Design Verification and Performance Analysis of Serial AXI Links in Broadcom System-on-Chip

by

Simran Sarai

B.A.Sc., Simon Fraser University, 2010

Research Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering Science

in the
School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY

Spring 2014

APPROVAL

Name: **Simran Sarai**

Degree: **Master of Engineering Science**

Title of Thesis: ***Design Verification and Performance Analysis of Serial AXI Links in Broadcom System-on-Chip***

Examining Committee: **Defence Chair:** Jie Liang
Associate Professor

Ivan Bajic
Supervisor
Associate Professor

Trac Hoang
External Examiner
Senior Principal Engineer
Broadcom Canada Ltd.

Date Defended/Approved: Jan 10, 2014

ABSTRACT

Design verification is an essential step in the development of any product. Also referred to as qualification testing, design verification ensures that the product as designed is the same as the product as intended. In this project, design verification and performance analysis of Thin Advanced Extensible Interface Links (T-AXI) is conducted on a Broadcom's SoC (System on Chip). T-AXI is a Broadcom's proprietary bus that interfaces all the subsystems on the System-on-chip (SoC) to the system memory. Test cases are developed to verify the functionality of the T-AXI and performance verification is implemented using scenarios derived from real world examples. A Field Programmable Gate Array (FPGA) is used to emulate the SoC design and C programming is used to write the test cases. The test results verify the T-AXI functionality and the performance analysis supports the theoretical calculations.

ACKNOWLEDGEMENTS

I would like to thank my manager, Allan Chan for supervising this project and providing guidance and critic in each phase of this project. I would also like to thank Dr. Ivan Bajic for co-supervising me in this project and providing valuable suggestions.

I would also like to thank Broadcom DVT for their help and co-operation throughout my project. Finally, I wish to thank my family and friends for their encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AXI	Advanced eXtensible Interface
ASIC	Application Specific Integrated Circuit
SoC	System on Chip
ACE	AXI Coherency Extensions
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
AMBA	Advanced Microcontroller Bus Architecture
DMA	Direct Memory Access
IP	Intellectual Property
DVT	Design Verification Testing
RTL	Register-Transistor Logic
QoS	Quality of Service
DDR SDRAM	Double data rate synchronous dynamic random-access memory

1.0 Introduction

In recent years, a smartphone is no longer a device for the road warriors; but rather, it has become the most personal electronic device that the consumers own to manage their online and offline lives. The trend to run various smartphone applications simultaneously, while ensuring that consumers do not have to wait for the “hourglass”, has triggered a cutting-edge research on reducing the round trip latency between memory and peripherals. The important aspect of a System-on-Chip (SoC) not only includes which components or blocks it houses, but also how they interconnect. One of the de-facto on-chip bus standards is the Advanced Microcontroller Bus Architecture (AMBA).

AMBA [1] is a registered trademark of ARM Limited and is an open standard, on-chip interconnect specification for the connection and management of functional blocks in a SoC. An AMBA-based architecture connects on-chip memory, the processor and other Direct Memory Access (DMA) devices to a high-performance system backbone bus that is able to sustain an external memory bandwidth. This bus provides a high-bandwidth interface between the elements that are involved in the majority of transfers. AMBA specification is further subdivided into more specific interfaces such as Advanced eXensible Interface (AXI), AXI Coherency Extensions (ACE), Advanced High-performance Bus (AHB), Advanced Peripheral Bus (APB), etc.

In this project, Broadcom's Thin Interconnect Intellectual Property (IP) is examined and evaluated. This IP is AXI 4 compliant, which is part of AMBA 4.0 released in 2010 [1]. The AXI interconnect facilitates transactions between various subsystems of the SoC and the system memory, as shown in Figure 1. The subsystems could be any sort of traffic generator peripheral, such as processor, video, LAN, graphic, modem, DMAs, etc. These peripherals send reads and writes to the memory using Thin AXI, which is further explained in this report.

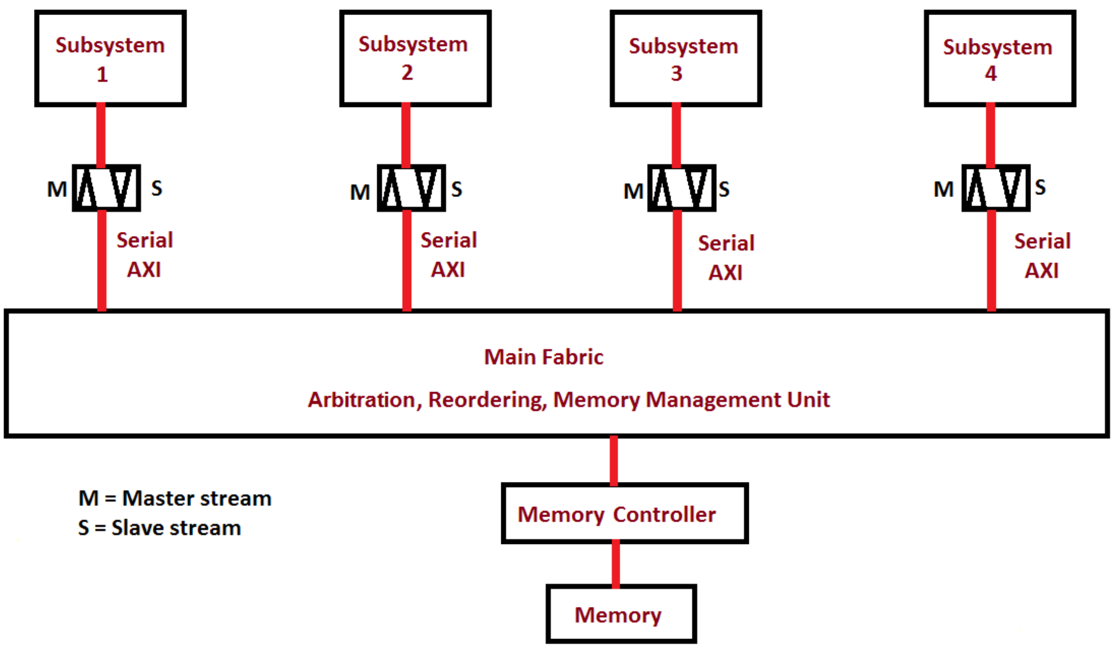


Figure 1: Block Diagram of System-on-Chip

1.1 Objective

The aim of this project is to understand and verify the design of Thin AXI Link used in Broadcom's SoC and analyze the performance metrics. In the full cycle, starting from Register-Transistor Logic (RTL) design through to marketing the SoC, Design Verification Testing (DVT) is an essential step. DVT ensures that the designed product is the same as the intended product. Typically, pre-ASIC (Application Specific Integrated Circuit) design verification is performed on an emulator such as Field Programmable Gate Array (FPGA), Cadence Palladium, etc. Hardware emulation imitates the behavior of SoC and is based on a hardware description language (e.g. Verilog) source code. Once the chip specifications are verified, the SoC is taped out. Then, post-ASIC design verification is performed on the ASIC before releasing the Chip in the market. In this project, pre-ASIC DVT is performed on the Thin links of the SoC.

1.2 Document Outline

This document is organized into six main chapters which describe the design, implementation and testing of the Thin AXI Link in Broadcom’s SoC. Chapter 1 provides an introduction to the report and Chapter 2 provides the background information on the Thin AXI and an overview of AXI4 specifications ported into the Broadcom’s IP. Chapter 3 describes the design and implementation of the Thin AXI Link. Chapter 4 discusses the test bench set up to verify the functionality of the Thin AXI Link. Chapter 5 describes the test plan and presents the test results. Finally, Chapter 6 concludes the report by summarizing the Thin Links implementation and test results.

