Universitatea "POLITEHNICA" Bucuresti
Facultatea de Electronica si Telecomunicatii
Catedra de Electronica Aplicata si Ingineria Informatiei

CERN-OPEN-2005-016
01 / 07 / 2004

AdvancedTCA Backplane Tester

referat de doctorat

Doctorand: ing. Alexandra Dana OLTEAN
Conducator stiintic: prof. dr. ing. Vasile BUZULOIU

August 2004

Content

1 Introduction

1.1 Advanced Telecommunications Computing Architecture

During 2002 the PICMG (PCI Industrial Computer Manufacturer Group) group of over 100 participating companies had to address the fact that the Compact PCI market had fallen between two targets. It had been too late to capture a significant portion of the embedded VME market and it was too slow, from a technical point of view, to capture the future telecoms infrastructure market. They realized that traditional bus-based architectures would never be able to deliver the required bandwidth and adopted a point-to-point switching architecture instead.

The result was the specification of *Advanced Telecommunications Computing Architecture*, known as AdvancedTCA or ATCA, the largest specification effort in PICMG's history. These new specifications, called PICMG® 3.0, incorporate the latest trends in high speed interconnect technologies, next generation processors and improved reliability, manageability and serviceability. The AdvancedTCA standard was aimed as a modular approach, which enables the use of the same chassis/backplane, with different modules, for multiple types of equipments. The standard attempts to meet the following requirements:

- *Scalable Capacity* of up to 2.5 Tbits/s (per chassis) with a centralized switching hub interconnected to all module slots in a star or a full mesh configuration
- *Modularity and configurability* to enable multiple modules with various interfaces and different technologies and storage media to be mixed and matched for diverse applications in the same platform
- *Redundancy*
- Advanced power distribution and cooling concepts
- Support for multiple types of *Switching Fabrics* (the core of the platform), such as Ethernet (GbE), PCI Express and others

The base specifications for the new AdvancedTCA family are defined in the PICMG 3.0 standard and were adopted in January 2003. PICMG 3.0 provides specifications for electromechanical issues (rack and chassis form factors), interconnect topology and electrical characteristics (backplane connectivity, power, cooling, management interfaces) for modular shelves. Subsequent specs in the PICMG 3.x series define standards for different kinds of protocols to be used with PICMG 3.0 backplane, such as PICMG 3.1 (Ethernet and Fibre Channel), PICMG3.2 (InfiniBand), PICMG3.3 (StarFabric), PICMG3.4 (PCI-Express and Advanced Switching) and PICMG3.5 (Advanced Fabric Interconnect and RapidIO).

The AdvancedTCA standard provides rapid development time, economies of scale coupled with scalability and the ablity to mix different types of modules and technologies in one platform. At the time when we started using AdvancedTCA standard (middle 2003), early availability products were undergoing lab trials by early adopters, and multiple interoperability workshops were held by the PICMG members to ensure multi-vendor compatibility.

1.2 Inside PICMG specifications for AdvancedTCA

The key feature of the AdvancedTCA standard is its very high bandwidth architecture. The base interface defined in PICMG 3.0 specification accommodates essential interoperability over a switched fabric supporting 10/100/1000BASE-T Ethernet always in a Dual Star configuration with redundant hubs. The fabric interface, defined in the PICMG 3.1 specification, is the main channel through which the serial data stream passes, in order to support higher-speed signaling technologies. Its specification supports many different topologies, from Dual Star to Full Mesh, making it a very flexible architecture. For our application (the 10 Gigabit Ethernet switch), a Dual Star Fabric Interface topology has been selected because it offers redundant backplane fabric scheme. This topology requires two dedicated slots for Hub1 and Hub2 boards to be inserted (Figure 1.1).

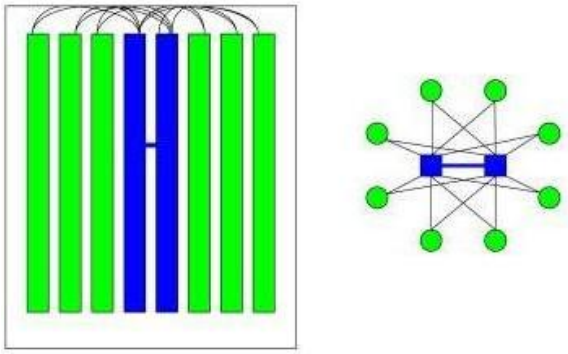


Figure 1.1: Dual Star topology

As defined under the PICMG 3.0 spec, the AdvancedTCA backplane has a maximum of 16 slots within 2 slots are dedicated to the redundant hubs and the rest are used for the Node cards. The backplane is populated with high-speed connectors which are disposed over three zones to interface with ATCA cards, as shown in Figure 1.2. Zone 1 is for power and system management. Zone 3 is for rear I/O access. Zone 2 is the primary *data transport interface* which contains five ZD connectors for four separate interfaces: (1) base interface, (2) fabric interface, (3) update channel interface, and (4) synchronization clock interface. The figure below illustrates the three connector zones on the ATCA backplane and on the matching board to be plugged in.

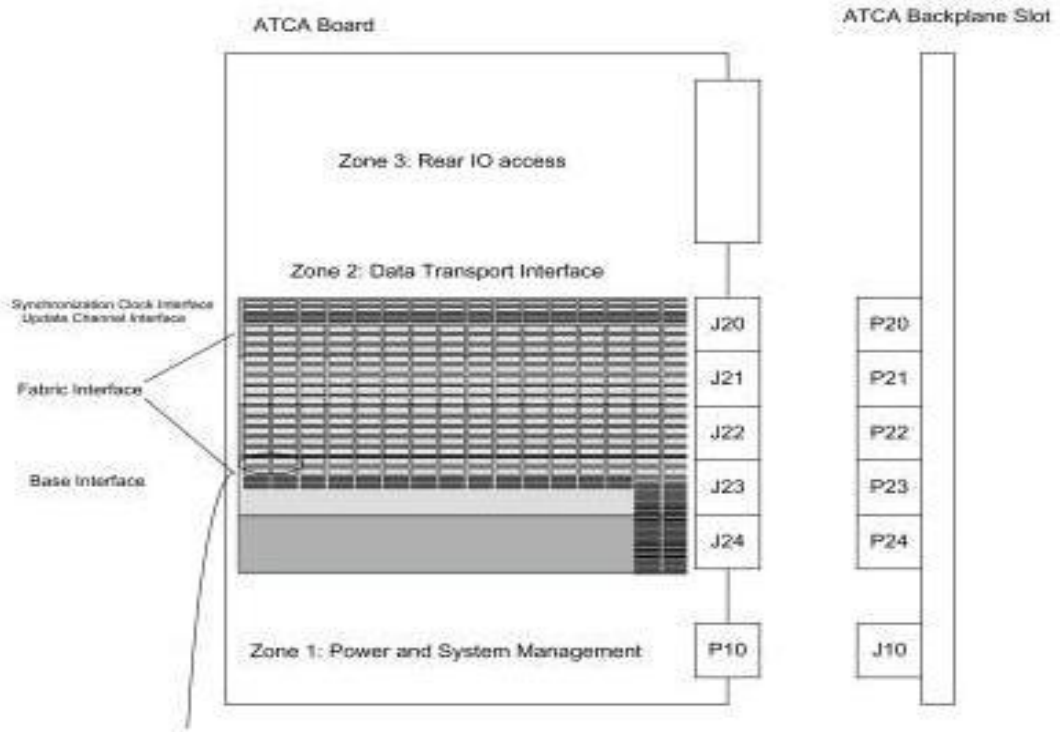


Figure 1.2: AdvancedTCA backplane and board interface

A complete rack may consume up to 3.2kW and to cope with this, the power is distributed with redundant 48V feeds through a 34-pin power connector designed by Positronic Industries. The high-speed switching fabric backplane is accessed, in Zone2, from a node board through a ZD signal connector (Erni [ERmetZD] or Tyco [Z-PACK HM-Zd]), having 10 rows of 4 differential pairs. The Zone2 is populated with high-speed connectors, capable of speeds up to 5 Gbps, and implements the required connectivity, as defined in the PICMG specifications, to support the Dual Star topology for the fabric and the base interface. The base interface uses 2 such connectors (P23 and P24) on any of the hub slots to support up to 14 base channels for the connectivity to every Node slot of the ATCA. On the other side, the dual-star fabric interface uses four connectors (P20, P21, P22, P23) in every hub slot of the ATCA chassis to ensure the communication to all nodes and to the other hub. In any node slot, the fabric interface uses always the P23 connector to receive the interconnection coming from the primary and the redundant hub. A base channel supports four differential signal pairs per link (node - hub), corresponding to a row in any of the ZD connectors. Each fabric interface channel, between any hub and any node slot, defines eight differential pairs per link suitable for enabling speeds of at least 10 Gbps. Such a fabric channel occupies 2 rows in any ZD connector.

In addition to the specifications for the Data Transport interfaces, their topology, slots interconnectivity and well-defined connectors, the AdvancedTCA standard includes specification for the cabling, shelf management, layout and trace routing etc., in order to ensure proper functionality over FR4.

1.3 Context and Motivation

The implementation of the Switch Fabric for the 10 Gigabit Ethernet switch is based on a passive backplane on which multiple 10 Gigabit Ethernet Line cards and one or two switch fabric cards are plugged. An Advanced Telecom Computing Architecture (AdvancedTCA) has been selected for the design of the T6Pro 10 Gigabit Ethernet Switch. The ATCA backplane is well specified and it has been the object of extensive and professional simulation, testing and demonstration for showing that it delivers in

practice what has been predicted in theory. It remains however a high end product, a combination of quality printed circuit and well characterized connectors, without any past history of application successes.

The AdvancedTCA standard defines the passive backplane, power sources, the chassis and also provides a set of necessary guidelines to assist with the development of the boards which will be accommodated within the chassis slots. However, implementation issues for prototype PCB designs and the quality maintenance over a production run are classical problems for any high performance product. At the logical level, the functionality of a prototype PCB can be affected by the system design and by the components functionality, which in the case of the digital circuits used in the state of the art boards, are also usually prototypes. For production boards, quality control is an issue because controlled impedance has to been maintained over a very large area and standard checks are usually only made on test coupons at the edges of the boards, where the tests are performed only in the low frequency domain. To maintain a competitive position, companies often have to introduce in their final systems prototype silicon, mounted on prototype PCBs, running high speed signals across a prototype backplane. This in turn gives rise to great difficulty in determining the source of any problems: are they in the silicon or in the backplane? The worst case scenario is one in which marginal silicon performance that escaped attention in the qualification phase results in low level errors when combined with transmission line characteristics that are just sufficiently out of the spec not to have been detected by traditional coupon tests. Tracking down those kinds of problems, and fixing them, can sometimes take as long as the original design cycle. To avoid that possible nightmare scenario, it would be preferable to characterize first the backplane and to remove it from the list of error sources.

At first, we considered specific test systems existing on the market for backplane validation. However, it turned out that such available testers are essentially checking only the connectivity and they work either at low frequencies or DC. More performing testing system are of course available, in terms of high end controllable data streams, from companies who address the prototype system characterization and test issues, but their price position excludes them from the production testing market. It was therefore decided to design and build an ***AdvancedTCA Backplane Tester***, able to address the most critical

situations possible to meet in the specified ATCA backplane. Our Backplane Tester system will be not only a simple continuity tester, but a real high-speed data traffic generator over the Dual Star fabric interface for a fully populated ATCA backplane. Some of the high-level requirements we have for the ATCA Backplane Tester system are:

- the possibility to provide a relatively *inexpensive AdvancedTCA backplane validation tool* that goes beyond connectivity testers or limited test coupons
- the possibility to demonstrate that any given AdvancedTCA backplane assembly can sustain *full bandwidth error free communication across all links* of the Dual Star fabric simultaneously
- the possibility to easily *identify fault conditions* to assist in the manufacturing process

The AdvancedTCA Backplane Tester design consists of a set of high speed Serdes devices mounted on hub and node boards that approach as closely as possible typical implementations on production modules. The Serdes of choice use the same transmission technology as the switch and node chips in a typical design and they can be programmed to simultaneously drive every interconnection link with pseudo-random bit sequences.

In the next chapters, detailed information will be provided about the hardware and the software design of the AdvancedTCA Backplane Tester. The chapter 2 presents the traffic generation over the Dual Star Fabric Interface of the AdvancedTCA backplane. The chapter 3 discuss the control issues of the Backplane Tester system and shows the final architecture and the PCB layout. The software structure is presented in chapter 4. The chapter 5 contains the conclusions.

2 Traffic generation within the AdvancedTCA Fabric Interface

2.1 Dual Star AdvancedTCA Fabric Interface

The AdvancedTCA backplane is tested in order to provide a suitable degree of confidence of its integral functionality, by providing 'real' traffic over every link in the Dual Star fabric interface. The active Backplane Tester system provides the Hub and Node boards to be plugged in an ATCA backplane to be tested during or after the manufacturing process.

Even though the AdvancedTCA standard specifies a maximum of 16 slots in the chassis, the target ATCA design has a total of 14 slots, so our active Backplane Tester system was designed in order to fully-saturate with traffic any of the fabric channels within these 14 available slots. Within the ATCA backplane, logical slot1 and 2 are dedicated for the Hub boards (primary and redundant) and they are connected to all the other slots of the Nodes across the ATCA backplane using the Dual Star fabric interface. Each Node board has two Fabric channels, one Fabric Channel connecting Hub1 and one Fabric Channel connecting Hub2. As a consequence, each Hub board has twelve Fabric Channels to connect all the Node boards plus one Fabric Channel to connect Hub boards together. The next figure shows the topology of the Dual Star fabric interface for an ATCA chassis implementing 12 Node cards and redundant hubs. Each arrow corresponds to one fabric channel.

