of it will be defined in context, but there are some general terms/names that are useful to know.

gFPGA – describes the twelve FF484-package Spartan-6 chips on this board. These are the “user FPGAs”, and are numbered F0 through F11. **The block diagram in section 2 of this document displays the numbering convention for these FPGAs.**

dFPGA – this is the “dataflow manager” FPGA, the Spartan-6T FF676 package with two daughter card headers and two DDR3 discrete chips.

pFPGA – this is the “PCI Express” controller on this board.

Pcie-e – this is the PCI Express Bus that the board uses to connect to the computer’s motherboard.

DMA – Direct Memory Access – a very fast way of transferring data to/from the board without the host CPU touching each byte.

2. Quick Start Guide

This section of the guide details how to bring up and begin playing with your DNBFC_S12_PClE board. It is recommended that users unfamiliar with Dini Group products or FPGA boards in general follow the instructions, so as to familiarize oneself with the board and avoid common board bring-up mistakes.

2.1. Setting Up Your Board

Follow the directions in the Hardware Manual's quickstart guide.

2.2. Load the Driver

2.2.1. Linux

You will need to compile the driver for linux.

Locate the "Host_Software/PCI_Software_Application/aetest" directory on the CD. Copy this directory to your linux machine where you have the board installed.

In the "Host_Software/PCI_Software_Application/aetest/linuxdrv-2.6" directory, run "make". This will compile the driver for your specific version of the linux kernel (you must have the kernel source installed on the computer for this to work).

Load the driver by running "dndev_load.sh". You must be logged in as Root to do this.

You should now see files such as "/dev/dndev0" "/dev/dndev1".

2.2.2. Microsoft Windows

2.3. Run AETEST

2.3.1. Linux

You will need to compile aestest for linux.

Locate the "Host_Software/PCI_Software_Application/aetest" directory on the CD. Copy this directory to your linux machine where you have the board installed.

In the "Host_Software/PCI_Software_Application/aetest/aetest" directory, run "make". This will compile the driver for your specific version of the linux (i.e. 32-bit or 64-bit).

Run "aetest_linux".

It should report:

Found Device --- v17df, d1903 name="DNBFC_S12_PClE Spartan6 PCI Express 4 lane"

If it doesn't work, run "dmesg" and look at the messages that were reported.

Run "aetest_linux -selftest" to run a bunch of tests on the card.

2.3.2. Microsoft Windows

3. Communicating with the board

The DNBFC_S12_PCIE is designed to plug into a PCIE slot in a computer. All communication to the board is designed to go over that PCIE bus.

Multiple boards can communicate to each other through daughter card cables, but this is only user application communication. Board setup must be done via the PCIE interface.

3.1. PCIE bus

The board can plug into a PCIE slot that is mechanically an x4, x8 or x16 slot. It will operate either x1 or x4 depending on the number of lanes connected in the motherboard.

Both Gen1 (2.5Gbits/second per lane) and Gen2 (5Gbits/second per lane) is supported.

The fastest communication with the host PC will be with an electrical x4 Gen2 slot.

3.2. Target access

3.2.1. **Ease of use**

Software can directly peek and poke registers in the user FPGAs with very little overhead by doing target reads and writes. No driver calls necessary, and with the exception of upper address banking registers, a memory access by software translates directly to a hardware register access.

3.2.2. **Bandwidth**

Target writes to the card are reasonably efficient. ##Mbytes/second is achievable.

Target reads from the card are significantly slower. ##Mbytes/second is achievable.

3.2.3. **Latency**

Target writes to the card can be queued up by the hardware between the host CPU and the FPGAs, and write latency isn't noticed.

Target reads from the card get hit with the entire round trip latency from CPU->pcie->pcie_fpga->user_fpga->pcie_fpga->pcie->CPU, and during that round trip time, the host CPU is doing nothing but waiting for the read transaction to complete. If host CPU processing power is a resource in need, then application should try to avoid using target reads. DMA is more efficient at transferring large blocks of data from the FPGAs to host memory, and should be evaluated for improving system performance.

3.2.4. **Functions**

```
void write_dword_addr64(int64 offset, int32 data);  
  
int32 read_dword_addr64(int64 offset);
```

3.2.5. **Timeout issues**

Target reads across the PCIE bus must complete. If they don't then the PC will hang (not even blue screen, but just hang).

Most PCIE chipsets have a timeout of several million clock cycles. If this timeout expires, then 0xffffffff is returned for the read data.