2.0 Background

System-level interconnect presents a challenging design of servicing all the masters meticulously and avoiding traffic bottle-necks. There are two proposed architectures:

- An architecture (say Arch1) that uses off-the-shelf AXI-4 fabric from ARM, in combination with wide source-synchronous busses. A source-synchronous interface is one where the clock accompanies the data on its journey from source to destination, and is used to clock the data into the receiver. A synchronization stage is then used to transfer the received data back to the global clock domain. An advanced multi-ported, re-ordering memory controller implements a Quality of Service (QoS) scheme based on configurable, per-master, time-to-live counters. The ARM, the modem and the multimedia subsystems have dedicated memory controller ports to minimize memory access latency. All other subsystems use this architecture for memory accesses. The more centralized QoS comes at the cost of implementation complexity and routing overhead, and does not fully eliminate local decision-making.

- An architecture (say Arch2) that uses a proprietary single-address-bus variant of AXI-4, with wide source-synchronous busses and a simple “sequencer-style” memory controller. A distributed QoS scheme uses credit arbitration at each multiplexing point in the fabric to share bandwidth between masters, with a panic signal to permit priority escalation with inheritance.

Both architectures have a considerable amount of bus multiplexing prior to arrival at the memory controller, and therefore their QoS implementation is distributed with imperfect local decisions being made on the basis of local information.

In this SoC, there are two DDR4 memory devices implemented. From the experience of previous Broadcom SoCs, a 128-bit master running at 250MHz is able to instantaneously saturate the bandwidth provided by a single 32-bit LPDDR2-800 device.

Bandwidth of Master = 128bits * 250MHz = 32000 Mega bits/second (Mb/s).

Bandwidth of Memory = 32bits * 800MHz = 25600 Mb/s.

By extrapolation, we would require 256-bit masters at 500MHz to come “close to” saturating the bandwidth provided by a pair of 32-bit DDR4L-2133 devices.

Bandwidth of Master = 256bits * 500MHz = 128000 Mb/s.

Bandwidth of Memory = 2 * 32bits * 2133MHz = 136512 Mb/s.

To match the available downstream SDRAM bandwidth, a multi-ported controller is needed which:

- Accumulates write data from several slower slave ports into internal buffers until the buffers are full and then, the contents of the buffer are transmitted at high speed to the SDRAM devices.

- Receives read data at high speed from the SDRAM devices into internal buffers and then the contents are forwarded at the earliest opportunity via the appropriate slave port.

To eliminate the delays caused by distributed QoS, and to provide an adequate supply of upstream bandwidth, this SoC interconnect uses a star-topology network to connect each non-CPU bus master with a dedicated port on the memory controller front-end. Each connection, and each controller port, provides 2GB/s of duplex bandwidth (equivalent to 32-bit AXI at 500MHz).

This choice of topology places a heavy burden on implementation, due to a large number of signals which converge on a single point. Therefore, narrow, fast and self-timed busses (as shown in Figure 2) are required to ameliorate the resulting congestion.

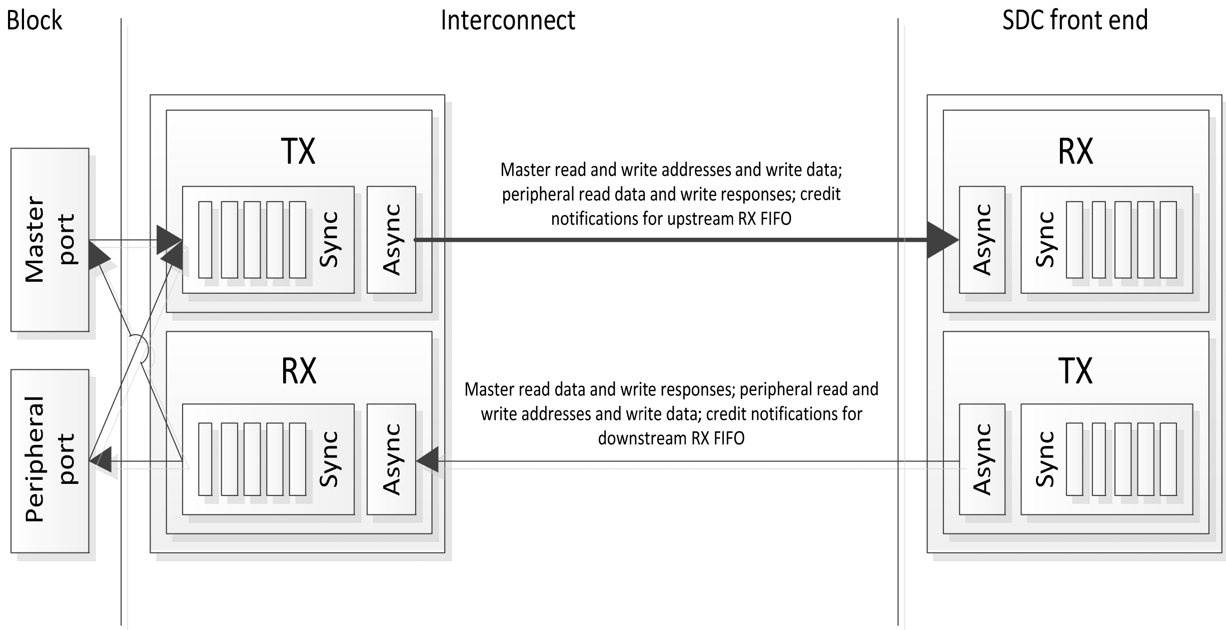


Figure 2: Fast, Narrow and Self-timed busses

In Figure 2,

- AXI transactions originating at the master are packetized into a local Transmit (TX) buffer.
- Data is read from the TX buffer by an asynchronous transmitter and sent over the bus.
- Data is captured by an asynchronous receiver, and copied into a Receive (RX) buffer.
- Transactions are reconstructed and consumed by logic in the memory controller.
- Read data and write responses return to the master by a similar mechanism.

This bus architecture is derived from AXI 4 ARM architecture which is described in next section.

2.1 AXI 4 Design Extracts

This document describes the flavour of AXI4 that is used in Broadcom SoC's Thin AXI design. It outlines some of the key behaviours of the system and offers some design guides for the AXI components. The masters and slaves that are connected by the Interconnect are described in Section 2.2 and Section 2.3. Some of the AXI4 features that are incorporated in Broadcom's bus architecture are:

- AXI4 has dual address busses. Reads and writes are completely decoupled.
- AXI4 does not have the inherent ordering. Read/write conflicts need to be detected and handled by the masters.
- AXI4 supports 256 beat bursts for INCR (unspecified length) bursts.
- AXI4 bursts cannot cross a 4K boundary of the memory.
- AXI4 renames Priority as QOS.

The following AXI 4 features are not supported by the Thin Interconnect:

- Lock signals and exclusive accesses are not supported.
- Regions are not supported.

2.2 Master Behaviour

This section discusses some of the requirements that masters/subsystems should abide in accordance to AXI 4 protocol. All masters should include a subset of the full AXI4 specification and should be suitable for use in future AXI4 systems that have a different subset of the specifications. AXI Masters must have the following behaviour:

- Must be able to limit the maximum read/write burst size and the number of outstanding requests. This limit is in powers of 2 for ease of implementation and allows system bandwidth behaviour to be tuned.
- Bursts > 16 beats can only be INCR which limits a maximum of 16 beats to reduce infrastructure storage requirements.
- Bursts cannot cross a 4K boundary.
- Transactions should be labeled as "NOT modifiable" ACACHE[1]=0.
- If masters launch read requests with different IDs, they must be able to handle the read data coming back out-of-order.