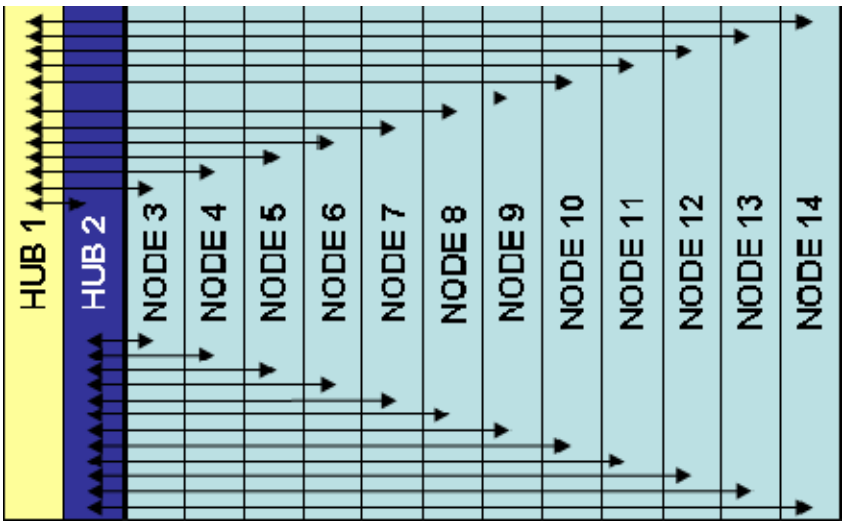


Figure 2.1: Fabric interface Dual Star topology

The PICMG® 3.0 specifications define the physical implementation of each Fabric Channel in terms of pin assignment on backplane connectors. It is based on four differential pairs per direction. In our application, each differential pair routing over the T6 Pro (10Gbps Ethernet switch) backplane operates at 3.125 GBaud, leading to a raw throughput of 12.5 GBaud per Fabric Channel. Signal encoding is 8B/10B so that the effective throughput is 10 Gbit/s per Fabric Channel. In the T6 Pro design, the each Fabric Channel has been enlarged beyond the requirement of the PICMG® 3.0. This extension is based on two extra differential pairs per direction so that the raw and effective throughputs of each Fabric Channel become respectively 18.75 GBaud and 15 Gbit/s. This extension is achieved by adding extra backplane and an extra connector (P19) in the rear I/O area Zone3 of the ATCA chassis. In this way, the fabric channel in the Backplane Tester system is composed of six differential pairs per direction. The fabric interface routing assignment is reported in Table 2.1. Each of the fabric channels is routed across the ATCA backplane from the connector P23 row1-2 on the node board to the matching connector in the Hub1, and from the same connector P23 row 3-4 on the node to the matching connector on Hub2.

			Physical slot #													
Channel #	**Connector**	**Row**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Fabric Channel 13	P20	9-10	14-1	14-2	Not implemented on backplane											
	P19	2														
Fabric Channel 12	P21	1-2	13-1	13-2												
	P19	3														
Fabric Channel 11	P21	3-4	12-1	12-2												
	P19	4														
Fabric Channel 10	P21	5-6	11-1	11-2												
	P19	5														
Fabric Channel 9	P21	7-8	10-1	10-2												
	P19	6														
Fabric Channel 8	P21	9-10	9-1	9-2												
	P19	7														
Fabric Channel 7	P22	1-2	8-1	8-2												
	P19	8														
Fabric Channel 6	P22	3-4	7-1	7-2												
	P19	9														
Fabric Channel 5	P22	5-6	6-1	6-2												
	P19	10														
Fabric Channel 4	P22	7-8	5-1	5-2												
	P20	5														
Fabric Channel 3	P22	9-10	4-1	4-2												
	P20	6														
Fabric Channel 2	P23	1-2	3-1	3-2	2-3	2-4	2-5	2-6	2-7	2-8	2-9	2-10	2-11	2-12	2-13	2-14
	P20	7														
Fabric Channel 1	P23	3-4	2-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10	1-11	1-12	1-13	1-14
	P20	8														

Table 2.1: Fabric interface routing assignment

The total of 25 Fabric Channels, 12 differential pairs each, occupies 300differential pairs on the ATCA backplane.

2.2 Traffic generation on Fabric Channels

In order to support the high-speeds for the serial data streams within the fabric channel, PICMG specifications recommend the use of 'SERDES' interfaces. There are several standard commercial SERDES devices available on the market, which are relatively cheap, small and able to transmit and receive a high-speed controllable pseudo-random bit sequence (PRBS). The basic idea is to use such devices on each of the hub and the node boards of the active Backplane Tester system, in order to fully-load with PRBS traffic any interconnection within the dual-star fabric interface.

In order to provide traffic for saturating all the Fabric Channels, Marvell Alaska 88X2040 transceiver has been selected because its built-in Bit Error Rate (BER) circuitry permits to generate and check traffic without external devices, which makes the design more

trivial. The Alaska transceiver is a fully integrated SERDES (serialization/deserialization) device that incorporates four independent lanes and delivers high-speed bi-directional point-to-point transmission at 10Gbps using eight differential pairs. The self test circuitry in each of those four lanes typically generates a 3.125Gbps PRBS data stream per direction, which is sent out via a set of four differential output pins of its eXtended Attachment Unit Interface (XAUI) interface. The 88X2040 functional block diagram is reported in Figure 2.2.

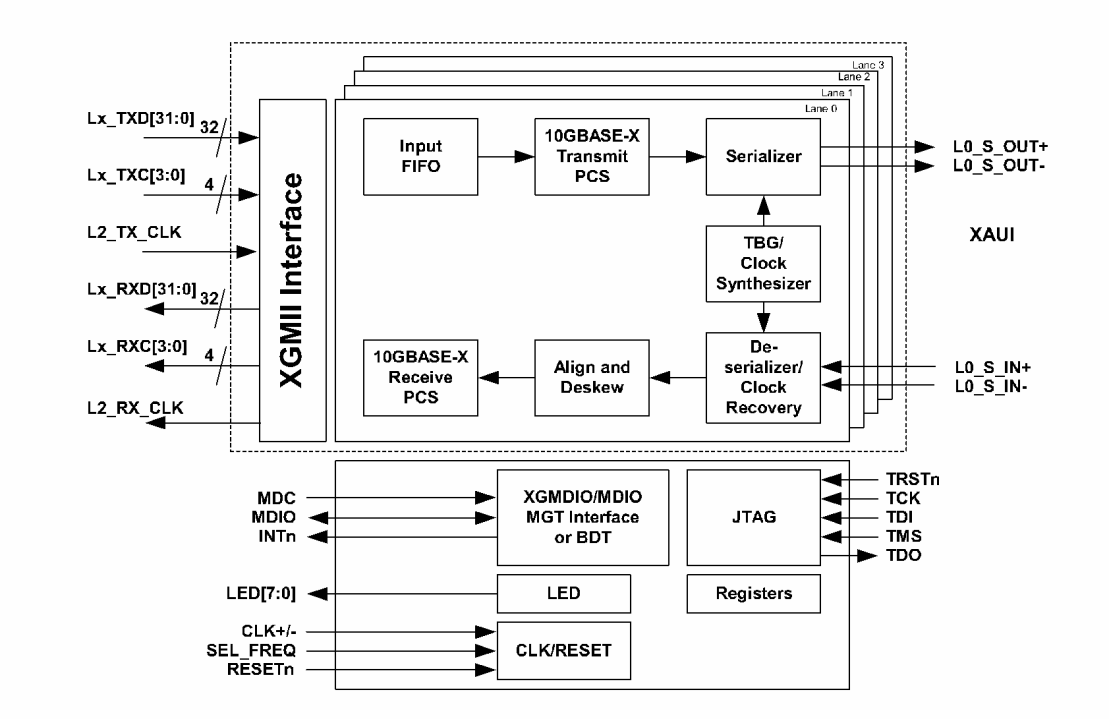


Figure 2.2: 88X2040 transceiver functional block diagram

In the Backplane Tester application, the 10 Gigabit Media Independent Interface (XGMII) is not used. Valid 8B/10B patterns are generated within the 10GBASE-X Transmit Physical Coding Sublayer (PCS), serialized and sent to the differential pairs using a high-speed interface. On the end of each differential pair, data are de-serialized, aligned, de-skewed, decoded and finally checked within the 10GBASE-X Receive PCS. Pseudo-Random Bit Sequence (PRBS) generators and checkers are also supported by the 88X2040 transceiver. At the Serdes receiver, the incoming PRBS data stream is CRC checked and five 32-bit counters are implemented – one error counter per each lane, plus a master counter register.

As the PRBS data stream on the ATCA backplane can be driven over long distances of PCB (FR4) trace, different levels of programmable pre-emphasis can be used in every lane of the transmitter serdes to compensate for the high frequency loss (e.g. skin effect). The control of the pre-emphasis is a very important feature, with significant influence over the signal integrity aspects studied during testing the AdvancedTCA backplane. The BER feature also permits to test independently each differential pair connection in order to isolate source of errors in a given Fabric Channel.

2.3 Marvell SERDES on the Backplane Tester boards

Since each Serdes device embeds four differential drivers and four differential receivers operating at 3.125 GBaud each, three of them are required to saturate the Fabric Channels of each Node board. A hub board supports a maximum of twelve fabric channels for all the Node slots and one fabric channel for the other hub; this corresponds to a need of 20 Marvell devices (in fact 19.5!) on a hub. The figure 2.3 shows the placement on the hub and the node boards of the Marvell serdes, used as PRBS traffic generators in the Dual Star fabric interface. For simplicity reasons, we highlighted only the star which emanates from the hub1; a similar star starts from hub2 towards all the slots.

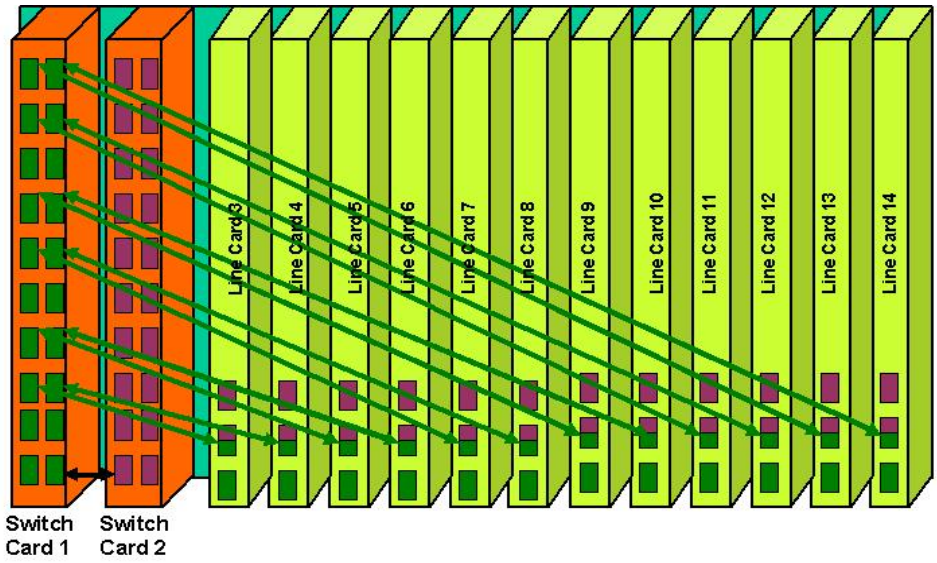


Figure 2.3: Marvell Serdes as traffic generators in the active ATCA BP Tester system

The figure above reports the implementation of the Serdes devices in the Backplane Tester system. The ‘green’ rectangles correspond to the Serdes devices assuring the communication with the primary Hub1 and the ‘purple’ color highlights the devices used for the communication with the redundant Hub2. The implementation of the Serdes devices in the Backplane Tester system is reported in Figure 2.4. In the full system, 76 devices are involved to saturate with traffic the total 300 differential pairs in the Fabric Interface: 20 for each Hub board and 3 for each Node board.

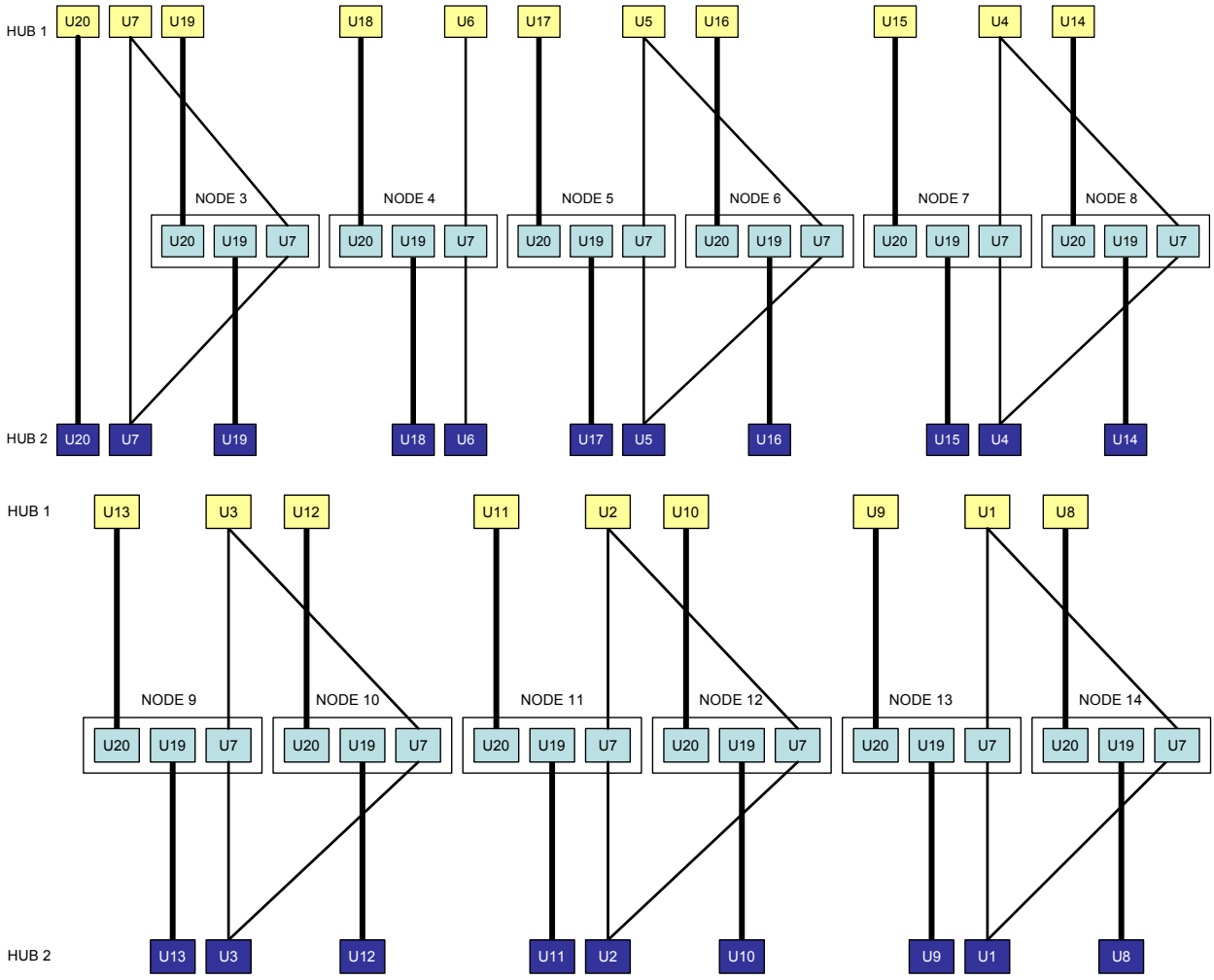


Figure 2.4: 88X2040 transceivers implementation in the Backplane Tester system

Thick lines represent four differential pairs per direction. Thin lines represent two differential pairs per direction.

