Pexaria5a - EtherBone

Test Plan

Revision:	1.0
Status:	Draft
Repository:	N/A
Project:	N/A
Folder:	N/A
Document ID:	CSL-TPL-12-yyyyyy
File:	Document4
Owner:	
Last modification:	June 20, 2016
Created:	October 16, 2014

Prepared by	Reviewed by	Approved by
Bor Marolt		

Document History

Revision	Date	Changed/reviewed	Section(s)	Modification
1.0	16. 10. 2014	Initial version	-	-

Confidentiality

This document is classified as a confidential document. As such, it or parts thereof must not be made accessible to anyone not listed in the Audience section, neither in electronic nor in any other form.

Scope

This document defines the Test Plan for the user space software communicating with the Pexaria5a card via the EtherBone library and Wishbone kernel driver. It provides instructions on how to use example programs used for interrupt handling and read/write latency measurements. The audience of this document are the test engineers who will execute the Test Plan and produce the Test Report.

The test plan covers the following aspects:

- Execution of tests confirming the proper installation of Wishbone kernel driver and EtherBone library
- Interrupt handling test procedure.
- Read/write latency tests.

This document is the Test Report of testing the aspects mentioned above.

The audience of this document are the staff responsible for managing development of the system under test and developers.

Audience

Document template and quick guide is targeted to all Cosylab employees and associates.

Table of Contents

Typography

This document uses the following styles:

A box like this contains important information.

A box like this would contain sidebar text.

Warning!
A box like this provides information, which should not be disregarded!

A box like this contains a feature, it should be pointed out.

? Use a box like this to pose a puzzle to the user.

A box like this MUST be omitted from the final documentation.
It is intended to solve any outstanding writing issues.

Glossary of Terms

SDB	Self-Describing Bus

References

[1]

1. Introduction

1.1. Overview

Test setup consists of the following HW and SW components:

- PC running 64bit SL 6.4
- Pexaria5a card inserted in a PCIe slot. Firmware, prepared by Cosylab, introduces the following SDB components relevant to this test plan:
    - Altera-PCIe-MSI-Tgt at base address 0x30000 (interrupt handling test)
    - WB-GPIO-Port-CSLIRQ at base address 0x200 (interrupt handling test)
    - WB4-BlockRAM at base address 0x10000 (read/write latency test)
- Wishbone kernel driver
- Etherbone library
- Test software developed by Cosylab

Test plan consists of 2 levels:

- Building and installing Wishbone kernel module and Etherbone library
- Running the test programs used for:
    - Measurement of latency of read/write operations using the Etherbone library
    - Demonstrating the interrupt handling using the Etherbone library

1.2. Assumptions

64bit Scientific Linux 6.4 properly installed.

1.3. References

[2]

2. Testing team

Tester	email

3. Test report summary

TEST FAILED! Basic quality objectives were not achieved!
ALPHA RELEASE Suitable for internal development.
BETA RELEASE Suitable for release to customers for testing purposes.
RELEASE CANDIDATE Suitable for release to customers for production. Ready for release pending minor modification.
MAINTENANCE RELEASE Suitable for replacement of a system in production.
FACTORY ACCEPTANCE TEST PASSED The system is ready for shipping to the installation site.
SITE ACCEPTANCE TEST PA SSED The system can be used in production.

Table 3-1: List of passed/failed tests *[optional]*

Test level	Total	blocker	critical	major	normal	minor	trivial	enhancement
Error! Reference source not found.								

Table 3-2: List of executed performance tests

Test case	Parameter [unit]	Value	Expected value [min-max]

4. Installation tests

4.1. Install software [TP-INS-0001]

Description

Installation of the Etherbone library, Etherbone tools and Wishbone kernel driver

Pre-requisites

1. 64bit Scientific Linux 6.4 is installed

Test procedure

1. Download the latest Etherbone sources by running:

```
$ git clone –recursive https://github.com/stefanrauch/bel_projects.git
```

Expected result: No errors are produced.

2. Navigate to the install location of the Etherbone library, e.g.:

```
$ cd /usr/local/lib/bel_projects
```

3. Build Etherbone:

```
$ sudo make etherbone
```

Expected result: No errors are produced. If there are errors, install the required dependencies.