Masters must expect the following ordering behaviour from the AXI subsystem.

- Transactions with the same direction and the same ID to the same slave will remain in order. For reads, the data will be returned in the same order. For writes, the responses will be returned in order.
- There is no guarantee of order between reads and writes. You must wait for a response from one transaction before starting the next.

- There is no guarantee of the order of transactions in the same direction but to different slaves. Read data (or write responses) to two different slaves may come back in any order even if they have the same ID. The memory controller will handle the SDRAM page interleaving, making the SDRAM look like one slave. Peripherals will be located on a 4K boundary and so masters that access peripherals should treat every 4K boundary as a separate slave.

In Broadcom's SoC, all the subsystems adhere to the above stated requirements of AXI 4 as well as follow the custom design guidelines given below:

- Use a different AXI ID for each transaction. Data read from memory may be out of order and using a different AXI ID allows the memory controller to re-order requests and responses and increase the SDRAM efficiency. System performance may be reduced if the same ID is used to different slaves because memory subsystem has to re-order reads that go to different physical SDRAM controllers.
- Indicate any outstanding requests still in flight while the link is powering down. Care should be taken to ensure that any outgoing requests are counted from the moment that they are started, and not from when they are accepted.
- Provide a fine grained QoS control of their (masters) bus behaviour to aid in system tuning and bandwidth limiting.

- Masters should behave in a fair and responsible manner. If they drive a high QoS value and continual back-to-back requests, then they will saturate the bus to the detriment of other masters in the system. It is the responsibility of the Master to control its bus behaviour and play fairly in a system. It is not the responsibility of the infrastructure or memory controller to throttle a greedy master. A system can usually support one greedy master by making it the lowest priority master in the system, so that it hoovers up all the spare bandwidth. However if there are several greedy masters, then a method needs to be in place to allocate the spare bandwidth amongst them.
- Masters should try and access bulk data in large bursts. Caches should be employed if necessary. DDR4 SDRAM have a minimum burst size of 256 bits, so a single beat 128 bit AXI transfer will only get 1/2 the possible bandwidth. A 2 beat burst should be considered as an absolute bare minimum, and a 4 beat burst is much more desirable.
- Masters should drive their priority outputs to zero when inactive.

2.3 Slave Behaviour

This section discusses some of the requirements that slaves/memory should abide in accordance to AXI 4 protocol. In general, slaves should cope with the full AXI4 specification and should report any unsupported behaviour with ASSERTS and return the appropriate response to unsupported behaviour on the bus. Slaves must have the following behaviour:

- Must be able to support up to 256 beat bursts. Bursts > 16 beats can only be INCR. If a slave does not support >16 beats then it should have an ASSERT to detect this, and it should return a bus error.
- Slaves do not have to support bursts across a 4K boundary (as this needs 32 bit address logic) but can do so if backwards compatibility with VC-AXI is required.
- Write response (BResp) cannot be issued until both address and last data have completed.
- Slaves must cope with all juxtaposition and combinations of address and write data, i.e. the address and the first beat of write data are no longer guaranteed to be in the same cycle.
- Error conditions should be returned on Bresp and Rresp when an AXI access is invalid.

Slaves must obey the following ordering rules:

- Once a response has been given, the slave must maintain data order with any subsequent transactions, i.e. once inside a slave, then data order must be preserved.
- Slaves can respond to transactions with different ID's in any order.

3.0 Thin AXI Architecture and Implementation

Thin AXI interconnect is the main fabric that routes various thin AXI busses onto the two SDRAM controllers. The main fabric (MFAB) has to sort out any re-ordering issues, provide buffering to prevent slow T-AXI links from stalling the system, enforce memory protection and also provide the system MMU functionality required for a memory system that allocates scattered 4K pages. The AXI channels are carried over the high-speed T-AXI bus in a simple Time Division Multiplexing (TDM) fashion using a synchronous T-AXI clock. In addition, the bus supports non-AXI commands for link control and a simple address compression scheme.

3.1 Signal List

The signal list for the T-AXI bus is given in Table 1 and the visual presentation is provided in Figure 3. Note that the naming convention uses downstream to indicate the channel moving data/commands away from the system memory and upstream for the reverse direction.

Table 1: Signal list of Thin AXI bus

Signal	Description
clk_taxi	Synchronous clock covering up/downstream blocks and repeaters.
rst_taxi_n	Async reset with rising edge synchronised to clk_taxi. Clk_taxi runs much slower during reset.
taxi_d_clkreq	Async clock request from downstream module to the clk_taxi
taxi_u_clkreq	Async clock request from upstream module to the clk_taxi
taxi_d_valid	Data valid marker.
taxi_d_data	Downstream channel carrying commands and data muxed.
taxi_d_stall	Synchronous stall from receiver to the transmitter and passing through each repeater. Causes a rippled stall up the T-AXI link with each repeater storing data in a holding register to cope with the stall delay per stage.
taxi_u_valid	Data valid marker. Always asserted for cycles carrying AXI data and negated for link control commands.
taxi_u_data	Upstream channel carrying commands and data muxed.
taxi_u_stall	Synchronous stall from receiver to the transmitter and passing through each repeater. Causes a rippled stall up the T-AXI link with each repeater storing data in a holding register to cope with the stall delay per stage.