The pcie_fpga on the DNBFC_S12_PCIE has it's own internal read timeouts. When they expire, then 0xDEADBEE2 or 0xDEADBEE3 is returned.

The user FPGA design should always return on read requests quickly. Read requests should have a deterministic maximum return time. If such a design is not desired, then a status register should be used to indicate that a read can now be completed quickly. If long reads occur in a system, then things like other device's interrupt routines start breaking, and things like the mouse, keyboard, USB, and Ethernet cease to function properly.

3.2.6. BAR1, BAR2, BAR4

3.2.7. Multiple windows in one BAR

3.3. DMA

3.3.1. Why use DMA

DMA is very efficient at moving large chunks of data between the PC and the FPGAs. And DMA can move data to and from the card at the same time on the PCIE bus, with very little speed degradation when both read and write running at the same time.

3.3.2. Bandwidth

PCIE Width (number of lanes)	PCIE Gen1/Gen2	DMA to host Bandwidth	DMA from host Bandwidth
X4	Gen2 (5Gb/s)	TBD	TBD
X4	Gen1 (2.5Gb/s)	610 MB/s	590 MB/s
X1	Gen2 (5Gb/s)	TBD	TBD
X1	Gen1 (2.5Gb/s)	TBD	TBD

3.3.3. Functions

```
dma_data_to_host();
```

```
dma_data_from_host();
```

3.3.4. Timeout issues

Unlike target accesses, DMA transactions are setup by the host CPU, but the CPU doesn't have to touch each byte during the transfer. The DMA engine in the pcie_fpga is responsible for pushing and pulling the data.

The user FPGA has a FIFO interface to transfer data to/from the DMA engines. The user FPGA has no restrictions on how long it can take to respond to a DMA request.

So a theoretical design could use DMA data to start processing, take seconds to finish the processing step, have the dma to host function waiting for data to be pushed back to software through DMA. And during this several seconds of data transfer, the CPU is free to do other processing steps.

3.3.5. To be moved to the driver

The current DMA implementation doesn't use interrupt, and the code is all in user application code. This will soon be moved into the driver with interrupt support. This will make it significantly more efficient for the CPU. And much easier to run multiple applications communicating through DMA to the same board.

3.4. Daughtercard header

The daughter card headers can be used for communication between cards. Either to extend an algorithm across multiple boards, or across multiple computers. No software support is provided for the daughtercard high speed interface.

3.4.1. GPIO pins

These GPIO pins are setup for use as low speed static signals, or for custom daughtercards to be used as I2C or SPI to an on-board memory.

3.4.2. GTP

No software support for these pins. It is up to you and your FPGA design to do something with them. We may eventually add some reference design code to identify which boards are connected together across these cables.

3.4.3. Aurora

The Verilog reference design instantiates an aurora protocol module to communicate over these high speed transceivers on the daughter card connector.

3.5. DDR3 SDRAM chips

The DDR3 SDRAM chips on the user FPGAs have an incomplete verilog reference design.

Currently there is a multi-register access sequence required to read/write them:

```
int read_ddr3();
```

```
void write_ddr3();
```

Eventually this will be changed in the reference design to have the DDR3 SDRAM chips memory mapped in the 64-bit address space.

3.6. Semaphores for exclusive access

Both DMA and target access require a sequence of registers to be written. To make it possible to have multiple programs running at the same time on a single board, there are mutex semaphores

implemented to protect multiple programs from accessing a single shared resource at the same time. These semaphores do slow things down.

3.7. Memory map

There is a 64-bit memory space for the FPGAs on the card. It is suggested that if you implement both DMA and target accesses in your FPGA design, and that you make the memory space identical to both access types.

The memory map (i.e. FPGA address decode) is as follows:

Address[63:48]	FPGA that gets accessed
0x0000	Data flow fpga (U112)
0x0100	gFPGA0 (U100)
0x0101	gFPGA1 (U101)
0x0102	gFPGA2 (U102)
0x0103	gFPGA3(U103)
0x0104	gFPGA4(U104)
0x0105	gFPGA5 (U105)
0x0200	gFPGA6 (U106)
0x0201	gFPGA7 (U107)
0x0202	gFPGA8 (U108)
0x0203	gFPGA9 (U109)
0x0204	gFPGA10 (U110)
0x0205	gFPGA11 (U111)
0xFF00	Pcie_fpga configuration registers

3.7.1. Amount of available memory space per FPGA

Each FPGA is allocated 2⁴⁸ bytes of 64-bit memory space.