4. Build Etherbone tools :

```
$ sudo make tools
```

Expected result: No errors are produced. If there are errors, install the required dependencies.

5. Build Wishbone kernel driver :

```
$ sudo make driver
```

Expected result: No errors are produced. If there are errors, install the required dependencies.

6. Install :

```
$ sudo make install
```

Expected result: No errors are produced.

Severity in case of failure:blocker

Test result: [✓ Not tested] Passed Failed

Comment:

4.2. Linux driver [TP-INS-0002]

Description

Insert kernel modules.

Pre-requisites

1. Test 4.1 has passed

Test procedure

1. Navigate to directory (this step assumes the location is /usr/local/lib/bel_projects):

```
$ cd /usr/local/lib/bel_projects/ip_cores/fpga-config-space/pcie-wb
```

2. Insert the following kernel modules:

```
$ sudo insmod wishbone.ko
$ sudo insmod pcie_wb.ko
```

Expected result: No errors are produced

3. Check if modules were inserted properly:

```
$ lsmod
Module           Size          Used by
…
pcie_wb          13125         0
wishbone         18442         1 pcie_wb
…
```

Also check if no problems were reported when running:

```
$ dmesg
…
[4063.611299]wishbone: version 474f0f0 (2014-07-11 12:09:07 +0200) loaded
[4074.338446] pcie_wb 0000:01:00.0: irq 52 for MSI/MIS-X
…
```

4. Check if wishbone master/slave device files have appeared:

```
$ ll /dev/wbm0
crw-rw---- 1 root wishbone 250, 0 okt 16 16:35 /dev/wbm0
$ ll /dev/wbs0
crw-rw---- 1 root wishbone 249, 0 okt 16 16:35 /dev/wbm0
```

Severity in case of failure:blocker

Test result: [✓ Not tested] ☐ Passed ☐ Failed

Comment:

4.3. List device components [TP-INS-0003]

Description

List SDB components on the card.

Pre-requisites

1. Test 4.2 has passed

Test procedure

1. Run:

```
$ sudo eb-ls dev/wbm0
BusPath      VendorID         Product                   BaseAddress(Hex)  Description
1            0000000000000651:eef0b198                   20000  WB4-Bridge-GSI
1.1          0000000000000651:8a670e73                   30000  Altera-PCIe-MSI-Tgt
2            0000000000000651:2d39fa8b                     400  GSI:BUILD_ID ROM
3            0000000000000651:5cf12a1c                 2000000  SPI-FLASH-16M-MMAP
4            0000000000000651:3a362063                       0  FPGA_RESET
5            0000000000000651:00000815                 1000000  Etherbone_Master
6            0000000000000651:10051981                     100  GSI_TM_LATCH_V2
7            000000000000ce42:66cfeb52                   10000  WB4-BlockRAM
8            000000000000c570:c007cafe                     200  WB-GPIO-Port-CSLIRQ
```

Severity in case of failure:blocker

Test result: [✓ X] Not tested ☐ Passed ☐ Failed

Comment: Component on base Address 0x30000 will be used in interrupt handling test and component on address 0x10000 will be used in read/write latency tests. If you are using a different firmware build, modify the addresses in the following tests accordingly.

4.4. Build test software [TP-INS-0004]

Description

Building the test software

Pre-requisites

2. Test 4.1 has passed

Test procedure

2. Download the pexaria_cpp_demo directory, containing the test programs from Cosylab SVN:

```
$ svn co <URL_TO_BE_DEFINED>
```

Expected result: No errors are produced.

3. Navigate to the location of the checked out software, e.g.:

```
$ cd ~/workspace/pexaria_cpp_demo
```

4. Makefile in this directory assumes that the install location of the Etherbone library is /usr/local/lib. Modify if this is not so – depends on the location of bel_projects directory (assumed location: /usr/local/lib/bel_projects). Run:

```
$ make
mkdir bin
g++ -Wall -g src/interrupt-example.cpp -L
/usr/local/lib/GSI/bel_projects/ip_cores/etherbone-core/api -Wl,-
rpath,/usr/local/lib -letherbone -leca -o bin/interrupt-example
g++ -Wall -g src/multiple-read.cpp -L
/usr/local/lib/GSI/bel_projects/ip_cores/etherbone-core/api -Wl,-
rpath,/usr/local/lib -letherbone -leca -o bin/multiple-read
g++ -Wall -g src/multiple-write.cpp -L
/usr/local/lib/GSI/bel_projects/ip_cores/etherbone-core/api -Wl,-
rpath,/usr/local/lib -letherbone -leca -o bin/multiple-write
```