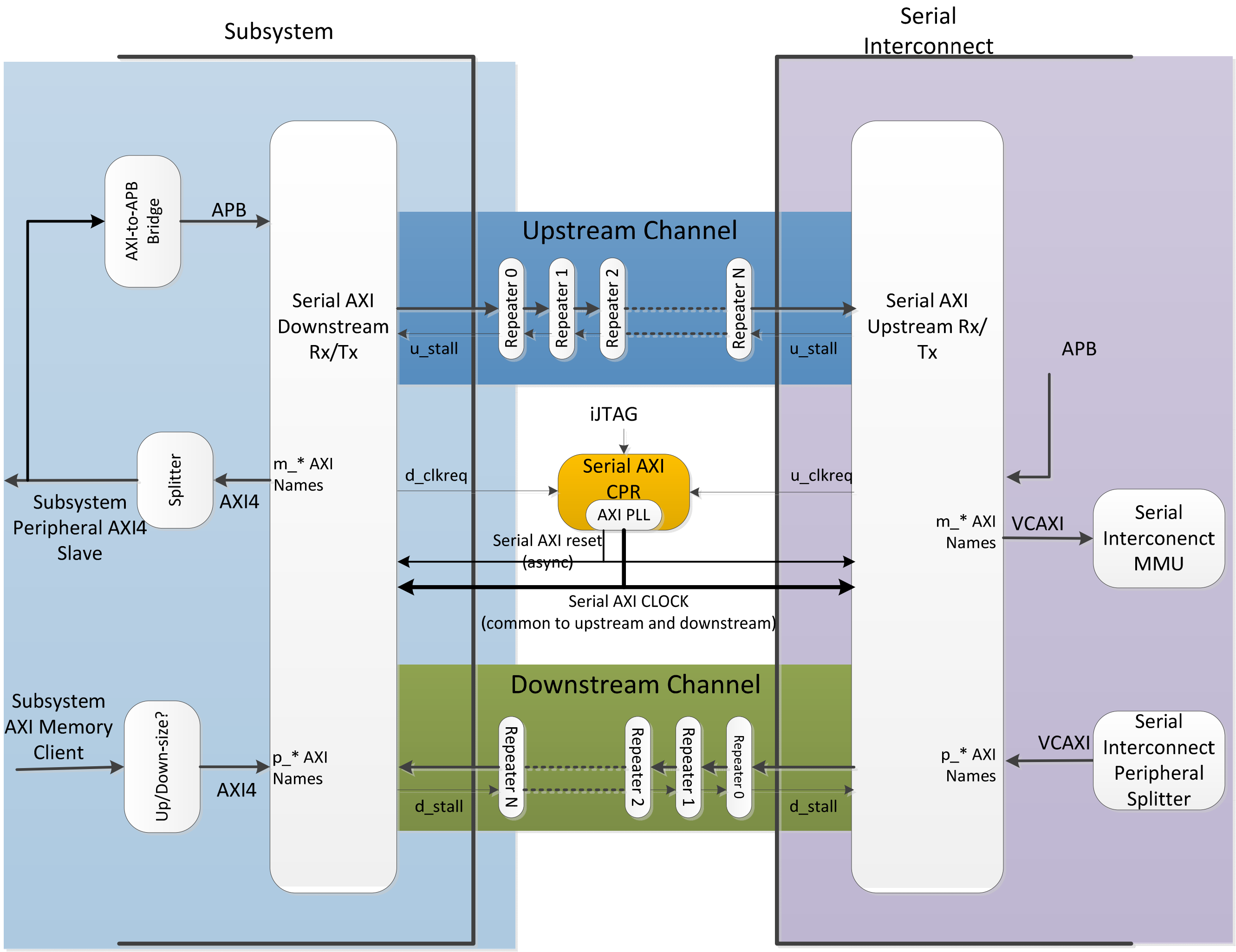


Figure 3: Visual presentation of signals of Thin AXI

3.2 Supported AXI Signals

AXI4 supports a large number of signals, not all of which are essential for the particular subset of AXI used within any particular system. By limiting the set of AXI transfers supported, we can increase the throughput of the T-AXI links. The mapping in Table 2 shows the AXI4 signals supported by the T-AXI link in SoC.

Table 2: AXI4 and Thin AXI Signal mapping

AXI 4 Signal	T-AXI Support	Notes
ACLK	clk_d/clk_u	Downstream and upstream source synchronous clocks
ARESETn	-	Not supported. Resets are sent as commands
A*ID[3:0]	A*ID[9:0]	Up to 10-bits of ID are supported.
A*ADDR[31:0]	A*ADDR[35:0]	64GB address range is supported
A*LEN[7:0]	A*LEN[3:0]	Limits burst size to 16 beats (same as AXI3)
A*SIZE[2:0]	A*SIZE[2:0]	Fully supported
A*BURST[1:0]	A*BURST[1:0]	Fully supported
A*LOCK[1:0]	-	Not supported
A*CACHE[3:0]	-	Not supported as there is no L3 cache in SoC.
A*PROT[2:0]	A*PROT[2:0]	Only A*PROT[1] is carried over T-AXI hence only two levels of security can be supported.
A*QOS[3:0]	-	Fully supported. Maxima of all transactions' QoS levels currently held in the link is forwarded by T-AXI link command which can override the individual transaction QoS value as AXI transactions leave the link.
A*VALID	-	Not required, indicated by T-AXI command.
A*READY	-	Not required, link status managed by T-AXI protocol
RID[3:0]	RID[9:0]	Same ID support as commands
RDATA[n:0]	RDATA[n:0]	Supports 32/64/128/256-bit data over any link widths.

RRESP[1:0]	-	Not required, indicated by T-AXI command
RLAST	RLAST	Supported
RVALID	RVALID	Gate off clock for idle cycles
RREADY	-	Not required, link status managed by T-AXI protocol
WID[3:0]	-	Write interleaving dropped in AXI4
WDATA[n:0]	RDATA[n:0]	Supports 32/64/128/256-bit data over any link widths.
WSTRB[3:0]	WSTRB[15/8:0]	Only used for AWWS and AWSHRTWS transactions that are issued when WSTRBs indicate a partial write.
WLAST	-	Not essential, so not supported
WVALID	-	Gate off clock for idle cycles
WREADY	-	Not required, link status managed by T-AXI protocol
BID[3:0]	BID[9:0]	Same ID support as commands
BRESP[1:0]	BRESP[1:0]	Carried in T-AXI command
BVALID	-	Not required, link status managed by T-AXI protocol
BREADY	-	Not required, link status managed by T-AXI protocol
CSYSREQ	-	Low power interface signalling not supported in T-AXI
CSYSACK	-	

3.3 Flow Control

Flow control of the Thin AXI is split into two parts: AXI data flow control and link control. Figure 4 shows the block diagram of Thin AXI link.