3 Control issues within the AdvancedTCA Base Interface

3.1 MDIO interface

The chips handling 10Gbps functions have a standardized control path (MDIO), defined by the P802.3ae 10Gb/s standard. The Serdes chips used for generating the test traffic are no exception, and so, the MDIO control needs to be fully understood in order to implement it on the test board. The 10 Gigabit standard defines the Management Data Input/Output (MDIO) interface between a Station Management (STA) and a managed physical layer device (PHY) entity, in order to allow the access to the internal registers of the PHY device. The MDIO is a slow serial interface that supports three types of frames - address, write and read frame as shown in the figure 3.1. Within these, the address (access type) and the write frame are driven by the STA to the PHY device, synchronously with the management data clock (MDC), while the read frame is driven by the PHY device.

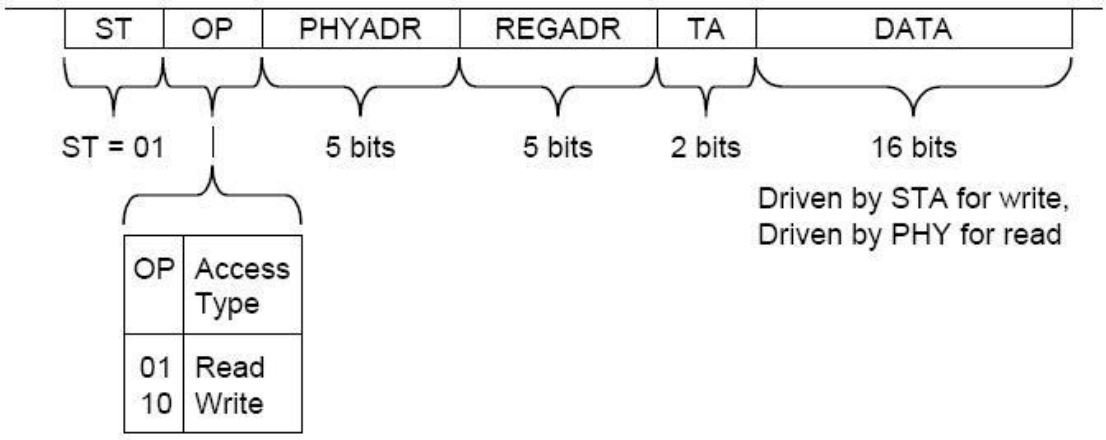


Figure 3.1: MDIO address, write and address frame as defined in 10 Gigabit standard

The MDIO control bus is a slow (0 to 10MHz) two wire bussed connection. One wire carries the common clock from the master (STA) to the bus slaves (PHY devices). The other wire is bi-directional. It carries address information during the start of the transaction and then data ‘out’ for a write or ‘in’ for a read typical MDIO frame. A typical write and read MDIO sequence is shown in the figure 3.2.

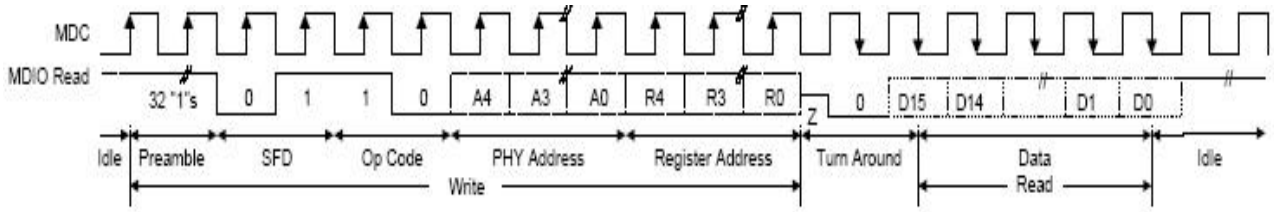


Figure 3.2: Typical MDIO sequence

Due to the 5 bits limitation in the PHYAD field, a single STA can access a maximum of 32PHYs through a single MDIO interface. For writing data to a PHY internal register, the STA first send on the MDIO bus, synchronously with the MDC, the 'MDIO address frame' which contains the physical address of the PHY and its register address to be access for a write operation. Secondly, the STA send on the MDIO bus the 'MDIO write frame' containing the data to be written at the address of the previous specified register in the PHY. Even though can be up to 32 such PHYs located on the same MDIO bus, only the device which recognizes its physical address will take the data sent by STA and found at that moment on the MDIO bus. For the MDIO read operation, STA is also first sending on the MDIO bus the 'MDIO address frame', followed by the 'MDIO read frame', containing the physical address of the PHY whose register we want to read. Only the PHY device which recognizes its address will answer by placing on the MDIO bus the content of the register previously specified in the 'MDIO address frame'.

3.2 Issues related to the control of Marvell SERDES

In the figure below we have a look inside the transmission path between two Marvel Serdes across the ATCA backplane and consider, for simplicity reasons, only a single lane in the device.

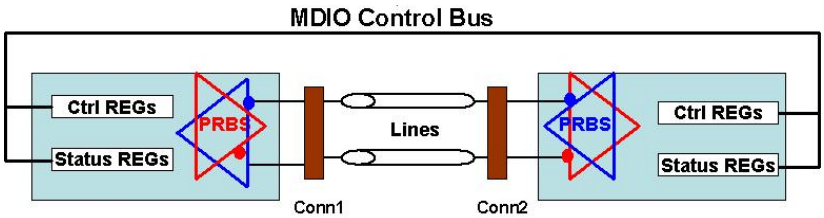


Figure 3.3: Single test channel between two serdes devices

The typical configuration phase for every Marvel Serdes includes device initialization, the pre-emphasis settings for every serdes lane and the setup of the device for driving PRBS or Jitter traffic. All this set of operations is realized by writing the internal serdes registers, as follows: Control Register, Pre-emphasis Registers for Lane0, 1, 2 and 3 and Random Sequence Control Register. Results monitoring means gathering information about the status of the device and statistics of the error counters, by reading the device Status Registers and the 32-bit error counter registers implemented for every lane.

A very important issue that will be discussed in more details in the next paragraph is the controlling of the PRBS data stream (generation and monitoring) within the ATCA Backplane Tester system.

3.3 Control processor

The STA within the MDIO interface can be any processor or microcontroller used to emulate the MDIO interface by generating the address, write and read MDIO frames synchronously with the MDC clock, in order to get the access to the Marvel serdes control and status internal registers.

It was not an easy job to choose a relatively cheap micro-controller to do this job (MDIO interface emulation) for the BP Tester system. A controller with a lot of embedded functionality was required to reduce to a minimum the hardware design of the hub and the node boards, the final test of these boards and the programming effort. One of the functionalities required is an Ethernet interface to connect to a computer from where an user to be able to specify and control the tests of the ATCA backplane. In addition, a reference platform was request to allow for software development and testing in advance of the board production.

The requirements are met by the Lantronix DSTni-LX-001. It is a 16-bit microprocessor, Intel 80186 compatible, with on-chip memory and all required communication peripherals as illustrated in the figure. It supports a real-time operating system (RTOS),

an embedded TCPIP stack and it is designed to use standard Intel/AMD x86 development tools.

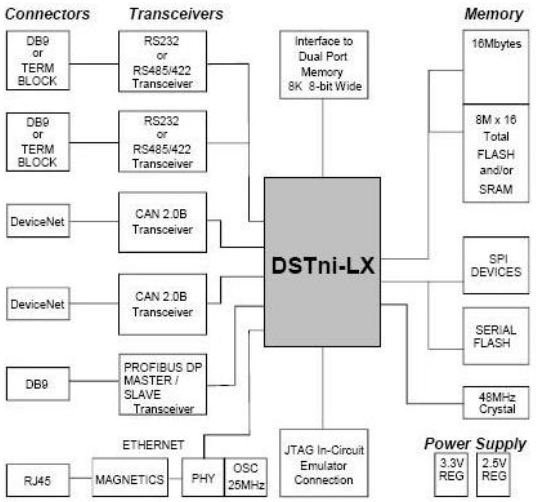


Figure 3.4: DSTNI-1 System Block Diagram

One of the attractions of the Lantronix is that is quite simple to integrate in the design:

- it already includes 256KB on-chip SRAM memory
- it has additional features, which allows to connect external RAM and non-volatile Flash memory, up to 1MB in the standard 80x86 address mode (20-bit), respectively 16MB in the extended address mode (24-bit)
- it drives a set of industrial field bus ports and it supports total of 32 programmable I/O pins
- it has an integrated Ethernet 10/100 Mbps controller with Media Independent Interface (MII), for connection to an external physical layer device (PHY)

The basic idea is to use the Lantronix controller to emulate the MDIO interface for all serdes devices in the BP Tester system, through its 32 I/O programmable pins. The control signals generated from the controller will be spread-out over the backplane using the ATCA Base Interface.

3.4 AdvancedTCA Base Interface Functionality

All PICMG 3.0 compliant backplanes support a Base Interface in a Dual Star topology. Each Base Interface Channel is composed of four differential pairs per direction and is designed to support 10/100BASE-TX or 10/100/1000 BASE-T Ethernet. Control data is fanned out over the Dual Star Base Interface in the ATCA backplane. Once the correspondence between the MDIO control signals and the BP connectors pins has been decided, the control data can be transmitted/received across the ATCA backplane using the base interface in a similar way as the Fabric Interface was used to fan-out the PRBS data streams. Even though a similar (Dual) Star emanates from the redundant Hub2, only Base Interface Channels connecting primary Hub1 are used.

Each link in the above Dual Star base interface defines a base channel, starting in the primary Hub1 and having the end point in any other slot (node or hub2) of the backplane. Each base channel has four differential pairs, in which two pairs are use to transmit the signal while the other two are used to receive that signal. When mapped into connectors, a base channel occupies an entire row in one of any of the connectors on Hub1, Hub2 or node board, as shown below.

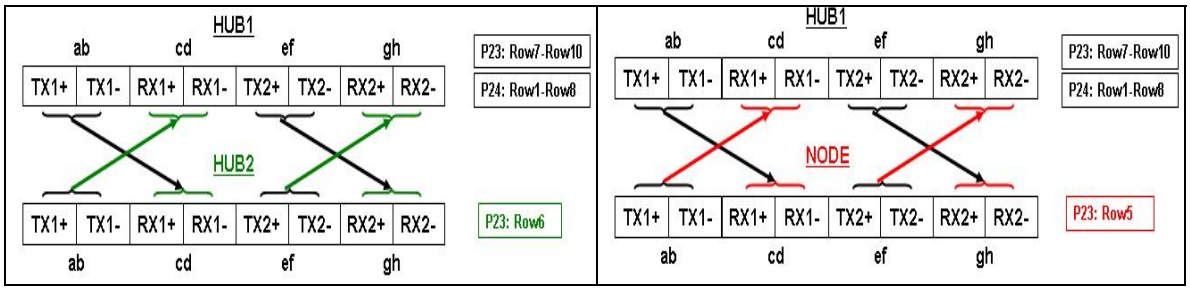


Figure 3.5: The base channel layout

On the entire ATCA backplane, there is a total of 25 Base Channels, eight differential pairs each, which occupy 200 differential pairs. Similar as we did for the Dual Star fabric interface, we studied the mapping of the base channels into the backplane connectors depending on the slot position. The Table 3.1 shows the Base Interface routing assignment.

			Physical slot #													
Channel #	**Connector**	**Row**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Base Channel 1	P23	5	Not used		1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10	1-11	1-12	1-13	1-14
Base Channel 2	P23	6	2-1	1-2	Not used											
Base Channel 3	P23	7	3-1	Not used	Not implemented on backplane											
Base Channel 4	P23	8	4-1													
Base Channel 5	P23	9	5-1													
Base Channel 5	P23	10	6-1													
Base Channel 7	P24	1	7-1													
Base Channel 8	P24	2	8-1													
Base Channel 9	P24	3	9-1													
Base Channel 10	P24	4	10-1													
Base Channel 11	P24	5	11-1													
Base Channel 12	P24	6	12-1													
Base Channel 13	P24	7	13-1													
Base Channel 14	P24	8	14-1													

Table 3.1: Base Interface routing assignment

Both P23 and P24 connectors are used on the Hub1 to support connections to all slots in the ATCA chassis. The node boards are always using only row 5 in the connector P23 for the connectivity to Hub1. The base channel starting in Hub1 is received in Hub2 at connector P23 row 6, a different point as in any node board, due to the presence of Shelf Management Controls (SHMC) which already occupies row 5 in Hub2. It is an asymmetry in the system between the Hub2 and the Node boards caused by the presence of the Shelf Management Controls (SHMC): the Base Channel from Hub1 is received in Row 6 of Hub2 and in Row 5 in any of the Node card.

For example, a set of signals sent from Hub1 connector P23 row 7, will be received on the Node card situated in the logical slot3 at the connector P23 row 5. If the same set of signals is sent from Hub1 connector P23 row 6, it will be received by the Hub2 at the connector P23 row 6.

3.5 Control signals

This microcontroller chip (Lantronix) is implemented in Hub1 which is the single point of control for the whole backplane tester system. The control structure uses the Base Interface Channels issued from Hub1 to distribute control signals to all Node boards and to Hub2.

Since 88X2040 transceivers are controlled with a Management Data Input/Output (MDIO) serial link, the microcontroller has to emulate MDIO protocol through its 32 I/O lines. Moreover, some fan-out is required because there aren't enough lines to give every 88X2040 transceiver an individual control pair. For controlling each 88X2040 transceiver, three lines are used: a MDIO bi-directional data line, a Management Data Clock (MDC) and a reset line (RESET). Each differential pair on the backplane has controlled impedance tracks designed for carrying 10/100/1000 Mbit/s Ethernet signaling so control signal integrity is not a problem. However, the control port lines need to be buffered to drive differentially the Base Interface Channels. Since the MDIO lines are bidirectional, the direction the data line buffers are driving is specified by a READIO signal provided by the microcontroller. It is then sent on the fourth signal pair.

As the AdvancedTCA standard already defines the differential connectivity across the backplane, our job was to find the best way to assign the four control lines to the differential pairs of the ATCA backplane connectors. This was quite easy as the base channel represents a row in the ZD connector with four differential pairs, and there are exactly four control signals (MDC, RESET, MDIO and READIO). So, it is enough to use one base channel per each link starting from the Hub1 to fan-out the control data towards the populated slots. Based on the base channel layout for Hub1/2 and Nodes presented Figure.3.5, we decided to use the following assignment of the MDIO control lines to the backplane connector pins.