3.7.2. Reference Design Address Space Mapping

If you're running our reference design, each FPGA's memory space is further partitioned into 40-bit sections for each feature of the reference design, as follows:

Address[47:40]	Region
0x00	Registers
0x01	DDR3 #1
0x02	DDR3 #2 (dFPGA only)
0x03	Bus Interconnect to higher-numbered FPGA (eg F0->F1, or dFPGA->F0)
0x04	Bus Interconnect to lower-numbered FPGA (eg F0->dFPGA, or F1->F0)
0x05	Bus Interconnect to vertically-connected FPGA (eg F0->F12, no dFPGA connection)
0x06	Aurora connection #1 (dFPGA only)
0x07	Aurora connection #2 (dFPGA only)
0x08	Internal Blockram

3.8. Interrupts

3.8.1. DMA interrupts

See [configuration FPGA DMA user manual](#).

3.8.2. User FPGA interrupts

Drive the user interrupts port over PCIe DDR user interface or SFB user interface. Please reference the [PCIe DMA \(ConfigFPGA\) User Manual](#).

4. AETEST

Overview of the application

Command line mode.

Menus mode.

4.1. Structure of AETEST

Multiple different boards are supported by aetest. There is lots of extraneous code that isn't relevant to this board. The majority of code that is relevant to this board is contained in the following files:

```
m_dnbfcsl2pcie.cpp/.h
```

```
pcie_dma_functions.cpp/.h
```

```
aetest.cpp/.h
```

```
m_diniproducts.cpp/.h
```

```
os_dep.cpp/.h
```

```
pci.cpp/.h
```

Windows and linux have different filenames for the aetest executable: aetest_wdm and aetest_linux.

The executables "aetest_wdm" and "aetest_linux" will be referred to as "aetest" in the rest of this document.

4.2. Command line arguments

Some of the more common high level settings for the board can be run from command line options on aetest.

4.2.1. **-d=[#]**

Multiple boards in a system can be accessed by selecting a different number. The default for this parameter is 0. Boards are numbered starting at 0.

The following commands will put 2 boards into the low power state:

```
aetest -d=0 -off
```

```
aetest -d=1 -off
```

4.2.2. **-selftest**

This option runs the manufacturing test for the board. The programming files for the fpgas need to be put in the `aetest/aetest/bfc_bitfiles/` directory for this command to operate correctly. An output file in the `"test_logs/"` will be created with a summary of the test results.

```
aetest -selftest
```

4.2.3. -osc=[100]

This option sets the GTP oscillator frequency to the specified frequency. Currently only 156.25Mhz is available. Future boards will have some other frequency options.

```
aetest -osc=100
```

4.2.4. -low

This option sets all parts of the board to their lowest power state. It de-configures all of the FPGAs, disables the clock trees, turns of the GTP oscillators, and sets the fans to a low speed.

```
aetest -low
```

4.2.5. -resetn=[0/1]

This option either asserts or deasserts the `soft_resetn` signal to the user FPGAs.

Assert reset.

```
aetest -resetn=0
```

De-assert reset.

```
aetest -resetn=1
```

4.2.6. -gfpga=<filename.bit>

This option sets the filename for programming the 12 user FPGAs. If different filenames are desired for different fpgas, then separate the filenames with commas. Run both the `-gfpga` and `-dfpga` options to program the fpgas.

```
aetest -gfpga=ufpga_top.bit -dfpga=dataflow_fpga.bit
```

```
aetest -  
gfpga=u0.bit,u1.bit,u2.bit,u3.bit,u4.bit,u5.bit,u6.bit,u7.bit,u8  
.bit,u9.bit,u10.bit,u11.bit -dfpga=dataflow_fpga.bit
```

4.2.7. -dfpga=<filename.bit>

This option sets the filename for programming the 1 dataflow user FPGA. Run both the `-gfpga` and `-dfpga` options to program the fpgas.

```
aetest -dfpga=dataflow_fpga.bit -gfpga=ufpga_top.bit
```

4.2.8. -erase

This option deconfigures the 13 user FPGAs.

```
aetest -erase
```

4.2.9. -mhz=[32 to 700]

This option sets the user programmable frequency synthesizer on board for the CLK_PROC tree. Integer Mhz frequencies between 32 and 700 are allowed.

```
aetest -mhz=125
```

4.2.10. -pcie=<filename.bit>

This option upgrades the firmware on the board. Do not power off the board, or kill aetest while this operation is in progress.

```
aetest -pcie=pcie_config.bit
```