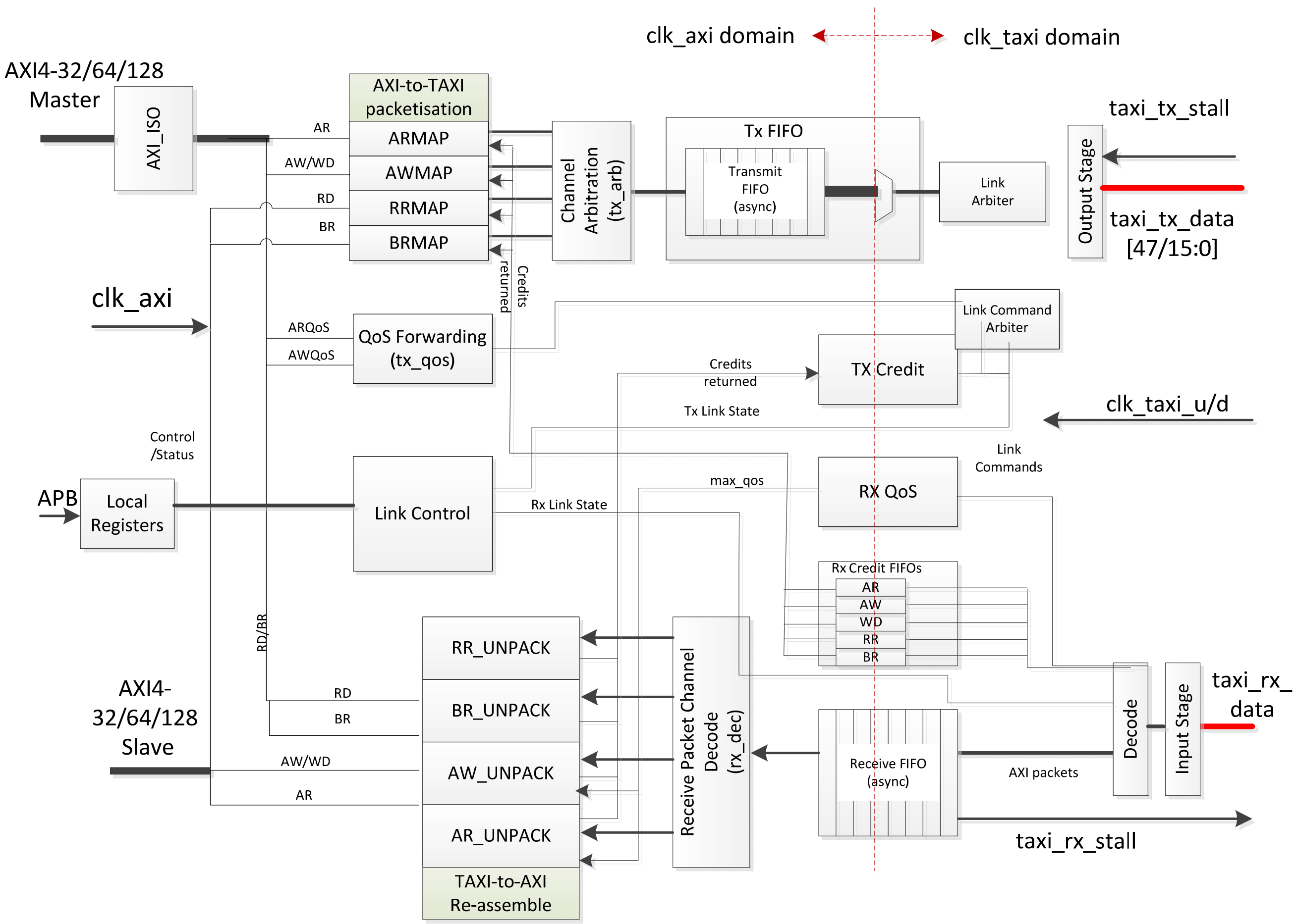


Figure 4: Block Diagram of Thin AXI Link interface

Link control is used to prevent the common receive FIFO overflowing and is simply a stall signal that tells the transmit-front-end to stop sending more data in cases when T-AXI/AXI clock ratios empty the receive FIFO at a slower rate than the rate at which they fill it. These stalls will clear quickly and not introduce headof-tree blocking, as there is nothing fundamentally blocking the flow; they are just due to data rate differences.

The AXI flow control is a credit-based mechanism that only allows the AXI command mappers to issue new T-AXI commands if they have sufficient credit. The mappers are given a starting credit corresponding to the amount of storage at the receive end of each of the AXI channels and they spend credit each time they issue a transfer that will use one of those storage words. A mapper cannot send any further data once it has spent all its credit. This prevents a channel stall at the receive end causing data from that channel filling the common receive FIFO and blocking all other AXI channels on the link (a condition that can cause a deadlock). Ensuring each channel has sufficient storage at the receive de-mapper means that the common receive FIFO cannot block. Credits are returned (via T-AXI link commands) each time the de-mapper clears a space in its local storage by issuing an AXI transaction.

3.4 Reset and Shutdown

T-AXI supports the ability to independently reset and power down the subsystems at either end of the link. For this purpose, the link provides software control (through APB registers) to reset the link and to disable it so that the subsystems at the two ends of the link are decoupled. Two hard reset inputs are provided that will fully asynchronously reset the registers in the T-AXI block. One has its rising edge synchronised to the AXI clock (rst_n) and the other to the T-AXI clock (rst_taxi_n). Rst_n is tied to the power-on reset and the subsystem reset. Rst_taxi_n is driven from the T-AXI LCPR located in the centre of the link. Link control commands (such as LINK_CTRL) are still supported when the link is in the shutdown state. The shutdown only refers to the AXI buses connected to the T-AXI link.

3.5 Power Control

There is limited support required within the T-AXI link for power control. It is essential that software uses the control and status registers in the T-AXI modules to disable the link cleanly before powering down.

3.6 Clock Control

The clock is generated from the T-AXI LCPR that is located in the centre of the link. This is a synchronous clock driven to both ends of the link in thick, wide metal using a single super-buffer in each direction. This ensures a very low skew synchronous clock available throughout the link. Each end of the link supplies an asynchronous clock request (taxi_u/d_clkreq) that it uses to request the clock from the T-AXI LCPR. The LCPR drives the clock to both ends of the link when either clock request is active. The two ends of the link asserts the clock request when they receive an AXI command and remove the request only when they have received the credits back for all transactions issued. The clock request can also be activated if a link command is required to be sent due to software intervention.

3.7 Security

The security scheme proposed for this SoC is a superset of ARM's trustzone scheme. The T-AXI link carries the AXI PROT signals that are used together with the masters' AXI ID to determine AXI permissions within the address map. Only two levels of security are required for the SoC; hence it was decided to carry only one bit of APROT across the T-AXI link in order to make space for additional ID bits. The choice of which APROT bit is carried, can be made at the AXI connection to the T-AXI instance but for the SoC this will be bit-1.

3.8 Quality of Service

QoS in AXI4 is transmitted per command, however this scheme fails in a distributed arbitration scheme when a high priority command is queued behind a low priority command with no opportunity to overtake it. The scheme for the infrastructure components in the SoC multimedia is for each infrastructure component to forward the highest QoS level of all its outstanding transfers. For the T-AXI link, it makes sense to separate the QoS communication from the AXI transfers and transmit the information of a separate link-control command (QoS). This allows changes in the QoS level (e.g. due to a master that requested data at a particular QoS level entering a higher state of panic) to be forwarded independently of the AXI transfer thus allowing downstream commands' QoS levels to be increased.

3.9 Interrupts

The suggestion to carry interrupts over T-AXI was examined but rejected as the benefit in wire saving is negligible compared to the cost due to increased complexity and the number of signal transitions that would need to occur just to signal an interrupt.

4.0 Test Bench

Since this is pre-ASIC DVT, the SoC is emulated on FPGAs. The FPGA systems are from Synopsys, configured with four Virtex-7 Xilinx FPGAs. Each FPGA box contains four FPGAs. An FPGA station is comprised of FPGA system, UMRBus controller, DSTREAM JTAG, PC and a remote power controller.

4.1 FPGA system

The simplest technique to deploy Thin AXI DDR is the use of IDDR and ODDR cells in the FPGA I/O pads, as shown in Figure 5 . One DDR cell is required per two signals, so a 16-bit T-AXI bus will require eight DDR cells in each direction for the data. In addition to the data, the source-synchronous clock and a valid/stall signal must be transferred in each direction. In total, a 16-bit T-AXI bus can be transferred between FPGAs with 20 I/Os (2 * (8 data + 1 stall/valid + 1 clock)).