Diff. Pair	HUB1 Connector Pair	HUB2/NODE Connector Pair
MDIO	a-b	c-d
READIO	c-d	a-b
MDC	e-f	g-h
RESET	g-h	e-f

Table 3.2: Differential Control Pair Assignments

There is further complication in that a fully-populated Backplane Tester system is controlled by only one STA (Lantronix controller) who needs to manage all 76 serdes devices and that the MDIO interface only supports five bits of addressing, which gives a maximum of 32 Serdes possible to be accesses by a single STA. In these conditions, the Lantronix controller needs to manage more than one MDIO chain. We decided for the following partitioning of the MDIO chains within the system:

- Chain A: Node boards located in slots 3:8 with 18 serdes devices
- Chain B: Node boards located in slots 9:14 with 18 serdes devices
- Chain C: Hub 2 with 20 Serdes devices
- Chain D: Hub 1 with 20 Serdes devices

The MDC clock can be a common signal for all these 4 chains, buffered at the Lantronix level. A common signal for all the boards will be as well the RESET and the READIO. However, the MDIO signals have to be differently generated and spread-out on the backplane, according to their corresponding chain and their directions (MDIO write or read). Below, we will cover in details the description of the control signals.

For each MDIO channel, one pin (within the 32 I/O Lantronix pins) is defined as an output (MDO). One output pin is enough to control all Serdes devices existing on one channel, as only the device who recognizes its physical address takes data from the MDIO bus. In conclusion, four pins within 32 I/O pins of the Lantronix will be used as MDIO output pins (defined as MDO_A, MDO_B, MDO_C and MDO_D) to support the MDIO write operation synchronously to the MDC.

- 1 LTRX output pins as MDO_A
- 1 LTRX output pins as MDO_B
- 1 LTRX output pins as MDO_C
- 1 LTRX output pins as MDO_D

Similarly, at the controller level I/O pins are defined as input (MDI) for the data available on the MDIO bus at a certain moment. They are used for the MDIO read operations. However, there is further complication for the MDIO read in the case when Serdes-es within one chain are located on several PCBs, as it maybe possible that previously write operations (MDIO address or beginning of the MDIO read frame) let some data on the bus. Therefore, we will consider at the LTRX level one input pin (MDI) per each Node slot in the system, as follows:

- 6 LTRX input pins as MDI_A from 3:8
- 6 LTRX input pins as MDI_B from 9:14
- 1 LTRX input pin as MDI_C
- 1 LTRX output pins as MDO_D

Both - write and read operations - can be managed by using a single pin at the Lantronix level for generating the MDC signal, which will be buffered and then spread out to all the slots. Similarly, one pin defined as output is used for RESET and another one for READIO.

- 1 LTRX output pins as MDC
- 1 LTRX output pins as RESET
- 1 LTRX output pins as READIO

Finally, the Figure 3.6 show the use of the I/O pins in the Lantronix controller in order to achieve the distribution of all control signals (within four MDIO chains) on a fully-populated AdvancedTCA chassis. In addition to these, fan-out and single to differential buffering is required to distribute the generated signals from Lantronix on Hub1 towards all other slots in the chassis.

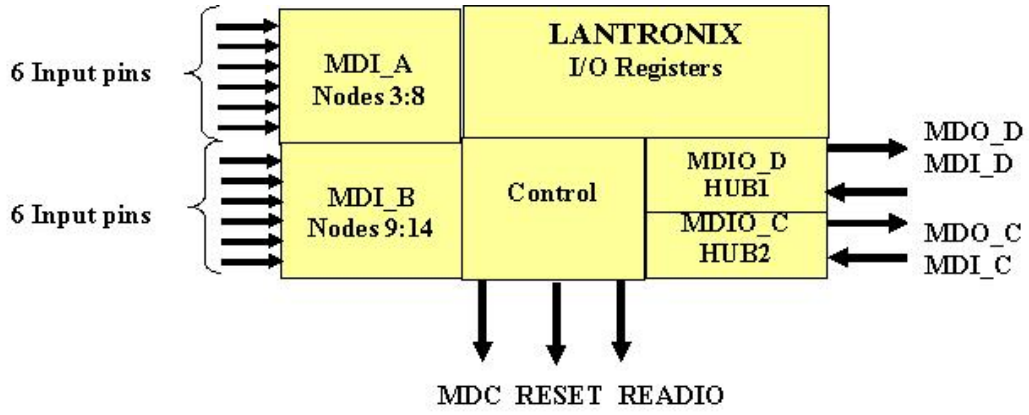


Figure 3.6: I/O pins in the Lantronix used to generate the control signals

3.6 Control architecture

The control architecture has been defined so that a single PCB can be used either for Node boards either for Hub boards. Hub boards are fully populated with 88X2040 transceivers whereas Node boards are partially populated with only three of them to handle two Fabric Channels. The control architecture is reported in Figure 3.7. The control signals MDIO, MDC, READIO and RESET are sent from Hub1 and received on Base Channel 1 of Node boards (P23 connector, row 5) and on Base Channel 2 of Hub2 (P23 connector, row 6). Due to this asymmetry in the system caused by the presence of the Shelf Management Controls (SHMC), the Base Channel from Hub1 to Hub2 starts in row 6 of Hub1 and is received in Row 6 of Hub2. This explains the array of jumpers shown in Figure 3.7, which are used to achieve a different functionality for each board (e.g. board used as Hub1 or as Hub2 or as a Node board).

Let us just take the RESET signal as an example to explain the control architecture. When the board is placed in the Hub 1 position, RESET is generated and fanned out to all other slots. The fan-out to Hub2 however uses pair Row 6 g-h. Now, when this design is placed in the Hub2 position, Row 6 g-h is used to receive the clock MDC so there is a first level of jumpers to cover this double usage of pair Row 6 g-h. Now when the board is placed in any of the Node slots, MDC is received from Row 5 g-h. There is therefore a second layer of jumpers to distinguish between double sources of this control signal. Note

that when the board is being used in the Hub1 position, there are no jumpers installed at all on this second layer since all the control signals to the on-board 88X2040 transceiver are generated on the board. When being used as Hub2 or Node card, the microcontroller and MDIO buffers are not mounted so there is no driving conflict on the control lines.

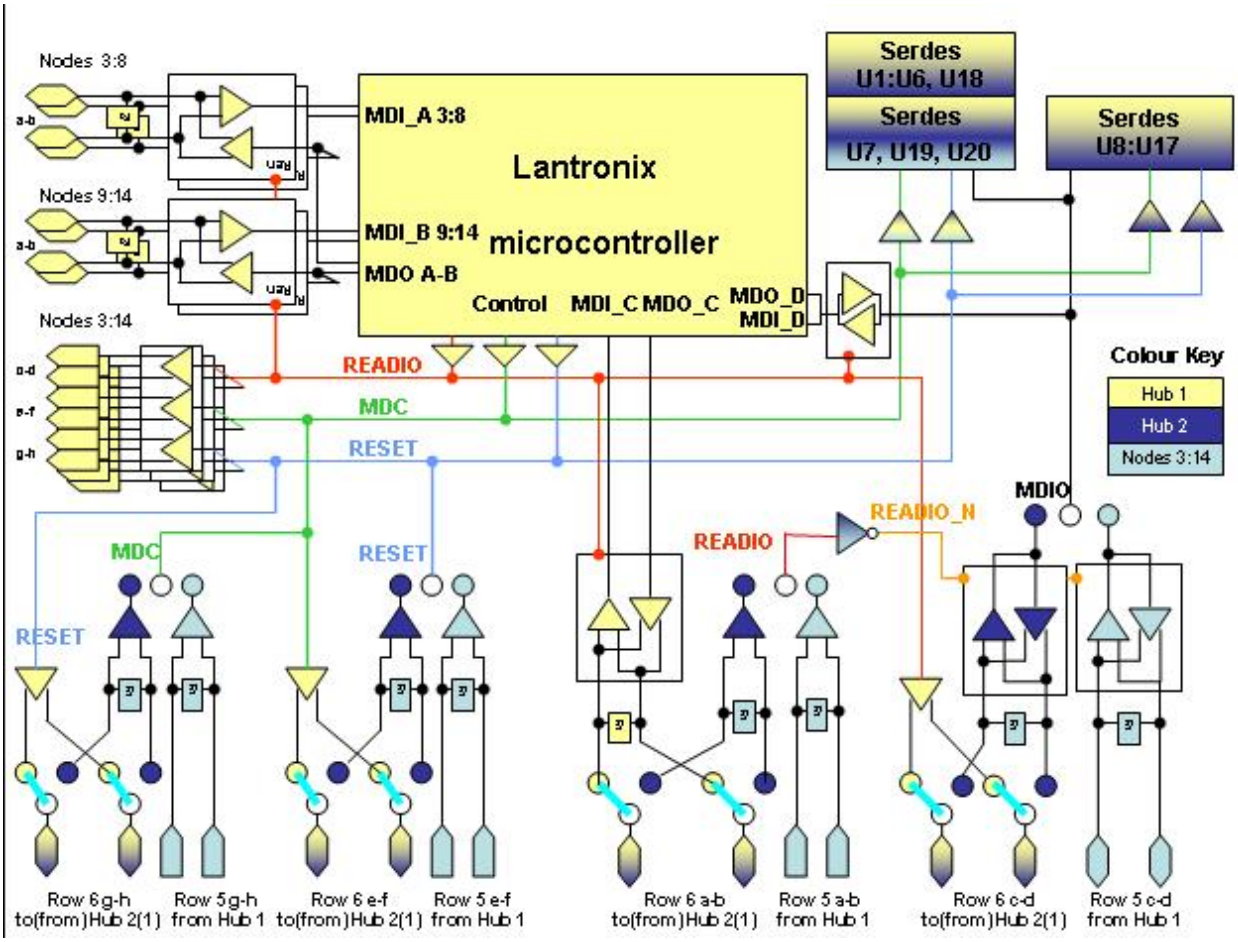


Figure 3.7: Control architecture

Taking the overall design now we cover the details on a block by block basis.

The MDIO line is split into In and Out at the level of the controller because of a pin limitation. This is to avoid the problem of individually enabling every input buffer. By splitting the Input and Output, we can bus together the outputs according to which chain they are on and we can receive all the inputs together and choose the one of interest with software. The one exception to this is the internal MDIO to the Hub1 88X2040 transceiver. The data line is buffered since the Controller pin does not have much drive capacity and there are twenty loads to be served.

The twenty 88X2040 transceivers on each Hub board are split into two groups presenting 10 loads each to the buffered control signals RESET and MDC.

READIO, MDC and RESET are also buffered at the level of the microcontroller chip since there are twelve line drivers to serve as well as the internal loads. Double buffering on the data line is not required because the line has a long time to settle in between clocks, the delay can even be extended by software if needed. MDC and RESET lines were buffered however to ensure that the edges are rapid and not likely to create false triggering inside the 88X2040 transceiver logic. Serial termination is included to avoid overshoot. Note that in the case of a Node card, there are only three 88X2040 transceiver ports present. For Hub2 or Node cards, all the logic shown in yellow is absent, having not been mounted. In this case, there must be some jumpers to select the function as being either a Hub2 or a Node card and the appropriate settings for rows 5 and 6 must also be selected.

3.6 Controller implementation

As shown in Figure 3.8, the microcontroller is implemented with two key interfaces to access a remote PC:

- a RS-232 serial link used to load compiled server application from the remote PC to the micro-controller
- an Ethernet interface for communications between the server application operating on the micro-controller and the client application operating on a remote PC.

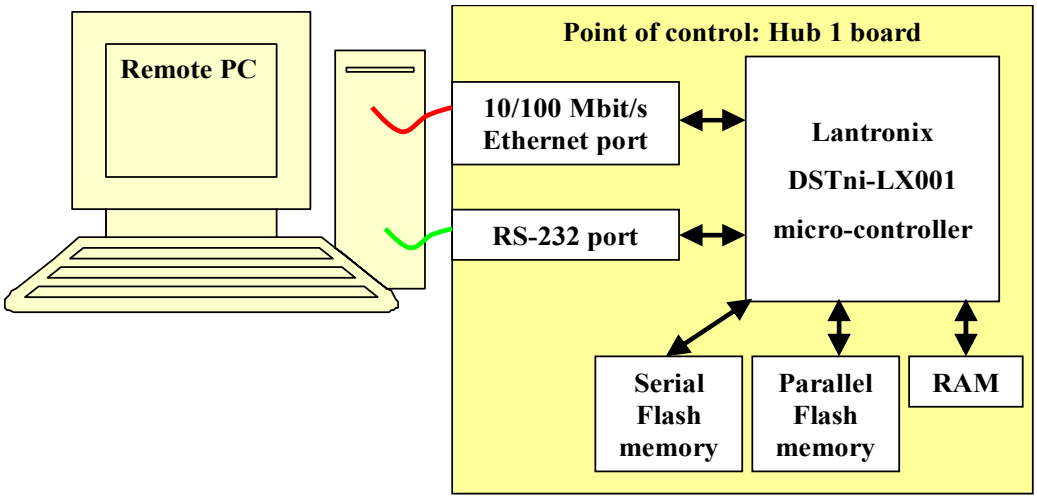


Figure 3.8: Microcontroller implementation

The microcontroller is associated with on-board parallel and serial flash memories, respectively 1M x 16-bit and 264 KByte, for storing the server application. It operated with 256KByte on-chip and 1MByte on-board RAM.

3.7 Testboards architecture

The hardware design goal of the Backplane Tester system was to finally have a single printed circuit board that can be used in every slot position: either as a hub or as a node board. This was not an original goal, but seeing that one was almost the subset of the other gave the incentive to try.

The single PCB requirement is achieved by having a single hardware design and mounting only the digital circuits required by the final board functionality during the manufacturing process. The figure below shows how there can be achieved three different functionalities with the same PCB, by only de-populating the Node and the Hub2 board.