Expected result: No errors are produced and pexaria_cpp_demo/bin directory appears, containing the following binaries:

```
$ ll bin
drwxrwxr-x 2 user user  4096 okt 16 16:46 ./
drwxrwxr-x 4 user user  4096 okt 16 16:46 ../
-rwxrwxr-x 1 user user 39641 okt 16 16:46 interrupt-example*
-rwxrwxr-x 1 user user 38586 okt 16 16:46 multiple-read*
-rwxrwxr-x 1 user user 40525 okt 16 16:46 multiple-write*
```

If any errors are produced, install the required dependencies.

Severity in case of failure:blocker

Test result: [✓ X] Not tested ☐ Passed ☐ Failed

5. Etherbone demonstration tests

5.1. Read/write latency tests [TP-RWL-0001]

Description

Measuring the latency of read write operations on the pexaria5a card through the PCIe, using Etherbone library.

Pre-requisites

1. Test 4.3 and 4.4 have passed

Test procedure

1. Navigate to directory pexaria_cpp_demo/bin and run:

```
$sudo ./multiple-write dev/wbm0 0x10000 10000 4
Cycle number 1 finished
Cycle number 2 finished
seconds: 0, nseconds: 421142, data transfered: 7280
seconds: 0, nseconds: 155208, data transfered: 2720
```

Expected result: No errors are produced.

Parameters:

- dev/wbm0: Physical address of the card
- 0x10000: Address of the RAM component (see test 4.3)
- 10000: Number of bytes to be written on the component. Multiple-write generates 10 kB of consecutive integers. Size of the integer is specified by the last parameter.
- 4: Number of bytes to be written in a single write operation – It is optimal that this parameter matches the component’s width

Maximum data block that can be written via one cycle (one call to socket.run) is 7280 B. Therefore writing 10 kB is decomposed into two cycles. Multiple-write reports, that both cycles have finished and it displays the number of bytes transferred and seconds, nanoseconds per cycle.

2. To check if *multiple-write* was successful, do:

```
$ sudo ./multiple-read dev/wbm0 0x10000 10000 4
Cycle number 1 finished
Cycle number 2 finished
data[0] = 0
data[1] = 1
data[2] = 2
data[3] = 3
data[4] = 4
data[5] = 5
data[6] = 6
data[7] = 7
data[8] = 8
data[9] = 9
```

```
data[10] = 10
data[11] = 11
...
data[2497] = 2497
data[2498] = 2498
data[2499] = 2499
seconds: 0, nseconds: 1938428, data transfered: 7060
seconds: 0, nseconds: 788929, data transfered: 2940
```

Maximum data that can be read via one cycle (one call to socket.run) is 7060 B. Therefore reading 10 kB is decomposed into two cycles. Multiple-read reports, that both cycles have finished and it reports number of bytes transferred and seconds, nanoseconds per cycle.

To clear the written data use *multiple-write* with the integer size parameter set to 0:

```
$sudo ./multiple-write dev/wbm0 0x10000 10000 0
./multiple-write: 10000 B on address dev/wbm0 will be set to 0.
Cycle number 1 finished
Cycle number 2 finished
seconds: 0, nseconds: 378295, data transfered: 7280
seconds: 0, nseconds: 143582, data transfered: 2720
```

Check if cleanup was successful:

```
$sudo ./multiple-read dev/wbm0 0x10000 10000 4
Cycle number 1 finished
Cycle number 2 finished
data[0] = 0
data[1] = 0
data[2] = 0
data[3] = 0
data[4] = 0
data[5] = 0
data[6] = 0
data[7] = 0
data[8] = 0
data[9] = 0
data[10] = 0
data[11] = 0
...
data[2497] = 0
data[2498] = 0
data[2499] = 0
seconds: 0, nseconds: 1946106, data transfered: 7060
seconds: 0, nseconds: 785517, data transfered: 2940
```

Use 2 bytes per write operation instead of 4:

```
$ sudo ./multiple-write dev/wbm0 0x10000 1000 2
./multiple-write:  Selected  data  size:  2  does  not  match  the  component's  bus
width: 4. Some data might not be written!
Cycle number 1 finished
seconds: 0, nseconds: 164381, data transfered: 1000
```

NOTE: Size per write operation does not match the component’s ‘bus’ width, so a warning is issued.