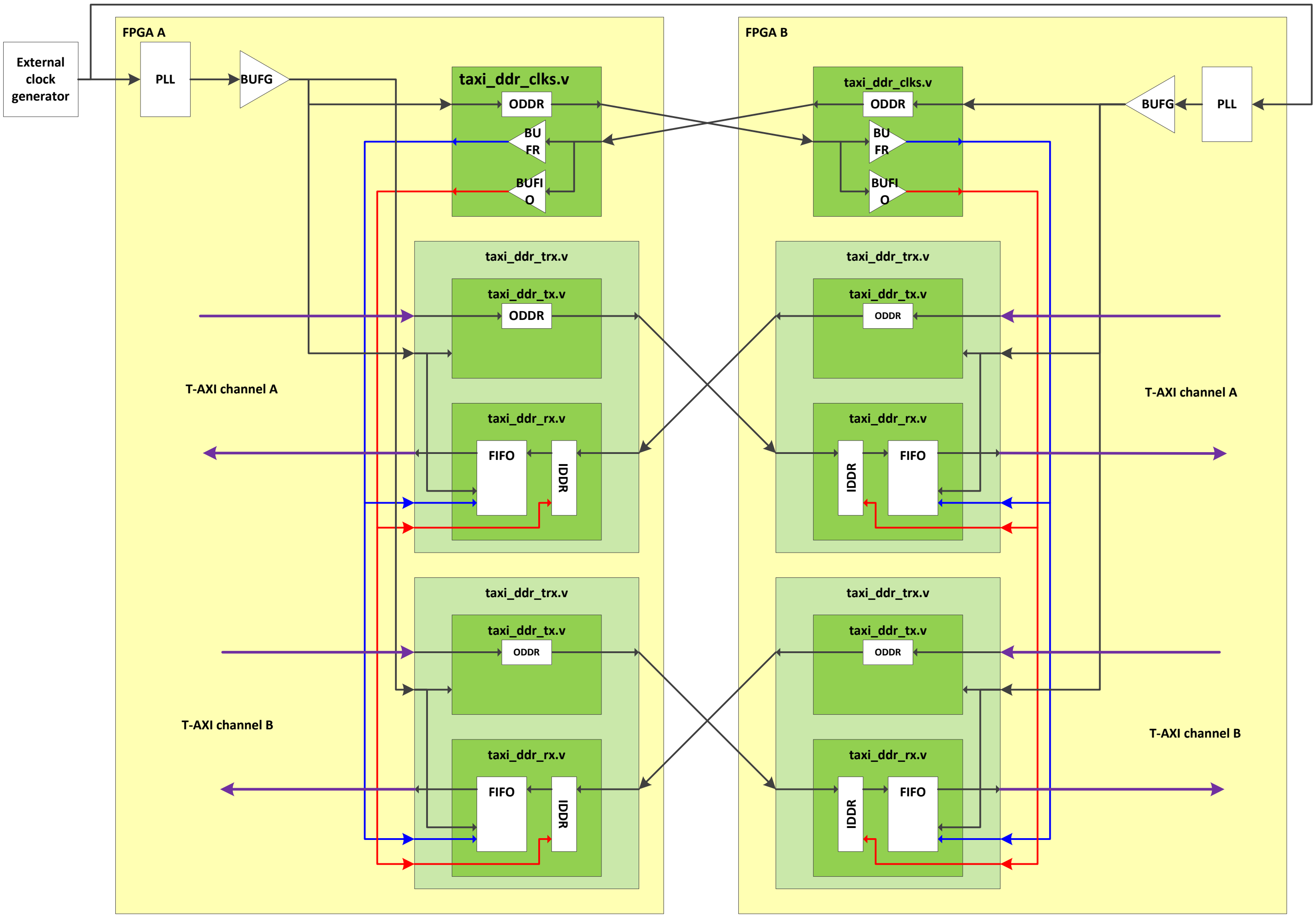


Figure 5: Thin AXI routing in FPGA

In Figure 5,

taxi_ddr_clk module:

- Takes the global clk_taxi within the FPGA and uses an ODDR cell to prepare the clock for output on an FPGA I/O pin.

- Receives a transferred clock from an FPGA pin and passes it through two Xilinx clock buffers (that is, one received clock is passed through these two separate buffers)
  - BUFIO: drives a local low-skew clock network to all I/O pads in the I/O bank and is used to drive the clock pins of all the IDDR cells in the receiver
  - BUFR: drives a regional clock network in the FPGA fabric and is used to drive the write side of the receiver FIFO

The received clock is passed through an IDELAY element to ensure that the inbound data stabilises before the sampling event (driven by the delayed clock) occurs. The IDELAY element requires that an IDELAYCTRL module is instantiated somewhere in each FPGA.

IDELAYCTRL module:

Due to the use of IDELAY primitives within taxi_ddr_clk, it is necessary to have an IDELAYCTRL module in each FPGA that instantiates taxi_ddr_clk. It is fed by a 200MHz clock. The instance have attribute "IDELAY_GROUP" associated with it, set to "taxi_ddr_rx". For Virtex 7 devices, the clock must be 200MHz ± 10MHz, and the RST must pulse high for a minimum of 52ns.

taxi_ddr_trx module:

Parameter D_WID is the width of the T-AXI data bus (must be 16, 32 or 48). The taxi_ddr_trx module wraps a taxi_ddr_tx module and a taxi_ddr_rx module as a pair. This ensures that the stalls travel in the appropriate direction, and also ensures that the stalls happen correctly. It takes three clocks:

- clk_taxi : the system-global T-AXI clock
- clk_io : the I/O clock received from the other end of the link by taxi_ddr_clk
- clk_r : the regional fabric clock received from the other end of the link by taxi_ddr_clk

taxi_ddr_tx module:

This module deals with the transmit side of the link. It organises outbound data into bit pairs and presents to ODDR cells for transmission. This module uses only the global clock.

taxi_ddr_rx module:

This module deals with the receive side of the link. Using clk_io, it receives the DDR data with IDDR cells. Using clk_r, this data is passed into a FIFO. This FIFO is used to manage both the domain-crossing into the global clk_taxi domain, and also to manage T-AXI stall cycles. The read-side of the FIFO is in the clk_taxi domain.

4.2 UMRbus controller

The HAPS UMRBus (Universal Multi-Resource Bus) Interface kit is a complete and reliable set of components that allow bi-directional data exchange (at runtime) between software (C/C++ or Tcl/TK applications) and hardware DUT (Device Under Test).

4.3 DStream JTAG and a host PC

ARM DStream is a debug and trace tool that facilitates powerful software debug and optimization on any ARM processor-based hardware target. It uses eclipse configured to work with DS-5 and connects the PC to the FPGA over USB connection. A connection is created using Realview as shown in Figure 6 and the cores are enabled as shown in Figure 7.