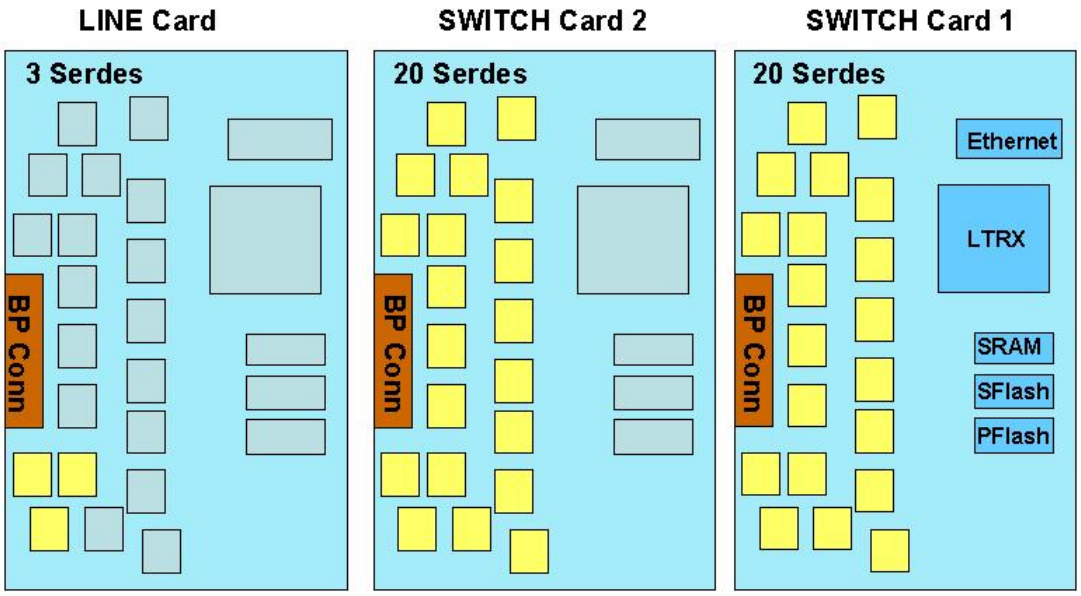


Figure 3.9: Test board architecture

Both hub boards carry the Serdes for all the links in the Dual Star fabric interface, but only the primary hub board (in logical slot 1) contains the Lantronix controller implementation and the control logic. In addition, the Node boards just have the Serdes devices required for the links coming from the two hub cards. The implementation of the Serdes devices in the Backplane Tester system is reported in Figure 2.4. In the full system, 76 devices are involved to saturate with traffic the total 300 differential pairs in the Fabric Interface: 20 for each Hub board and 3 for each Node board.

3.8 Testboards PCB design

The Backplane Tester boards have the dimensions 322.5 mm x 280 mm. The 14-layers stack-up is reported in Figure 3.10.

Layer	Name	Copper thickness (mils)	Dielectric thickness (mils)
1 (coated microstrip)	**Sig-1**	2.1[(1)]	
Pre-preg			7.5
2 (ref. plane)	**P1V5/GND**	1.4	
Core			8
3 (ref. plane)	**P1V8/P2V5/GND**	1.4	
Pre-preg			7.5
4 (ref. plane)	**P3V3/GND**	1.4	
Core			8
5 (ref. plane)	**GND**	1.4	
Pre-preg			8.5
6 (stripline)	**Sig-2**	0.7	
Core			8
7 (ref. plane)	**GND**	1.4	
Pre-preg			8.5
8 (stripline)	**Sig-3**	0.7	
Core			8
9 (ref. plane)	**GND**	1.4	
Pre-preg			8.5
10 (stripline)	**Sig-4**	0.7	
Core			8
11 (ref. plane)	**GND**	1.4	
Pre-preg			8.5
12 (stripline)	**Sig-5**	0.7	
Core			8
13 (ref. plane)	**GND**	1.4	
Pre-preg			7.5
14 (coated microstrip)	**Sig-6**	2.1[(1)]	
Total thickness (including plating) +/- 10%		**3.1 mm**	

Figure 3.10: Backplane tester PCB stack-up

[(1)] 1.5 OZ Cu Finish.

The key design challenge in this PCB design is the routing strategy for the differential pairs related to Fabric Interface Channels and the routing of the differential pairs related to Base Interface Channels. Among the six signal layers reported in Figure 3.11, four of them are dedicated for routing those signals. One of those layers is displayed on Figure 3.11.

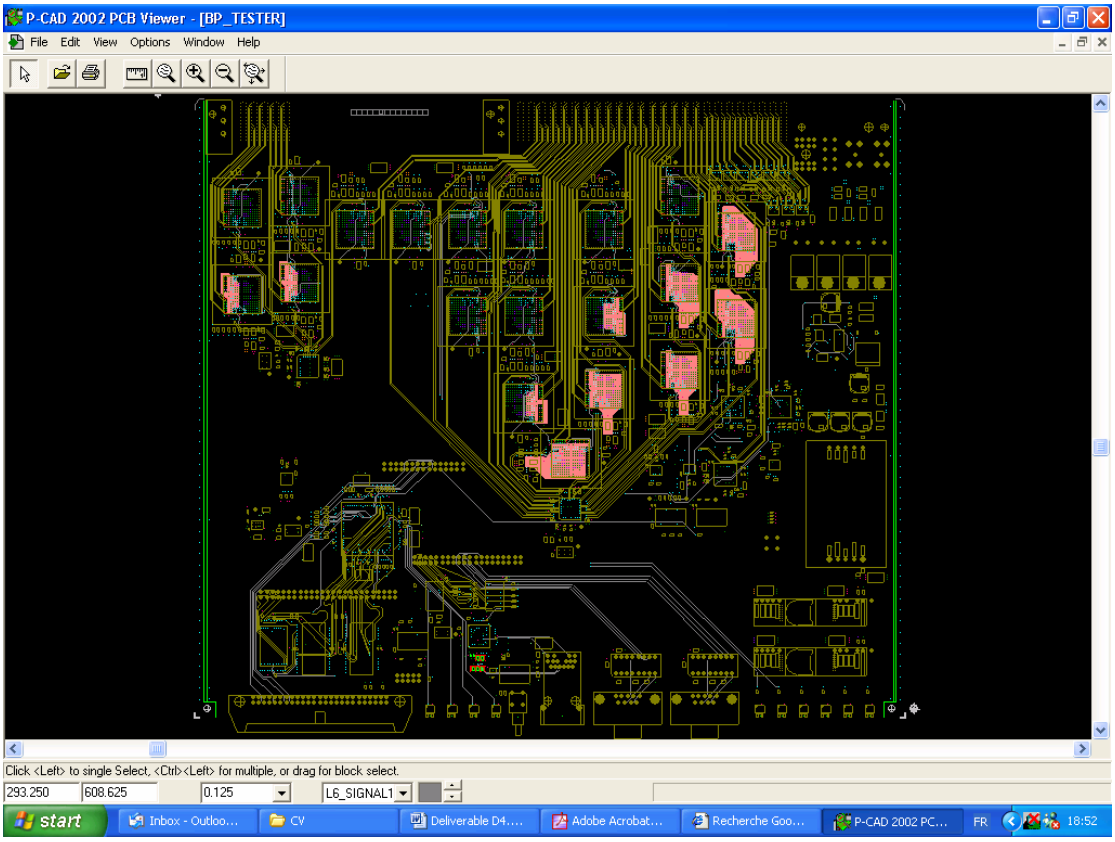


Figure 3.11: Routing on layer Sig-2.

4 Control Software

4.1 Software architecture model

The most commonly used paradigm in constructing distributed applications is the client-server model. In this scheme, the client application request services from the server process, which normally listens at a well known address (port) for service requests. When a connection is requested by a client to the server's address, this one services it, by performing whatever appropriate actions the client requests. The second task performed by the server is to supply information regarding the status of its host. The client and server require a well known set of conventions, known as a ‘protocol’, which must be implemented at both ends of a connection.

This client-server model suits quite well our Backplane Tester application and we decided to develop the software based on it. From a remote PC, which is connected to the Backplane Tester system, the user can completely define the test to be performed over the backplane. The controller task is to set-up the traffic across the backplane according to the user-defined test, and to maintain the statistics to the remote PC. It comes then naturally the distribution of the client tasks for the remote PC and the server task to the controller, as shown in Figure 4.1. The connection between the client and the server, through which the client requests and the server statistics are sent, is made via TCP/IP over a 100Mbps Ethernet link.

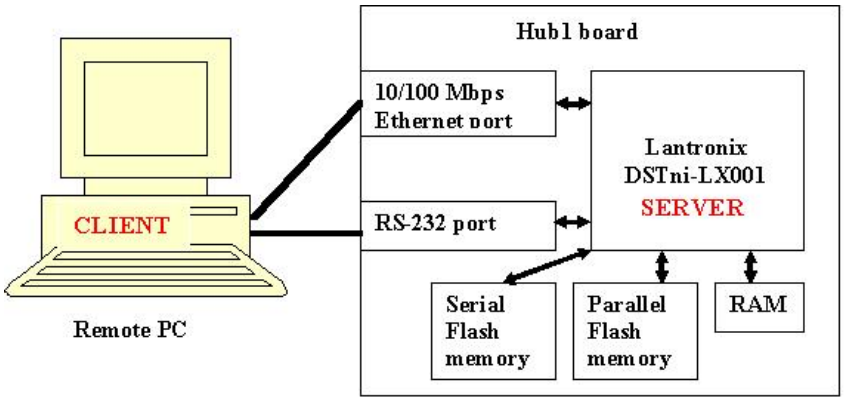


Figure 4.1: Client-Server model for the Backplane Tester system

In this chapter we will consider some of the problems in developing client and server applications and look more closely how the user interacts with the tester system for defining and performing a test of the ATCA backplane.

4.2 Software architecture design issues

The user interface for the Backplane Tester is represented by an Excel spreadsheet (see paragraph 4.3), to which the user interacts directly on the remote PC, by defining the test to be performed over the backplane.

A first design issue is related to the place where the conversion of the Excel data into a valid request for the server can be done. There are two possibilities:

- to download the Excel table over an FTP connection on the controller and to convert the data there, at the server level;
- to convert the Excel table on the remote PC into a client request and to sent it to the controller.

We have decided to keep the software on the controller (the server!) simpler as possible, in order to debug it easier in case of failure. In addition, we would like to have a direct access to the conversion result on the remote PC. For these reasons, the Excel data are converted locally, on the remote PC into a valid request to be sent to the server.

A second design issue is about the format in which the user specifications given in Excel are transformed. It is known that in order to program a typical PHY device (serdes) to perform certain operations, several serdes registers has to be written with certain values, which corresponds to a succession of WRITEMDIO commands. On the other hand, a certain succession of WRITEMDIO commands corresponds to a certain task, for which a macro can be created (e.g. the initialization of a Marvell serdes device needs 3 registers to be written –> 3 WRITEMDIO commands). Similar, reading the error counters registers of a serdes devices contains several (8) READMDIO commands, which can be included into a macro. Therefore, we have two possibilities of conversion:

- Convert Excel table into a succession of command lines (macros);
- Convert Excel table into a succession of WRITEMDIO/READMDIO commands.

We chose to convert the Excel spreadsheet into a sequence of macros. In order to achieve that, the Excel data is converted into three configuration files, containing a succession of macros for each defined test. This conversion in pure Excel is impossible. However, it can be done by programming or scripting. C programming, here is heavy and difficult to manage, but the Visual Basic scripting has the advantage of being managed without compilation. We used Visual Basic scripting for converting the Excel spreadsheet into the sequence of macros contained in the configuration files.

A third design issue, very close related to the second one, is about how should look the client request sent to the server. As the Excel data is transformed into a sequence of macros, we have two possibilities:

- Send the client request as a succession of macros;
- Send the client request as a succession of WRITEMDIO/READMDIO commands corresponding to a certain macro.

If we chose to send the request in terms of macros, this would mean that the server has to expand each macro in its corresponding sequence of WRITE/READMDIO commands. As mentioned above, we want to keep the server application simpler as possible. Also, we prefer to expand the macro into a sequence of corresponding WRITE/READMDIO commands at the client level, on the remote PC. Therefore, the configuration files are

considered as input to the client application, which expands each macro into its corresponding sequence of WRITE/READMDIO commands and sends it as a request to the server.

A fourth design issue is related to the communication protocol between the client and the server. The client request can be send to the server:

- MDIO command-by-command;
- One single packet, which encapsulates all MDIO commands corresponding to a macro.

The acknowledge from the server can be received:

- for each MDIO command separately;
- one acknowledge packet for all the MDIO commands corresponding to a macro.

Bandwidth economy reasons determine the choice of the second possibility. Only one packet will be sent in request to the server for each macro and in replay to this packet, an acknowledge packet will be received back to the client.

Based on these four architectural decisions, the conversion of the user request (paragraph 4.3), the client (paragraph 4.4) and the server (paragraph 4.5) applications have been created and their descriptions is given in the next paragraphs.

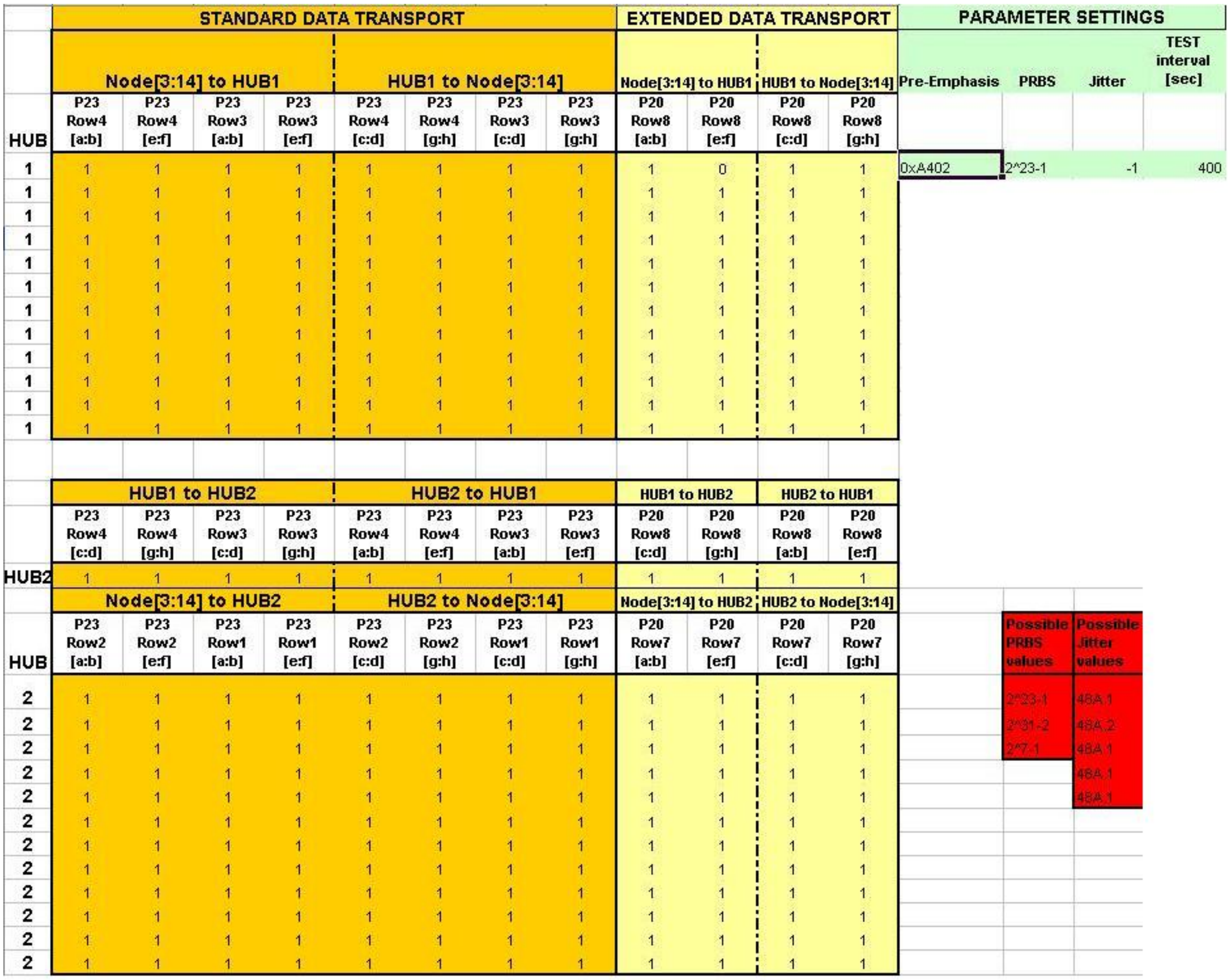
4.3 User interface

The user interacts directly with the remote PC where the backplane test is decided via an Excel spreadsheet, which represents the user interface. In this Excel spreadsheet, the user defines for any given test which set of total available differential pairs to be activated on the backplane, the type of bit sequence (PRBS or Jitter) to be sent on these pairs, the level of pre-emphasis on the drivers and the duration of the test. The spreadsheet contains 3 different sheets, but for defining a test the user interacts only by modifying the areas in the sheet1 and then by running a macro in Visual Basic over the whole spreadsheet. All

the rest of the operations, including the processing done in Excel for filling-in the second sheet2 and third sheet3, and Visual Basic coding, are completely transparent for the user. As result of their execution three different configuration files are automatically generated and they will be further used as inputs by the client application.