3. Check what was written by issuing:

```
$ sudo ./multiple-write dev/wbm0 0x10000 1000 2
./multiple-read: Selected data size: 2 does not match the device bus width: 4.
Some data might not be read!
Cycle number 1 finished
data[0] = 0
data[1] = 1
data[2] = 2
data[3] = 3
data[4] = 4
data[5] = 5
data[6] = 6
data[7] = 7
data[8] = 8
data[9] = 9
data[10] = 10
...
data[492] = 492
data[493] = 493
data[494] = 494
data[495] = 495
data[496] = 496
data[497] = 497
data[498] = 498
data[499] = 499
seconds: 0, nseconds: 598617, data transfered: 1000
```

4. Data was successfully written and read, however if a larger data block is chosen:

```
$ sudo ./multiple-write dev/wbm0 0x10000 10000 2
./multiple-write: Selected data size: 2 does not match the component's bus
width: 4. Some data might not be written!
Cycle number 1 finished
Cycle number 2 finished
seconds: 0, nseconds: 391779, data transfered: 7280
seconds: 5, nseconds: 5077293, data transfered: 2720
```

5. Check by issuing a read:

```
$ ./multiple-read: Selected data size: 2 does not match the device bus width:
4. Some data might not be read!
Cycle number 1 finished
Cycle number 2 finished
data[0] = 0
data[1] = 1
data[2] = 2
data[3] = 3
data[4] = 4
data[5] = 5
data[6] = 6
data[7] = 7
data[8] = 8
data[9] = 9
data[10] = 10
…
data[4991] = 0
data[4992] = 0
data[4993] = 0
data[4994] = 0
data[4995] = 0
```

```
data[4996] = 0
data[4997] = 0
data[4998] = 0
data[4999] = 0
seconds: 0, nseconds: 946859, data transfered: 7060
seconds: 5, nseconds: 5104944, data transfered: 2940
```

Not all data is successfully written. This is due to the fact that the read/write operations in the etherbone cycle can not be chained if the write size (2B) does not match the component width (4B).

Similar result is obtained if 1B per read/write is used.

Severity in case of failure: critical

Test result: [✓ X] Not tested ☐ Passed ☐ Failed

Comment:

5.2. Interrupt handling [TP-IRQ-0001]

Description

Trigger interrupts and catch the using the Etherbone library

Pre-requisites

3. Test 4.3 has passed

Test procedure

5. Run:

```
$ sudo ./interrupt-example 0x30000
./interrupt-example: Waiting for interrupt!
```

If a different firmware build is used, be sure to modify the IRQ address accordingly (see test case 4.3)

and press and release the S2 push button. The terminal should display:

```
intrHandler::write called! Printing parameters:
address: 30000
width: D
data: c007beef
intrHandler::write called! Printing parameters:
address: 30000
width: D
data: c007beef
```

First interrupt is triggered on the push of the button and the second interrupt is triggered on the release.

Severity in case of failure: critical

Test result: [✓ X] Not tested ☐ Passed ☐ Failed

Comment:

Appendix A. Requirements traceability matrix

This test plan is designed to verify that requirements are met. The table below lists the requirements (including a link to the document where they are defined), their description, and the test case in this document where they are tested.

It is advised that requirements documents use a scheme for identifying requirements that ensures that the ID doesn't change in an uncontrolled way, so that the IDs listed in the table below are less likely to change.

Table A-1: Requirements traceability matrix

Requirement ID	**Description**	**Test case**
NISYNC-SRS-123 **Error! Reference source not found.** PX-6682	PX-6682 generate_pulse generates strange pulses at start	**Error! Reference source not found.**

Appendix B. Anomaly traceability matrix

This test plan also covers for testing of anomalies that were hitherto discovered (**regression tests**). For each anomaly, the table below provides an ID with a link to more information about the anomaly, and a reference to the test case section in this document where checking for the anomaly is defined.

Table B-2: Anomaly traceability matrix

ID	Description	Test Case