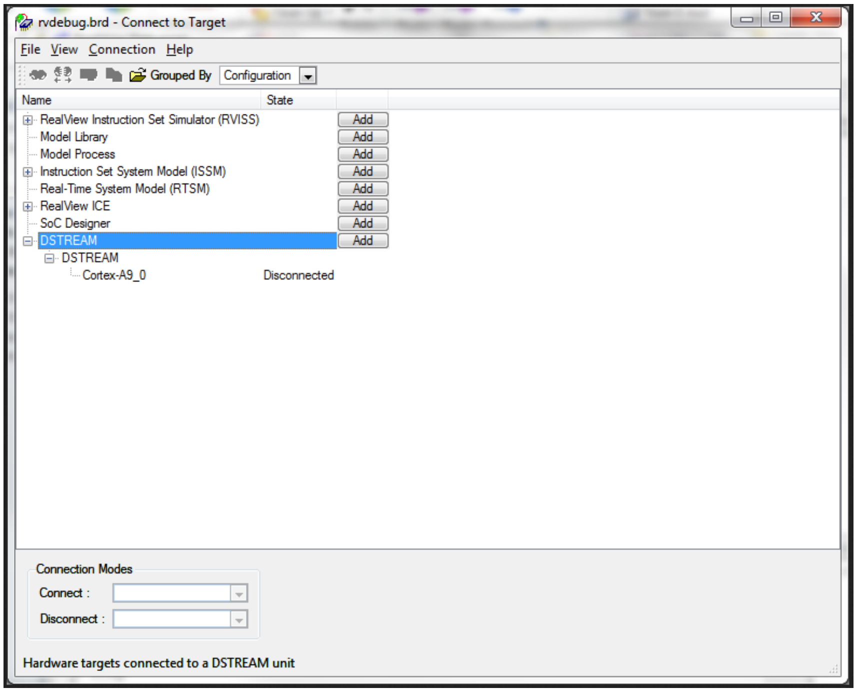


Figure 6: Connecting DStream to FPGA

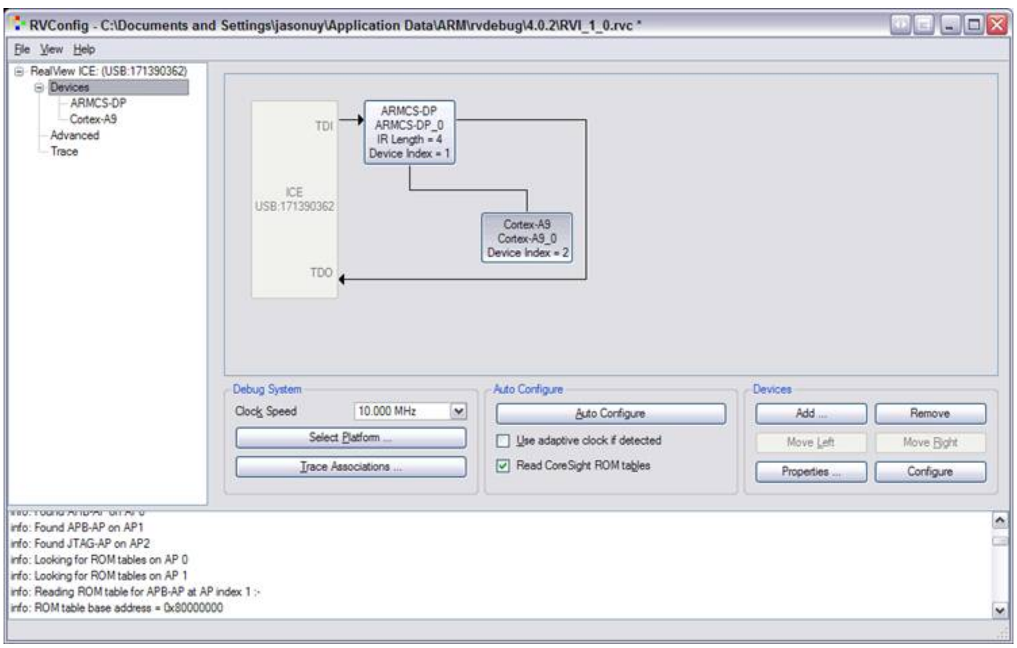


Figure 7: Configure SoC cores using ARM RVI

4.4 A remote power controller

Each FPGA station is on its own electrical circuit and protected by a 15A breaker. In the event an FPGA system or PC locks-up, power must be cycled to get the equipment operational again. To allow for better remote access, each FPGA station has been equipped with a remote power controller so a user can control the power to a station remotely through a web interface.

5.0 Test Cases and Results

The T-AXI link is implemented as two individual modules: taxi_upstream (subsystems to memory) and taxi_downstream (memory to subsystems). All blocks below these top-level structural modules are common between the two and are managed by parameters. There are a number of parameters at the top level of the T-AXI upstream and downstream modules that allow the link to be tuned for area, power and performance and to set its AXI connectivity configuration. Some of these parameters need to be consistent at both ends of the link whereas others are set to tune the operation of that end of the link only. A description of how to use each parameter is provided in Broadcom's RDB (Register Data Base). As an example, one of the T-AXI module's registers are given in Figure 8. This is part of the RDB and defines the memory locations, register names and their description.

Offset (8-bit space)	Flags	Register Name	Description
0x15034000		SLVSYS_TAXI_TAXI_CTRL	TAXI_control
0x15034004		SLVSYS_TAXI_TAXI_STATUS	TAXI_status
0x1503400c		SLVSYS_TAXI_TAXI_AR_CREDIT_CTRL	TAXI_AR_CREDIT_CTRL
0x15034010		SLVSYS_TAXI_TAXI_AW_CREDIT_CTRL	TAXI_AW_CREDIT_CTRL
0x15034014		SLVSYS_TAXI_TAXI_RR_CREDIT_CTRL	TAXI_RR_CREDIT_CTRL
0x15034018		SLVSYS_TAXI_TAXI_BR_CREDIT_CTRL	TAXI_BR_CREDIT_CTRL
0x15034020		SLVSYS_TAXI_TAXI_PARAM_STATUS_0	TAXI_PARAM_STATUS_0
0x15034024		SLVSYS_TAXI_TAXI_PARAM_STATUS_1	TAXI_PARAM_STATUS_1
0x15034028		SLVSYS_TAXI_TAXI_PARAM_STATUS_2	TAXI_PARAM_STATUS_2
0x1503402c		SLVSYS_TAXI_TAXI_PARAM_STATUS_3	TAXI_PARAM_STATUS_3
0x1503407c		SLVSYS_TAXI_TAXI_ID	TAXI_ID

Figure 8: RDB of SLVSYS Subsystem T-AXI

The test cases are written in C language and use a Broadcom's OS-less (Operating-System-less) infrastructure to implement the code. Figure 9 shows a list of all the test cases that were implemented to fully test and characterize a T-AXI Link.

```
Console 'taxi' Commands
-----------------------
dump <port_num> ......................... Dump port registers
access .................................. Basic register access test
enDis <port_num> ........................ Test TAXI Enable/Disable
enable <port_num> ....................... Enable TAXI port
disable <port_num> ...................... Disable TAXI port
set_max_credit <port_num> ............... Sets maximum credit
get_mstr_link <port_num> ................ Get the Master Control Link End
set_mstr_link <port_num><mst/fab> ...... Set MFAB or MST as Master Control Link End
status <port_num> ....................... Test status bits
qos <port_num> .......................... Test QoS Forwarding
clk_gating <port_num> ................... Test Clock Gating en/Disable
reset <port_num> ........................ Test link reset
wd <port_num> ........................... Test WD credit performance by changing credits
rr <port_num> ........................... Test RR credit performance by changing credits
br <port_num> ........................... Test BR credit performance by changing credits
ar <port_num> ........................... Test AR credit performance by changing credits
aw <port_num> ........................... Test AW credit performance by changing credits
ext_buff <port_num> ..................... Test external buffer of credit cntrl
lockup <port_num> ....................... Test link lock up scenario
```

Figure 9: ScreenShot of T-AXI Test Cases written in C

In Figure 9, each of the commands on the left hand side can be executed as shown in Figure 10. Figure 10 shows the register dump of a T-AXI Link. The address field shows the memory location of the Link Registers, Value_Read is the current value in the Register (these values are explained in Table 3 and Table 4), Default_Value is the register value when the link is turned on or reset.

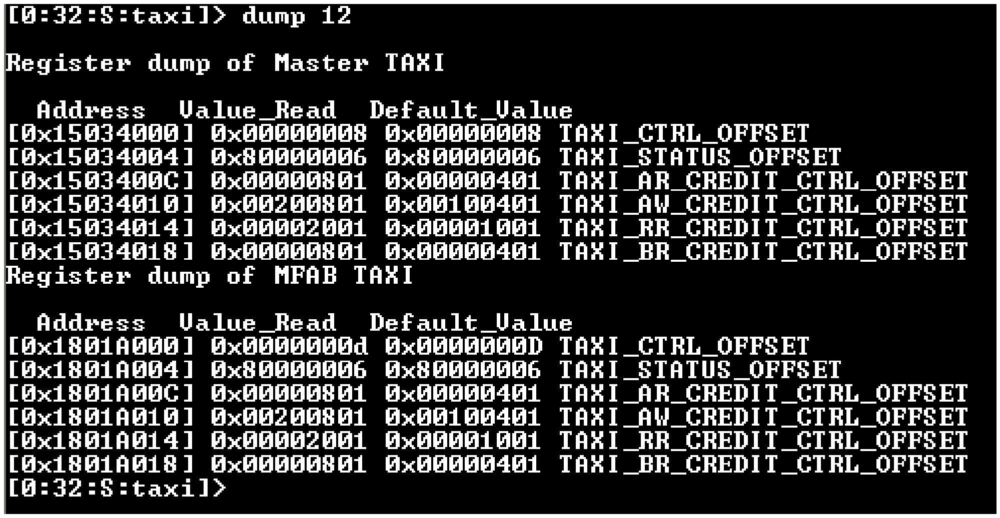


Figure 10: Register dump of a Master and Slave T-AXI Link

5.1 Control and Status Registers of T-AXI Link

There are a number of essential control/status register bits to use and understand when managing the T-AXI link. Table 3 gives a glimpse of Control register of T-AXI Link. Bits 09:31 are reserved and must be written with 0. Table 4 gives an overview of Status Register bits. The bits not indicated in the table are reserved bits.