Now, we will describe the sheet1 areas of the Excel spreadsheet (ATCA_BP_Tester.xls), to which the user is directly interacting and which can be seen in the figure 4.2. There are three areas highlighted with different colors (dark yellow, light yellow and light green area) where the user defines a test. Every entry in the dark and light yellow shaded area can be modified to fully define the active differential pairs for testing over the backplane. The dark yellow, represented by the columns C to J, defines the standard[1] differential pair connections and the light yellow area (columns K to N) defines the extended[2] differential pair links within the Dual Star Fabric Interface between a node and a hub board. In the light green zone, the user can set the pre-emphasis level at the transmitter, the type of the traffic to be run (PRBS or Jitter), and the time (in seconds) to run the test. In the dark red area below, there are given the existent possibilities for PRBS or Jitter traffic to choose between.

[1] Four differential pairs per direction within a Fabric Channel
[2] Two extra differential pairs per direction in addition to the standard Fabric Channel



HUB	STANDARD DATA TRANSPORT: Node[3:14] to HUB1				HUB1 to Node[3:14]				EXTENDED DATA TRANSPORT: Node[3:14] to HUB1		HUB1 to Node[3:14]	
	P23 Row4 [a:b]	P23 Row4 [e:f]	P23 Row3 [a:b]	P23 Row3 [e:f]	P23 Row4 [c:d]	P23 Row4 [g:h]	P23 Row3 [c:d]	P23 Row3 [g:h]	P20 Row8 [a:b]	P20 Row8 [e:f]	P20 Row8 [c:d]	P20 Row8 [g:h]
1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1

PARAMETER SETTINGS: Pre-Emphasis	PRBS	Jitter	TEST interval [sec]
0xA402	2^23-1	-1	400

	HUB1 to HUB2				HUB2 to HUB1				HUB1 to HUB2		HUB2 to HUB1	
	P23 Row4 [c:d]	P23 Row4 [g:h]	P23 Row3 [c:d]	P23 Row3 [g:h]	P23 Row4 [a:b]	P23 Row4 [e:f]	P23 Row3 [a:b]	P23 Row3 [e:f]	P20 Row8 [c:d]	P20 Row8 [g:h]	P20 Row8 [a:b]	P20 Row8 [e:f]
HUB2	1	1	1	1	1	1	1	1	1	1	1	1

HUB	Node[3:14] to HUB2				HUB2 to Node[3:14]				Node[3:14] to HUB2		HUB2 to Node[3:14]	
	P23 Row2 [a:b]	P23 Row2 [e:f]	P23 Row1 [a:b]	P23 Row1 [e:f]	P23 Row2 [c:d]	P23 Row2 [g:h]	P23 Row1 [c:d]	P23 Row1 [g:h]	P20 Row7 [a:b]	P20 Row7 [e:f]	P20 Row7 [c:d]	P20 Row7 [g:h]
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1

Possible PRBS values	Possible Jitter values
2^23-1	48A.1
2^31-2	48A.2
2^7-1	48A.1
	48A.1
	48A.1

Figure 4.2: View of the user interface

In the second sheet2 of the same spreadsheet, the developer already defined the standard and the extended connectivity between the Hubs and the Node boards on the backplane for all the existent devices. For the standard connectivity, there is implied one device in each of the hubs for serving the connectivity to a corresponding device existent on every node board. In the case of extended links, one single device on the node card serves both hubs and on each hub one device handles extended links from two node cards. The connectivity between a chosen Node and a Hub is defined in terms of MDIO bus (A, B, C, and D) that is controlled by the Lantronix, and the device number of the serdes on that MDIO bus. For example, the standard link between Node3 and Hub1 is controlled by the Serdes#0 on MDIO bus A (A0 – device U20) in Node3 and by the Serdes#6 on MDIO bus D (D6 – device U19) in Hub1. Equally, the extended links from Node3 run from

Serdes#12 MDIO bus A (A12 – device U7) to Serdes#12 on the MDIO bus D (D12 – device U7) in the Hub1. Note that Serdes D12 also handles the extended links from C12 (device U7) in Hub2. MDIO bus A serves nodes 3 to 8, B serves nodes 9 to 14, C serves Hub2 and D serves Hub1.The mapping is derived from the diagrams showing the serdes device attribution in the Figure 2.5.

The third sheet3 of the spreadsheet is automatically filled-in, in a totally transparent mode from the user point of view. The test information are extracted and manipulated in the Excel format at the developer level from the information supplied by the user in the sheet1 and the defined connectivity on the backplane in the sheet2. At the Excel level, the spreasheet3 defined the test by the mean of each lane, on which MDIO bus, in each device to be transmitting, receiving or inactive. So for example, if at least one lane is active in the communication between the Node3 and Hub1, A0-D6 (device U20 on the Node – device U19 on the Hub1) are automatically considered as drivers/receivers for the standard and A12-D12 (U7-U7) for the extended interface. The status of each lane as TX/RX, active/inactive is automatically determined from the user test setup in the sheet1. For example, in the current test between the Node3 and the Hub1 (see Figure 1) all the TX and RX standard differential pairs, the RX pairs and the first TX extended differential pair are active, while the second extended TX pair is inactive/disabled.

Once the user enters the desired test setup in the sheet1 of the Excel spreadsheet (ATCA_BP_Tester.xls), the filling-in of the sheet3 with the complete test definition is automatically performed. The user just has to run on the Excel spreadsheet a Visual Basic macro, which automatically extracts three configuration text files (setup.txt, reset_counters.txt and read.txt). These files will be further used as input to the Client application for defining the current test over the ATCA backplane (setup.txt) and for reading the error statistics for the active links over the ATCA backplane (read.txt).

Figure 4.3: User interface overview

The data processing in Excel and in Visual Basic is simpler than C programming and we manage to obtain without compilation, though a simple and automatic process, the configuration files used by the client application.

4.4 Client application

The location of the client in our Backplane Tester application has been chosen on the remote PC, to which the user interacts directly by defining the test via the Excel user interface above presented. One main task of the client is to take the user specifications for the test, to transform and send them at the server, according to a well-defined client-server communication protocol. The second task of the client is to get statistics back from the server (based on the same communication protocol) and to report these statistics in real-time to the user.

For achieving the first task, the client application takes as input parameters the configuration files, which have been created based on the user specifications in Excel. There are 3 different configuration text files: <setup.txt>, <reset_counters.txt> and <read.txt>, used for achieving specific tasks in the current backplane test. For example, <setup.txt> is used to configure each individual lane, defined as active in the serdes device, in order to make it able to transmit and/or receive traffic. The <reset_counters.txt> configuration file is used to perform the reset of the error counters of each active serdes at an interval of 100seconds, as it is recommended in the serdes datasheet. The third configuration file, <read.txt>, is used every 60seconds when the client requests the server to read the status and the values of the error counters for each active serdes device in the test. Each of these three configuration files contains a series of command line sequences (macros), shown in the figure below.

Config. file	**Macro Name**
<setup.txt>	reset_global initialization & initialization_ext

	led_control Pre_Emphasis_Setup Lanes_Dezactivation & Lanes_Dezactivation_ext Set_Traffic & Set_Traffic_ext
<reset_counters.txt>	Reset_counters
<read.txt>	Read_Status Read

Table 4.1: List of available macros

The server running on the Lantronix controller only can execute commands, which allow a low level WRITE or READ access to the MDIO bus. Therefore the client application has to interpret and 'expand' each macro into an equivalent sequence of WRITE/READ MDIO commands and send this sequence to the server. All the macros in the <setup.txt> and <reset_counter.txt> configuration files are transformed in an equivalent sequence of WRITE MDIO commands. The macros in the <read.txt> file are converted into several READ MDIO instructions. For a more detailed description of each macro, see Appendix1, 2 and 3.

Once achieved the expansion of a macro, the next step is to encapsulate its corresponding MDIO commands into a packet, and send it as a request to the server application, via the TCP/IP connection. In the communication protocol we defined between the client and the server, a packet request has the following format:

MACRO_Name\\MDIOinstr[1]\\MDIOinstr[2]\\…\\MDIOinstr[n]

Figure 4.4: Format of the client request to the server

Note1: MDIOinstr[i] can be either a WRITE MDIO command (for any of the macros in the <setup.txt> or <reset_counters.txt>), or a READ MDIO instruction (for the macros in the <read.txt> configuration file).

After the client request has been sent, it is the server application task to execute command by command and to return back an acknowledge packet. The ACK packet contains the execution results of each command from the previously sent request together with the client request itself. The ACK package format as defined by the client-server communication protocol is the following:

MACRO_Name\\ACK[1]:…:ACK[n]\\MDIOinstr[1]\\…\\MDIOinstr[n]

Figure 4.5: ACK received from the server

Note2: ACK[i] contains the result of the MDIOinstr[i] execution. If it is a WRITE command, then a positive +1 value in the ACK[i] confirms that the MDIOintrs[i] has been executed successfully in the server, while a negative -1 value shows a failure in the execution at the server level. In the case of a READ MDIO instruction, the ACK[i] value contains the read value itself on the MDIO bus.

The received ACK packet is analyzed at the client to determine if all the WRITE operations inside one packet have been successfully executed on the server, or to get (READ) inquired information about serdes devices (status and/or error counter values). On the remote PC, the client displays in real-time on the terminal console all received information (ACK packets) from server and the results obtained after the ACK processing. For allowing later access to the user, all the displayed information is also stored in corresponding LOG files (<setup_LOG.txt>, <reset_counters_LOG.txt> and <read_LOG.txt>). A brief overview of the client-server inter-operability is given in the figure 4.6.

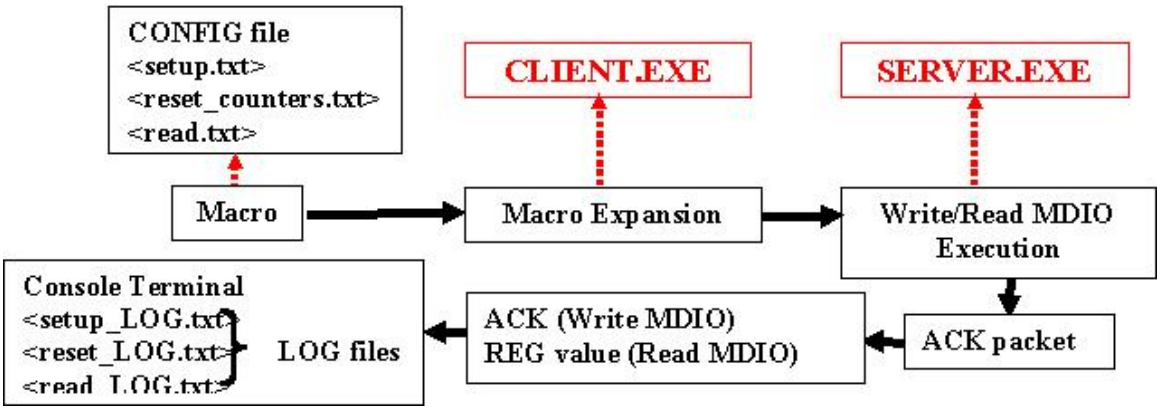


Figure 4.6: Client-server inter-operability overview

An important high-level task performed by the client application is the periodical reading and counters resetting for all active serdes devices in the current test. The client sends each minute reading request to the server and gets back from this one the information about the serdes status and their error counters. Also, the client sends requests for resetting the counters each 100seconds, interval recommended by the serdes datasheet. The current test is stopped either when the user-defined duration of the test is achieved or

when the user press <Ctrl^C> buttons. At the end of the test, the LOG files contains the results of the setup execution, the latest counters reset and the results of the latest reading operation. For any given test, the read values of the counters are showing accumulated errors from the beginning of the test.

4.5 Server application

The server is running on the Lantronix controller. The application is downloaded via the RS-232 serial port from the remote PC and it is stored into the RAM memory, the serial flash or the parallel flash on the Hub1 board.

The Lantronix processor controls the traffic on the backplane, monitoring the state registers of the serdes devices, through four MDIO control chains. The control, by the mean of write and/or read access to the internal registers, is emulated through the I/O pins of the Lantronix, by implementing at the server level the commands for the MDIO write and read operations. The MDIO commands permit a low level access to the MDIO bus, by allowing to write, respectively read data to/from the bus in a synchronously to a locally generated clock. The format of the WRITE and READ MDIO commands send from the client and received by the server can be seen in the figure 4.7.

write_mdio:MDIO_BUS:PHY_ADR:REG_ADR:REG_DATA
read_mdio:MDIO_BUS:PHY_ADR:REG_ADR

Figure 4.7: Format of the WRITE MDIO and READ MDIO commands

Note3: MDIO_BUS can any value between A, B, C or D. The values for the device physical address (PHY_ADR), the register address (REG_ADR), respectively the register data (REG_DATA) are given in hexadecimal. PHY_ADR can have any value between 0 and 19 for the Hubs (MDIO_BUS C or D) and values between 0 and 2 for the Nodes (MDIO_BUS A or B). The REG_ADR can have any value available as a register address in the serdes datasheet.

The request from the client is received as MDIO commands, encapsulated and formatted according to the client-server communication protocol into a packet as presented in figure 4.7. According to the client-server protocol, the server unpacks the client request into

low-level MDIO commands, which then it solves and gets an ACK for each of them individually. For example, in the case of a packet containing MDIO write commands, every instruction in the packet is acknowledged with a positive (+1) in the case of success, or a negative (-1) ACK in the case of failure. In the case when the package contains a sequence of MDIO READ instructions, every operation performed at the server returns the value read on the MDIO bus. The ACK package sent to the client is formatted according to the client-server protocol and contains an individual value (either ACK or read value) for each executed MDIO command in the received packet.