4.2.11. -temp

This option prints out the current board temperatures read from the on-board temperature sensors.

```
aetest -temp
```

4.2.12. -fan=[duty cycle percentage]

The on-board fan headers (if installed) can send a PWM'd power to the fans to make them run slower to make them quieter. Set this option to the percentage of the full speed you wish the fans to run at.

```
aetest -fan=50
```

4.2.13. -user

Runs the "sample_customer_application()" function.

```
aetest -user
```

4.2.14. -mutex

If the application crashes during an operation to the board, the software based mutual exclusion code may get broken. Run this option to clear all of them. Be warned that running this option if some existing programs are currently accessing the board can cause unreliable operation.

```
aetest -mutex
```

4.2.15. -diemp

All systems are different for their cooling capabilities. The Spartan6 FPGAs don't have on-die temperature sensors.

But this command will allow you to get a pretty accurate reading of how hot the FPGA's die is in your particular system (with your particular airflow and heatsinks). This is important to help make sure that your design keeps the FPGAs within their maximum temperature range for reliable operation (i.e. meet your timing constraints as reported by Xilinx timing analyzer).

This test does require a long calibration step (20 minutes) to generate a reference correlation between FPGA speed and board temperature, and requires a paper shroud to be put around the board as an insulator. There should be a PDF template of the shroud on the CD called "temperature_shroud.pdf". Cut it out, bend at the lines, and you will need to put it around the board in the beginning of the test (follow the software's directions please). This calibration step will only need to be performed once per board.

The fpga programming files "bfc_bitfiles/dfpga_dietemp.bit" and "bfc_bitfiles/ufpga_dietemp.bit" will need to be present for this test to do anything.

You should measure the power consumption of your design (i.e. current on +12V), and compare it to this design. Then adjust the clock frequency to make this test design match your design. The default clock frequency for this design is 175Mhz which is about 6.2A on +12V, and about 4W per FPGA.

```
aetest -dietemp
```

Here is some typical output when a board (no heatsinks, on a desk, 2 M/s airflow) is run with this command:

```
Approximately 71.200000 Watt/s board, 3.900000 Watts/FPGA
F 0: 54.77 52.19 57.60 die_temp=81.9 C (delta=11.4 C)
F 1: 53.05 55.83 56.85 die_temp=84.8 C (delta=14.3 C)
F 2: 54.67 54.73 57.35 die_temp=80.2 C (delta=16.1 C)
F 3: 53.08 55.13 57.29 die_temp=76.5 C (delta=12.4 C)
F 4: 57.96 54.08 59.25 die_temp=63.8 C (delta=14.4 C)
F 5: 54.44 58.79 59.45 die_temp=61.8 C (delta=12.4 C)
F 6: 57.55 56.78 59.54 die_temp=62.6 C (delta=13.3 C)
F 7: 54.41 58.55 59.24 die_temp=63.7 C (delta=14.3 C)
F 8: 55.47 55.84 57.07 die_temp=73.8 C (delta=9.6 C)
F 9: 52.99 56.09 57.46 die_temp=76.5 C (delta=12.3 C)
F10: 54.79 53.48 56.71 die_temp=79.8 C (delta=9.3 C)
F11: 53.90 55.98 57.54 die_temp=76.6 C (delta=6.1 C)
F12: 52.44 54.47 0.00 die_temp=75.7 C (delta=14.1 C)
```

Note that the "delta=#" is the difference from the roughly calculated die temperature from the closest on-board temperature sensor – but only with these exact same operating conditions: no heatsink, 2 M/s airflow, 3.9W/fpga. About 16 degrees above the temperature sensors is the die temperature for the FPGAs. So a Commercial speed grade FPGA would need to have the board temperature read 85C-16C=69C or less to keep the fpga cool enough to meet the timing constraints as reported by Xilinx Timing Analyzer.

For any of these numbers to be realistic, you need to let the system temperatures stabilize. This takes quite a while, so please be patient. About 10-20 minutes.

1.1. sample customer application()

This example function is provided that does the basic startup calls to make the board do something useful. It is suggested that your application should be created from the building blocks in this function.

General things:

- Set clock frequency
- Program fpgas
- Access those fpgas
- Do some DMA transactions
- Put board into low power state

1.2. Menus mode of AETEST

1.2.1. Power Waste

Option 'H' in the main menu executes an FPGA power waste design. This is meant as a DVT test of the board's power supplies, but you might find it useful for determining system cooling requirements.