4.6 Client- Server validation

For evaluating the control software for the Backplane Tester system before the actual Hub and Node boards are coming from the manufacturing process, we used the evaluation board of the Lantronix DSTni-LX microcontroller. The DSTni-LX evaluation platform is the equivalent of the primary Hub card, having the Lantronix controller on it, adjacent memory (RAM, serial flash and parallel flash) and Ethernet interface. The evaluation board implements some other interfaces available in the controller, such as CAN, Profibus etc., which are not of interest for us. The next figure 4.8 shows a view of this evaluation platform.

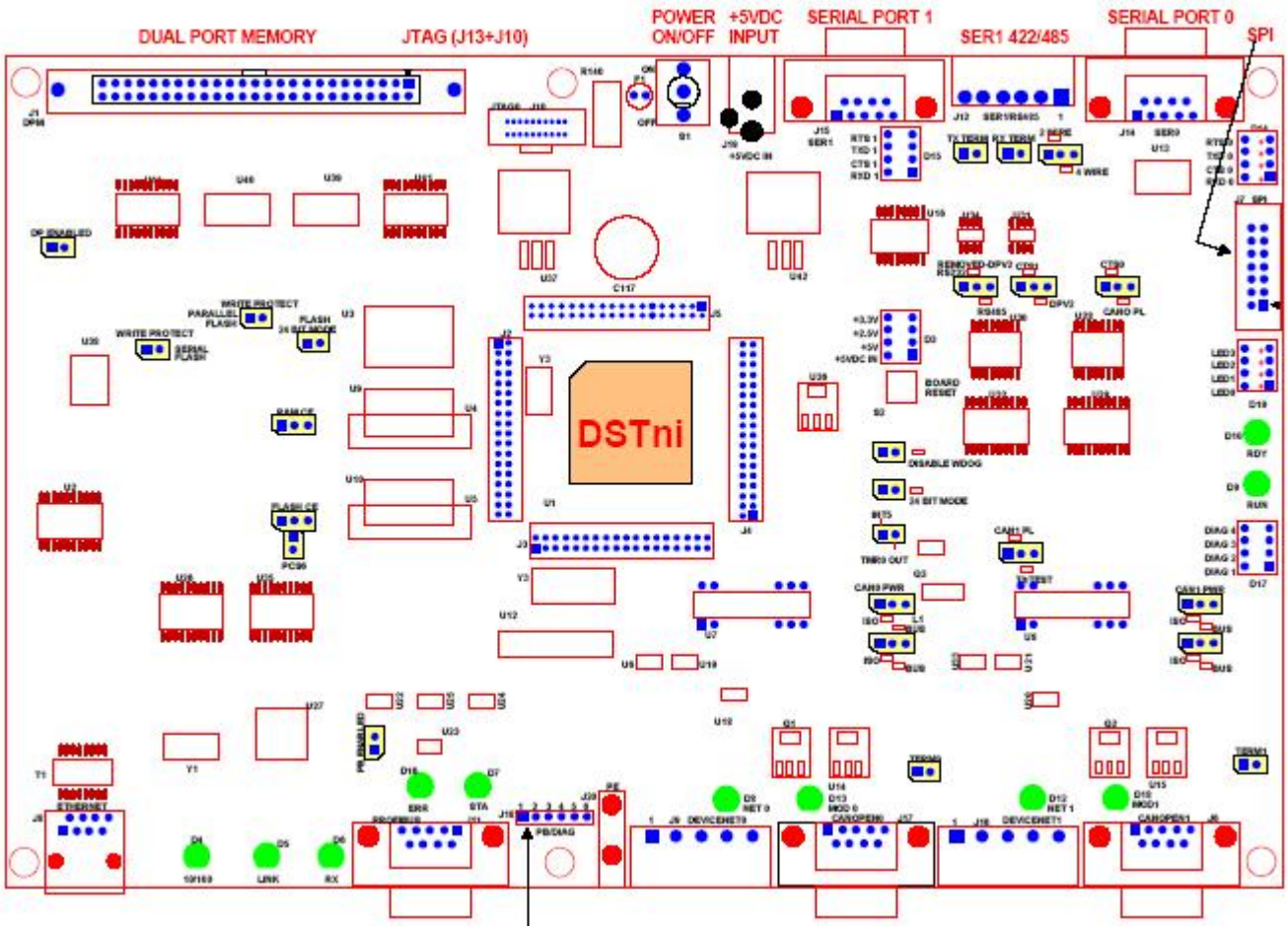


Figure 4.8: DSTni-LX evaluation platform

As in the case of the Backplane Tester system with the primary Hub1, the remote PC connects to the evaluation board using the Ethernet interface. The Server application is downloaded from the remote PC, via the serial port, and it can be stored in any of the memories existent on the evaluation board (RAM memory, the serial flash or the parallel flash). However, once the server application is downloaded and starts executing, the Ethernet connection with the client (on the remote PC) is established and the server stays in an infinite loop, waiting to serve the client requests.

The server application can treat only those requests from the client who contains low-level MDIO commands, such as write or read MDIO. The client used by the Backplane Tester application sends requests in terms of packets with write or read MDIO commands, which are handled by the server and acknowledged back.

For the purposes of the client-server validation, a simple client has been created, which allows the user to send simple 'writemdio' and 'readmdio' commands from a terminal prompt, or to execute a script containing write-read MDIO commands. The client application created for the validation purposes is called 'MDIO_Client'. The server application is the same as for the final Backplane Tester system. The evaluation platform does not contain the serdes devices required by the Backplane Tester application. As we cannot write or read any value to or from serdes registers, we validate our software by scoping the corresponding MDIO outputs or inputs at the controller level, to see if the software generated MDIO frames are looking as expected (see figure 3.1).

WRITEMDIO validation

For validating the write MDIO command, the user sends the following WRITEMDIO command (see figure 4.7) from the remote PC terminal:

```
writemdio:A:0x0:0x802a:0x0
```

Figure 4.9: WRITEMDIO command used for client-server validation

The oscilloscope is used for probing the signals at the Lantronix pins - MDC and MDO_A - where the MDIO (Address and Write) frames, corresponding to the above WRITEMDIO command are sent. According to MDIO frames description in figure 3.1, we would expect the following content for the MDIO Address and MDIO Write frame, for the WRITEMDIO command in figure 4.9.

MDIO Address frame
PRE | ST | OP | PhyAdr | RegAdr | TA | REG_ADR |
[1...1 | 01 | 00 | 00000 | 11111 | 10 | 1000000000101010]
MDIO Write frame
PRE | ST | OP | PhyAdr | RegAdr | TA | REG_DATA |
[1...1 | 01 | 10 | 00000 | 11111 | 10 | 0000000000000000]

Figure 4.10: Address and Write MDIO frames corresponding to WRITEMDIO command

Probing MDC and MDO_A, we obtain the MDIO frames at the Lantronix for the WRITEMDIO instruction above, shown in the figure 4.11. In that figure four signals can

be seen: the first is the clock (MDC), while the second corresponds to a zoom on the clock, the third signal is the MDO_A, while the fourth is the zoom of the MDO_A. On the first signal, MDC, a gap can be distinguished in the middle of the signal. This corresponds with the end of the MDIO Address frame and the beginning of the MDIO Write frame.

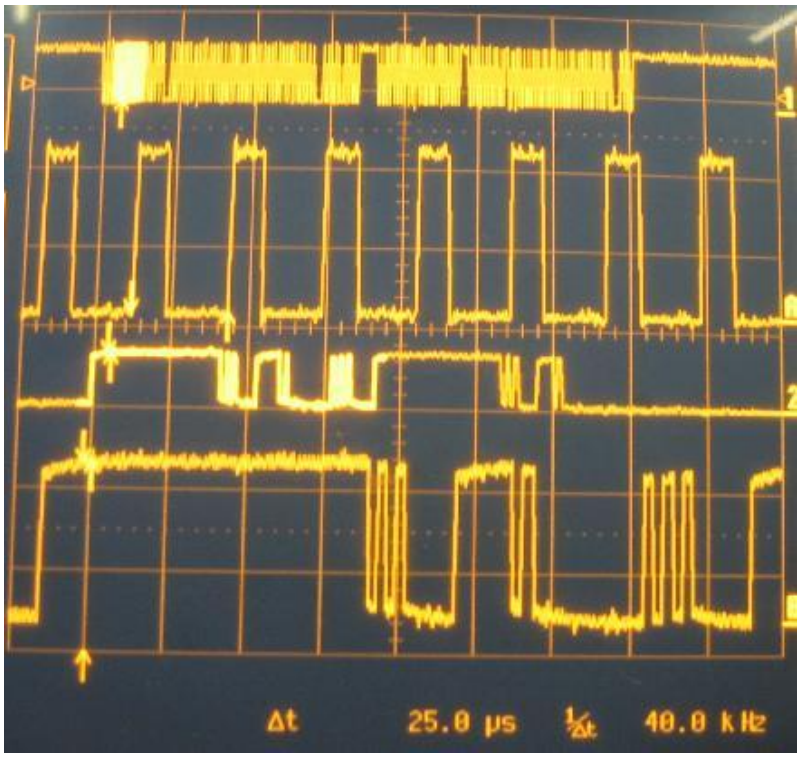


Figure 4.11: Oscilloscope probes on MDC and MDO_A pins at the Lantronix

On the zoom of the MDC (second signal), I have measured the clock period, which appears to be equal to 25us. This corresponds to a clock frequency of 0.4MHz, a low speed clock signal in the limits of the MDIO specifications.

I have checked in details, bit-by-bit, the content of the above MDIO Address and MDIO Write frame and they are as expected. The WRITEMDIO command compatibility with a real PHY device (Marvell serdes) cannot be checked until the Backplane Tester boards will come from the manufacturer. However, until then it is not much more to do about the WRITEMDIO validation than the present test. We concluded that the WRITEMDIO command has been properly executed and that the validation of the WRITEMDIO has been successfully passed.

READMDIO validation

A second step for validating the server application is to send a READMDIO command from the client and to probe the generated frames (MDIO Address and MDIO Read) at the Lantronix level.

Concerning the READMDIO validation, we are in the same situation as in the previous WRITEMDIO test: no PHY device existent on the board, whose register value to read. Therefore, we connect one of the Lantronix input pins (MDI_A3) to GND and we performed the READMDIO of thgis pin. Then, we connect the same pin to the 3.3V power and we read again its status.

For validating the READMDIO command, the user sends the following command from the remote PC terminal:

```
readmdio:A:0x3:0x0
```

Figure 4.12: READMDIO command used for client-server validation

The oscilloscope is used for probing the signals at the Lantronix pins - MDC and MDI_A3 - where the MDIO (Address and Read) frames, corresponding to the above READMDIO instruction are sent. According to MDIO frames description in figure 3.1 and 3.2, we would expect the following content for the MDIO Address and MDIO Read frame for the above READMDIO command.

MDIO Address frame

PRE | ST | OP | PhyAdr | RegAdr | TA | REG_ADR |

[1...1 | 01 | 00 | 00000 | 11111 | 10 | 1000000000101010]

MDIO Read frame

PRE | ST | OP | PhyAdr | RegAdr | TA | REG_DATA |

[1...1 | 01 | 01 | 00000 | 11111 | z |

Figure 4.13: Address and Read MDIO frames corresponding to WRITEMDIO command

For the MDIO Read frame, the second bit of the TA field and the REG_DATA is sourecd by the PHY device. In out case, the value will be determined by the pull-down (connection to Ground) or by the pull-up (connection to 3.3V power source) at the Lantronix level.

At the remote PC terminal, the following acknowledge is received from the server, when the MDI_A3 is connected to the Ground:

ACK COMMAND=ACK\READMDIO:A:0x3:0x0\0x0

Read Register DATA=0x0

Figure 4.14: READMDIO when the read pin is connected to the Ground

Now, when the MDI_A3 pin is connected to the power source, the acknowledge received from the server looks as below:

ACK COMMAND=ACK\READMDIO:A:0x3:0x0\0xffff

Read Register DATA=0xFFFF

Figure 4.15: READMDIO when the read pin is connected to the Power Source

Due to the read values obtained for the two situations presented above, we concluded that the present validation test of the READMDIO command is successfully passed and that the READMDIO command is corrected executed at the server level.

WRITEMDIO and READMDIO validation

Even the validation of both WRITEMDIO and READMDIO commands went as expected, modifications can appear when the Lantronix is driving the MDIO bus to write or to read the register contents of a 'real' PHY device. However, this behavior cannot be predicted with the present evaluation board and a complete functionality of the client-server application can only be demonstrated when the Backplane Tester boards will be received from the manufacturers.

5 Conclusions

The implementation of the Switch Fabric for the 10 Gigabit Ethernet switch is based on a passive backplane on which multiple 10 Gigabit Ethernet Line cards and one or two switch fabric cards are plugged. An Advanced Telecom Computing Architecture (AdvancedTCA) has been selected for the design of the T6Pro 10 Gigabit Ethernet Switch. This standard has been defined by the PCI Industrial Computer Manufacturers Group (PICMG) to provide solution for the next generation of carrier grade communications equipment. With more than 100 companies participating, PICMG collaboratively developed open specifications for AdvancedTCA®, which is the largest specification effort in PICMG's history. These new specifications, called PICMG 3.0, incorporate the latest trends in high speed interconnect technologies, next generation processors and improved reliability, manageability and serviceability.

The Switch Fabric relies on prototype silicon from Marvell mounted on prototype Printed Circuits Boards (PCB), running high speed signals across a prototype passive backplane. We are fairly confident in the performance of the Marvell Serializer/Deserializer (Serdes) chips if they perform as well in practice as they do in simulation. On the other hand, we are less confident in our prototype PCB manufacture because experience shows that it can take a couple of iterations with any given manufacturer to obtain the specified performance. We have confidence in the design simulation work done within the AdvancedTCA consortium. It is obviously extensive and professional but the actual backplane construction is subject to those PCB manufacturing control problems. The nightmare scenario is the following: the line card and switch fabric chips work properly, one or two line cards in a system show no errors but a fully populated chassis gets some low random error rate. Finding the precise cause of those errors and fixing them could consume as much effort as the original design cycle. The errors could be in the silicon or in the backplane.

In order to be able to subtract the backplane from the set of things to worry about, an AdvancedTCA Backplane Tester system has been designed and manufactured. The idea is to use transceiver chips embedding Serdes, pattern generators and checkers to fully

saturate every interconnection simultaneously for a long period of time to establish a degree of confidence to a suitable order of magnitude.

In Chapter 2, we described how the Marvell 88X2040 transceivers are used to provide 10Gbps traffic for saturating all the Fabric Channel in the Dual Star AdvancedTCA Fabric Interface.

In Chapter 3, we presented the choice we made for the Lantronix microcontroller, which is used to control the Backplane Tester system. An important design decision is that a single control point (on the primary Hub) is used for the whole Backplane Tester system. Since Marvell transceivers are controlled with a Management Data Input/Output (MDIO) serial link, the controller has to emulate the MDIO protocol through its 32 I/O pins. All the architectural decisions concerning the serdes devices control over MDIO, the generation of the control signals at the controller level and their distribution over the backplane using the AdvancedTCA Base Interface are presented. At the end of Chapter3, the final control architecture for the Backplane Tester boards is brought up. An important aspect (especially from the cost point of view) is that the control architecture has been defined so that a single PCB can be used either for Node boards either for Hub boards in the Backplane Tester. At the moment when the present report is been written, not only the architectural decisions but also the boards design have been finished. To prove this, the last figure in Chapter 3 shows the layout of such a board.

Chapter 4 is exclusively dedicated to the software architecture. Details are given, related to how the software application is built, based on the client-server model, and how the user interacts with the system for defining a given test over the AdvancedTCA backplane. The client-server application is built and validated using the reference platform of the Lantronix controller. All the possible software tests for the validation of the client-server application have been performed with the existing evaluation platform and considered successful.

In the present report, we present in details the hardware designs of the Backplane Tester system and its associated software applications. The hardware design phase is completely finished and the boards are launched for production. The software applications have been validated using the existing resources (Lantronix evaluation platform). The complete

functionality of the Backplane Tester system will be demonstrated as soon as the Hub and the Node boards will be received from the manufacturers and a detailed report about the AdvancedTCA backplane tests will be provided.

Appendix1 - SETUP Macros

The **SETUP** sequence requires a succession of macros for setting-up the serdes devices, situated at both ends of each communication link over the backplane. The serdes are setup to send or/and receive for a certain amount of time, a certain type of traffic over each link of the standard (4 differential pair) or the extended (2 differential pairs) data transport interface. The connectivity scheme for the standard interface implies node-hub connection over 4 pairs and the extended connectivity contains the connections over 2 differential pairs between 2 adjacent slots and the 2 hubs (primary and the redundant). The following sequence of macros (each of the macro repeated several times) is required for setting up the traffic over the standard and the extended links between available nodes and the primary hub1 or the redundant hub2:

- **reset_global** – reset all the system, once, at the beginning
- **initialization** – serdes reset & clock re-timing (for the standard link)
- **initialization_ext –** as above (for the extended link)
- **led_control** – programming the LEDs (corresponding to U0, U7, U12) on the node card to show the status of the link
- **Pre_Emphasis_Setup** – setup of the PE level at the transmitter
- **Lanes_Dezactivation** – power-down TX and/or RX corresponding to the differential pairs selected to be non-active in the standard link (standard link)
- **Lanes_Dezactivation_ext** – as above (extended link)
- **Set_Traffic** – setup the type of the bit sequence (PRBS or Jitter) to be send on the standard link
- **Set_Traffic_ext** – as above (extended link)

The test configuration setup (contained in the <setup.txt> configuration file) is performed at the beginning and the information (ACK) from the server is displayed in real-time to the client terminal console and it is also stored for later access in the <setup_LOG.txt>.

1.1. reset_global
Macro: reset_global

1.2. initialization
Macro: initialization:slot:hub
Excel: - slot (Sheet2.G)
- hub (Sheet2.H)
Functionality: - expansion in 6 write_mdio instructions (3 for a slot and 3 for a hub)

Reg ADDRESS	Reg DATA
0XFF2C	0X377
0X0	0Xa040
0XFF39	0X8000

Table 1: write MDIO for the 'initialization' macro

1.3. initialization_ext
Macro: initialization_ext:slot1:hub1:slot2:hub2
Excel: - slot1 / slot2 (Sheet2.I)
- hub1 / hub2 (Sheet2.J)
Functionality: - expansion in 12 write_mdio instructions (3 instruction for each slot1, slot2, hub1 and hub2), as in Table2.

1.4. led_control
Macro: led_control:slot1:slot2
Functionality: - macro expanded in 2 write_mdio instructions (1 instruction for slot and 1 for hub)

Reg ADDRESS	Reg DATA
0x8004	0x4819

Table 2: write MDIO for the 'led_control' macro

1.5. Pre_Emphasis_Setup
Macro: Pre_Emphasis_Setup:slot:hub:param_PE
Excel: - slot (Sheet2.G)
- hub (Sheet2.H)
- param_PE (Sheet1.O) – see Appendix4
Functionality: - expansion in 16 write_mdio instructions (8 instruction for slot and 8 for hub)

Reg ADDRESS	Reg DATA
0XFF28	0XFF03 (twice)
0XFF29	0XFF03 (twice)
0XFF2A	0XFF03 (twice)
0XFF2B	0XFF03 (twice)

Table 3: write MDIO for the 'Pre_Emphasis_Setup' macro

1.6. Lanes_Dezactivation
Macro: Lanes_Dezactivation:slot:slot2hub:hub:hub2slot
Excel: - slot (Sheet2.G)

- hub (Sheet2.H)
- slot2hub (Sheet1.C/D/E/F)
    - 4 numbers defining the lanes status (Lane0-Lane3) for the links from NODE to the HUB
- hub2slot (Sheet1.G/H/I/J)
    - 4 numbers defining the lanes status (Lane0-Lane3) from HUB to the NODE)

Functionality:

- Deactivate a specific lane (Lane0-Lane3) in the SERDES, by powering off its corresponding transmitter/receiver in the node/hub (see Table5)
- expansion in 8 write_mdio instructions (4 instructions for slot and 4 for hub)

slot2hub (L0-L3)	**hub2slot (L0-L3)**	**NODE**	**HUB**
0	0	Power off TX and RX	Power off TX and RX
0	1	Power off TX	Power off RX
1	0	Power off RX	Power off TX
1	1	Default (see Table6)	Default (see Table6)

Table 4: Disabling lanes

The default values for every lane from L0 to L3 are:

Lane_i_Control_Reg_ADR	**Lane_i_Control_Reg_DATA**
Lane(0)0x8000	0x0730
Lane(1)0x8001	0x0030
Lane(2)0x8002	0x0030
Lane(3)0x8003	0x0030

Table 5: Default Lane[i] value when the link is enabled

To disable the TX or the RX in a lane (conform to Table5), it is applied on the default Lane[i] value (in Table6) a corresponding mask (Mask_TX or Marsk_RX).
Mask_TX = 0x0800 - Power off TX in HUB or NODE
Mask_RX = 0x0080 - Power off RX in HUB or NODE

1.7. Lanes_Dezactivation_ext

Macro: Lanes_Dezactivation_ext:slot:slot2hub:hub:hub2slot:slot_num

Excel:
- slot (Sheet2.I)
- hub (Sheet2.J)
- slot2hub (Sheet1.K/L)
    - 2 numbers defining the lanes status (Lane0-Lane1 or Lane2-Lane3) for an extended link from NODE to the HUB
- hub2slot (Sheet1.M/N)
    - 2 numbers defining the lanes status (Lane0-Lane1 or Lane2-Lane3) for an extended link from HUB to the NODE
- slot_num = 0 (if slot1) / 1 (if slot2)

Functionality:
- deactivate a specific lane (Lane0-Lane1 or Lane2-Lane3) in the SERDES, by powering off its corresponding transmitter/receiver in the node/hub, as presented in Table1
Note: Special attention should be taken as it is not necessary anymore a Lane[i] to Lane[i] communication between the node and the hub. The possible scenarios are presented in Table7.
- expansion in 4 write_mdio instructions (2 instructions for slot and 2 for hub)

HUB	SLOT	Lanes in the HUB	Lanes in the SLOT
Hub1	Slot1	Lane0-Lane1	Lane0-Lane1
	Slot2	Lane2-Lane3	
Hub2	Slot1	Lane0-Lane1	Lane2-Lane3
	Slot2	Lane2-Lane3	

Table 6: Extended lanes mapping between a Node and a Hub

1.8. Set_Traffic

Macro: Set_Traffic:slot:hub:param_prbs_or_jitter:param_PRBS:param_Jitter
Excel: - slot (Sheet2.G)
- hub (Sheet2.H)
- param_PRBS (Sheet1.P) - see Appendix4
- param_Jitter (Sheet1.Q) - see Appendix4

Functionality: - expansion in 2 write_mdio instructions (1 instruction for slot and 1 for hub), depending on the user-defined parameters, as follows:

- **PRBS traffic** – user defined parameter in the Excel table
  - param_prbs_or_jitter=1
  - param_PRBS = 0 (2^23-1) // 1 (2^31-1) // 2 (2^7-1)

Reg ADDRESS	**Reg DATA** (2^23-1)	**Reg DATA** (2^31-1)	**Reg DATA** (2^7-1)
0x802A	0x000A	0x000F	0x006A

Table 7: write MDIO command for 'Set_Traffic' macro when PRBS traffic defined

- **JITTER traffic** – user defined parameter in the Excel table
  - param_prbs_or_jitter=0
  - param_Jitter = 0 (48A.1) // 1 (48A.2) // 2 (48A.3) // 3 (48A.4) // 4 (48A.5)

Reg ADDRESS	**Reg DATA** (48A.1)	**Reg DATA** (48A.2)	**Reg DATA** (48A.3)	**Reg DATA** (48A.4)	**Reg DATA** (48A.5)
0x802A	0x0C00	0x1C00	0x2C00	0x3C00	0x4C00

Table 8: write MDIO command for 'Set_Traffic' macro when Jitter traffic defined

1.9. Set_Traffic_ext

Macro:
Set_Traffic_ext:slot1:hub1:slot2_hub2:param_prbs_or_jitter:param_PRBS:param_Jitter
Excel: - Slot1 / slot2 (Sheet2.I)
- Hub1 / hub2 (Sheet2.J)
- param_PRBS (Sheet1.P)

- param_Jitter (Sheet1.Q)

Functionality:

- expansion in 4 write_mdio instructions (1 instruction for each slot1, slot2, hub1 and hub2) as in 1.8.

Appendix2 - RESET COUNTERS Macros

The **RESET COUNTERS** sequence is necessary to be performed every 100seconds for all active serdes devices in the test (indication from the Marvell 88X2040 datasheet). At this interval, the error counters are read, their value is stored at the server level and then they are reset. Before getting out of reset, the traffic is setup again. Corresponding macro in <reset_counters.txt> configuration file:

- **reset_counters**

The counters reset is performed every 100seconds and the information (ACK from the server) is displayed in real-time to the terminal console and it is also stored in the <reset_LOG.txt>.

2.1. reset_counters
Macro: reset_counters:slot:param_PRBS:param_Jitter
Excel: - slot (Sheet2.G/H/I/J)
- param_PRBS (Sheet1.P)
- param_Jitter (Sheet1.Q)

Functionality:
- expansion in 2 write_mdio instructions:
  - 1st instruction for resetting the error counters
  - 2nd instruction for setting the traffic as PRBS or Jitter

Reg ADDRESS	Reg DATA	Reg DATA
0X802A	0x0010	
0X802A	PRBS	Jitter

Table 9: write MDIO commands for the 'reset_counters' macro

Appendix3 - READ Macros

During the **READ** sequence, contained in the <read.txt> configuration file, there are gathered statistics in terms of serdes status (macro read_status) and error counters (macro read) from all the serdes devices actively implied in the defined test:

- **Read_status**
- **Read**

The reading is performed every 60seconds and the information are displayed in real-time to the terminal console and they are also stored in the <read_LOG.txt>.

3.1. Read_Status

Macro: Read_Status:slot:hub:print
Excel: - slot (Sheet2.G)
- hub (Sheet2.H)
- print – 0 (don't print) / 1 (do print read statistics)

Functionality: - expansion in 6 read_mdio instructions (3 instructions for slot and 3 for hub)

Register	Reg ADDRESS
Status Register 1	0x0001
Status Register 2	0X0008
PCS Status Register	0X0018

Table 10: MDIO READ commands for the 'read_status' macro

3.2. Read

Macro: Read:slot:param_prbs_or_jitter:print:param_PRBS:param_Jitter
Excel: - slot (Sheet2.G/H/I/J)
- param_PRBS (Sheet1.P)
- param_Jitter (Sheet1.Q)

Functionality:
- expanded in 10 read_mdio instructions for PRBS traffic
- expanded in 12 read_mdio instructions for Jitter traffic

Register	Reg ADDRESS	Type of traffic
Random_Jitter_Seq_Error_Counter_LSB[0] & MSB[0]	0x8020 & 0x8021	PRBS & Jitter
Random_Jitter_Seq_Error_Counter_LSB[1] & MSB[1]	0x8022 & 0x8023	PRBS & Jitter
Random_Jitter_Seq_Error_Counter_LSB[2] & MSB[2]	0x8024 & 0x8025	PRBS & Jitter
Random_Jitter_Seq_Error_Counter_LSB[3] & MSB[3]	0x8026 & 0x8027	PRBS & Jitter
Random_Jitter_Seq_Timer_ ADR_LSB & MSB	0x8028 & 0x8029	PRBS & Jitter
Jitter_Packet_Transmit_Counter_ ADR_LSB & MSB	0x802C & 0x802D	Jitter

Table 11: MDIO READ commands for the 'read' macro

Appendix 4 - Parameters setup by the user in EXCEL

- param_PE

(Sheet1.O) - see the 'Legacy PE support' (datasheet 88X2040)

- param_PRBS

(Sheet1.P) – possible values are: 2^23-1 // 2^31-1 // 2^7-2

- param_Jitter

(Sheet1.Q) – possible values are: 48A.1 // 48A.2 // 48A.3 // 48A.4 // 48A.5

- t_def_test

(Sheet1.R) – duration of the test in seconds

References

"PICMG 3.0 – Advanced Telecommunication Computing Architecture", Version 0.91, December 2002

"P802.3ae - Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation", IEEE, 2001

"88X2040-Alaska X Integrated Single Chip Quad 3.125/3.1875 Gbps Transceiver", Revision D, Marvell datasheet

"DSTni-LX Lantronix Evaluation Board Schematics"

"DSTni-LX Lantronix Data Book", Revision D, Part Number 900-252

"High Performance Backplane Design Using the Marvell Alaska X Quad 3.125Gb/s SERDES", Kamal Dalmia, Nov.2001, Marvell White Paper,

"Intel LXT971A – 3.3V Dual-Speed Ethernet PHY Transceiver"

"Using Decoupling Capacitors", San Jose, March 1999, Cypress Semiconductor Corporation

"MC100LVE111 – 3.3V ECL 1:9 Differential Clock Driver", ON Semiconductor