Requirements:

- bfc_bitfiles/dfpga_lx150t_80len.bit
- bfc_bitfiles/gfpga_lx150_80len.bit
- Lots of airflow.
- Lots of airflow!
- And LOTS OF AIRFLOW!!!
- Must use +12V power cable (with jumper set to pins 2&3)

Limits:

- The on-board power supplies have current limits at which point they shut off, and the card becomes inaccessible. This is somewhere around 100W to 140W for the entire board.
- The pcie_fpga firmware will (eventually when we finish adding the feature) de-configure the FPGAs when the board temperature gets hot enough that the die temperature of the other FPGAs is above their max temperature rating (85C for commercial, and 100C for industrial speed grade FPGAs).

The FPGA design that is run is a worst case power design. It uses 80% of Slices, 80% of RAMs, 100% of IOs. The internal logic is toggling at about 50% (i.e. pure random), most FPGA designs toggle at about 12.5% (see Xilinx power estimator spreadsheet). The IO pins are all switching every ½ clock cycle. The clock frequency for this design is programmable using the PROC_CLK network.

The software starts the clock frequency at 35Mhz, and keeps increasing by 1Mhz every few seconds (until the board stops responding!). This software will get upgraded so that it senses when either the temperature gets too high, or some of the powers supplies go into current limit.

If the pcie_fpga goes into power limit, it will get re-configured, and all pcie accesses will return 0xFFFFFFFF. You will need to reboot the computer if this happens.

At 35Mhz, the board is drawing approximately 60W of power from the +12V power connector.

At 75Mhz, each FPGA is dissipating about 6W of power (106W for the board), which is approximately the maximum amount of power that the FPGA can dissipate without overheating (i.e. above the 85C die temperature for commercial speed grade chips).

At 110Mhz, each FPGA is dissipating about 9W of power, and the board is drawing about 156W! This is way beyond what the board will be reliable at. Don't run your design (that has to function properly) at this much wattage.

The table below is for your use to see what a "reasonable airflow" system looks like for wattage and temperatures of the board.

PROC_CLK frequency	+12V input current	Watts per FPGA	Max board temperature
FPGAs not configured	1.5 A	0.322 W	?
0 (fpgas configured)	2.1 A	0.322 W	?
35	5 A	2.6 W	30 C (ramp up)
40	5.6 A	3.0 W	38 C (ramp up)
45	6 A	3.3 W	45 C
50	6.5 A	3.7 W	49 C
55	6.9 A	4.0 W	52 C
60	7.4 A	4.3 W	55 C
65	7.8 A	4.7 W	57 C
70	8.4 A	5.0 W	60 C
75	8.9 A	5.5 W	62 C
80	9.5 A	5.9 W	64 C
85	??	??	66 C
90	??		68 C
100	??		72 C
105	??		75 C
110	13 A (estimated)	8.5 W (estimated)	77 C

1.3. Misc functions

Don't modify other functions! Have us fix bugs in our code.

1.3.1. Target read/write

These functions read/write 32-bit words to the board's 64-bit address space. They require multiple accesses to registers, and are semaphore protected for multiple program access. These functions use the bar1 address space to perform the access. These functions are defined in "m_dnbfc12pcie.cpp".

```
void write_dword_addr64(int64 offset, int32 data);
```

```
int32 read_dword_addr64(int64 offset);
```

1.3.2. DMA

These functions transfer data between the user FPGAs and the Host PC's DRAM. These transfers can achieve huge bandwidths. These functions are defined in "pcie_dma_functions.cpp".

```
void dma_data_from_host(int dma_engine_num, int64 board_address,  
void *data, int dword_size, 0, 0, 0);
```

```
int dma_data_to_host(int dma_engine_num, int64 board_address,  
void *data, int dword_size, char *got_eof, 0, 0);
```

There are 3 DMA engines: 0,1,2

The upper bits of board_address are used for routing the DMA transfer to the correct FPGA. The lower bits are FPGA implementation specific.

The "*data" pointer is to the user allocated memory to be transferred to/from the card.

The DMA engine only operates on 32-bit integer sized quantities. It will only transfer multiples of 4 bytes. The dword_size parameter is the number of 32-bit integers to transfer to/from the card. You will probably need to pay attention to 4-byte vs 8-byte alignment, because the DMA engine will pass your software alignment of the data directly through to the user FPGA.

The FPGA can terminate a DMA transfer early by setting the EOF flag. This will cause the dma_data_to_host function to quit early (before transferring the entire requested data).

These 2 DMA functions are in process of being moved into the driver. Please do not try to modify these functions.

1.3.3. Eeprom

The on-board eeprom contains information about the features of the card build. It can be read into a structure by calling:

```
void fetch_eeprom();
```

The data will be in the structure "eeprom_data". See "m_dnbfc12pcie.h" for the definition of the structure.

1.3.4. temperature sensors

To read the on-board temperature sensors. The values (in degrees-C) is saved in temps[0-5].

```
void read_board_temps(double *temps);
```

1.3.5. programming fpgas

To program the 13 user fpgas, call the following functions with the filenames of the Xilinx .bit files. The GFPGAs can be programmed with the same .bit file, or different .bit files by comma separating the filenames in the input string.

```
void program_fpgas(char *dfpga_filename, char *gfpga_filename);
```

1.3.6. setting oscillator frequencies

There are 2 frequencies on the board that are programmable: gtp oscillator, and proc_clk.

Proc_clk (to all 13 user FPGAs) can be programmed to any integer Mhz frequency between 32 and 700.

```
void set_proc_freq(int mhz);
```

The GTP oscillator for the daughtercard interface can be set to 156.25 mhz. Other frequency options are available, but they will need to be specified when ordering the board.

```
void set_gtp_osc(double mhz);
```

1.3.7. reset

There is a global board reset for the user FPGAs. It is automatically de-asserted after programming the FPGAs, but can be toggled later by the user's application.

```
void assert_soft_resetn(void);
```

```
void deassert_soft_resetn(void);
```

2. Board Runtime Monitoring

Temperature sensors

3. Board Firmware Upgrade

When firmware updates are available, there will be a set of files to download from <http://www.dinigroup.com>

The download will consist of several items that will have changed, and they must be used together (partial upgrades will probably not work at all):

- "pcie_config.bit"
- "dnbfcs12_dfpga.bit"
- "dnbfcs12_ufpga.bit"
- Driver for the operating system (Linux source or Windows pre-compiled)
- Source code for aetest.

Run the following steps in this order upgrade the board:

- 1) Run "Aetest_linux -pcie=pcie_config.bit" with the OLD software that you have been running, but the new "pcie_config.bit" from the download.
- 2) Then power cycle the board. It is not sufficient to reset the computer. It must be powered off and then on again.
- 3) Copy all of the software from the download.
- 4) Compile the new version of the driver (for linux), or install the new windows driver.
- 5) Compile the new version of aetest.
- 6) Run "aetest_linux -selftest" to make sure that the hardware and software are working together correctly.

If you do the upgrade in the wrong order (i.e. use the wrong software, and can't go back). Then you will need to replace step 1 above with:

- Get a Xilinx JTAG cable, and plug it into J5 "PFGA JTAG".
- Run the ISE software "IMPACT".
 - o Automatically detect JTAG chain.
 - o Attach a SPI flash to the single FPGA that is found.
 - o Program that SPI flash with an MCS file that is created from the "pcie_config.bit" from the download (the file might already exist in the download).
- Now power cycle the computer, and continue on with the steps above.

Customers can download the current DNBFC_S12_PCIE support package at:

http://www.dinigroup.com/files/cust_cd/DNBFC_S12_PCIE.zip

username: download

password: dlNiCD

4. Erasing All Non-Volatile Storage

There are 3 types of non-volatile storage on this board. If for whatever reason, you need to erase everything on this board, you can erase everything.

4.1. Flash for PCIE FPGA configuration

The 64Mbit flash device can be programmed with any file you like. So to erase it, create a blank file that is 8Mbytes in size, and call:

```
at45.program_flash("8mbytefile.bin");
```

This flash device can be re-programmed by using Impact to program the FPGA over the JTAG interface, and the upgrading the firmware by running "aetest -pcie=pcie_config.bit".

4.2. EEPROM for board info

The 64Kbit eeprom can be programmed with 0's by calling:

```
I2c.write_i2c_device(BFC_I2CADDR_EEPROM,0,&buffer_of_zeros,8*1024,EEPROM_BLOCKSIZE);
```

4.3. Battery backed encryption key

The FPGAs each have an encryption key that is battery backed. Remove the battery. Remove the card from all power sources (i.e. unplug it from the PC). And wait until the voltage across C207 becomes low enough for the FPGAs sram cells to get cleared (0V).

5. Additional Software Features

Ask for what you want... We might provide it.