Table 3: Control Register of T-AXI Link

Field Name	Bit Field	Description
DisableTaxiClkGate	08	Disables the clock gating used on the T-AXI link. Will result in higher power but can reduce latency for the first transfers after the clock was gated.
ForceLinkCtrlState	07:05	Forces the link_state control state machine to a specific state. Only for use in lock-up scenarios.
ForceLinkCtrl	04	Forces the link_state control state machine to a specific state. Only for use in lock-up scenarios. State forced is set by ForceLinkCtrlState.
QosForwardEnable	03	Enables QoS forwarding, i.e taking the QoS values of all transactions in the link and forwarding the maximum value to both the awqos and arqos outputs from the receiver.
LinkCtrlMaster	02	Enables this end of the link to be the master for link control, i.e. only shutdown/reset changes from this end of the link will be observed. Setting this bit at both ends of the link may result in lock-up. If it is desired to change the control from one end of the link to the other then this should be done when the link is enabled and active.
Reset	01	Resets the link. Prevents any new AR/AW AXI transactions being accepted and issues a reset command to the other end of the link to tell it to do likewise. Waits for outstanding transactions to complete then issues a synchronous reset to all control logic and forces a reload of credit starting values. Reset is released automatically without needing to re-write this register.
Enable	00	Set to enable the T-AXI link. Resets to disabled to ensure low-power after start-up When cleared, the T-AXI link will stop taking any new AXI read/write commands and attempt to complete any in-flight transactions.

Table 4: Status Register of T-AXI Link

Field Name	Bit Field	Description
RemoteStatusActive	31	When set, the remote end of the link is currently in the active state. Software checks that this bit is set prior to enabling traffic into the link.
RemoteStatusShutdown	30	When set, the remote end of the link is currently in the shutdown state. Any AXI commands issued into the remote end of the link is swallowed by a dummy responder when in this state and likely results in instabilities in the system
OustandingWriteCount	28:21	Number of write transactions currently issued but not yet completed.
OustandingReadCount	20:13	Number of read transactions currently issued but not yet completed.
DummyAccessed	08	This bit is set whenever the dummy slave is accessed and cleared by writing. The dummy slave should never be accessed in normal operation so if this bit has been set then the link has either been shut down prematurely or accesses started before the link was enabled.
LinkState	03:01	Current state of link control state machine. Only included to assist with debug in a potential lock-up scenario. This state machine is used to safely manage the reset/shutdown/enable of the link. Resets can be issued transparently while the link is active.
Idle	00	Indicates no outstanding transactions and that the link is ready for reset/power-down

5.1.1 Test Case – Link Enable/Disable and Dummy Slave

Whenever a link is disabled, a dummy AXI slave is switched into the bus, so that transactions sent to the bus are completed and the bus does not lock up. Any transaction issued while the link is disabled is swallowed by the dummy slave. A write command/data is swallowed and ignored and a BRESP is issued. A read command/data is returned with a random dummy data. Software should take care that the bus is idle when enabling/disabling the link to ensure that transactions to not enter dummy slave. Dummy slave is tested on SoC's UART (Subsystem) using the following steps:

1. T-AXI link connecting UART to memory is enabled. Link "enable" is verified by reading and writing data to the memory.
2. T-AXI link connecting UART to memory is disabled. The console gets stuck at this point because dummy slave turns on. Since UART constantly reads data/commands from the console, dummy slave supplies random data which causes system hang due to software not being able to handle garbage data (See Figure 11)

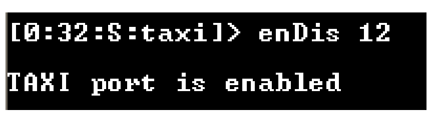
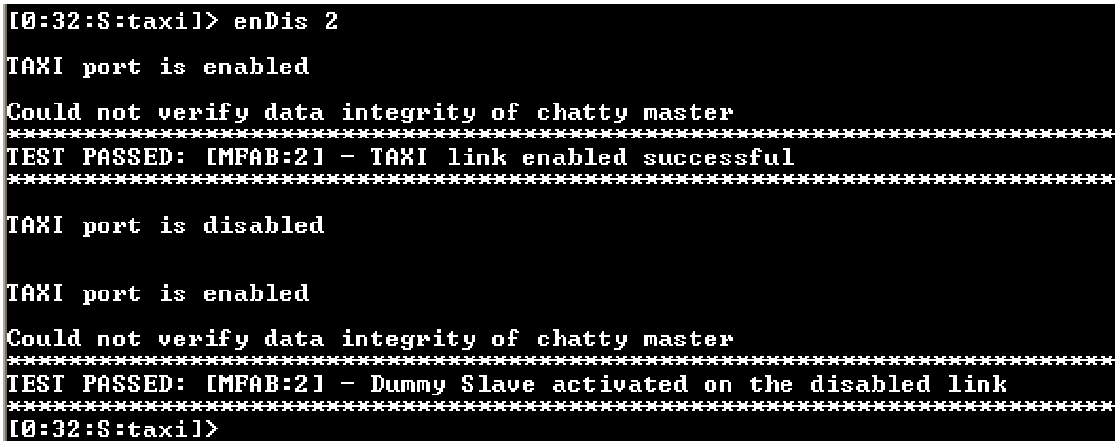


Figure 11: Enable/Disable UART T-AXI Link with dummy slave

3. A proper T-AXI "disable" functionality is verified using another port. The port was enabled and verified as given in step 1. Then the port was disabled. The verification of disabled port is provided in step 2 (which is

dummy slave activation). Figure 12 shows the test completion screen shot.



```
[0:32:S:taxi]> enDis 2

TAXI port is enabled

Could not verify data integrity of chatty master
********************************************************************************
TEST PASSED: [MFAB:2] - TAXI link enabled successful
********************************************************************************

TAXI port is disabled


TAXI port is enabled

Could not verify data integrity of chatty master
********************************************************************************
TEST PASSED: [MFAB:2] - Dummy Slave activated on the disabled link
********************************************************************************
[0:32:S:taxi]>
```

Figure 12: Enable/Disable of T-AXI Link

5.1.2 Test Case – Link Shutdown

Link shutdown requires more careful management by software as the dummy slave is switched on when the link is disabled; hence, transactions can get lost if the correct sequence is not followed. The logic in the T-AXI hardware can ensure that clean transitions are achieved between the enabled and disabled states but it has no way of knowing when a subsystem has finished using the link so it cannot wait for a specific event to tell it to switch the link off. This is the responsibility of software. The routine for powering down a block connected to the rest of the chip via T-AXI is:

1. Instruct block to go idle. This is block-specific but generally, it tells the block to complete all necessary housekeeping and then stop generating any new AXI transactions. The block provides status information (Idle bit

goes high) to the host CPU when it has reached its quiescent state so that the host knows when it can move to step 2.

2. Disable the T-AXI link. This puts the T-AXI link into its quiescent state and switch on dummy slaves at the command-receiving AXI interfaces. To complete transactions over the link, the AXI clock in the subsystem being shutdown, is kept running until the T-AXI link enters the disabled state. Once the link is disabled, it is clearly not possible for the host to use the path via T-AXI to write to a register to switch off the clocks. Disabling clocks is done through the clock manager, which is accessed using ARM JTAG.
3. Power down the subsystem.
4. Power down the T-AXI link. The dummy slave component lies outside of the T-AXI power domain and remains enabled when the T-AXI link is powered down.

```
[0:32:S:taxi]> shutdown 2

Chatty Master is idle. Idle bit = 1.
TAXI port is disabled

Could not verify data integrity of chatty master
******************************************************************************
TEST PASSED: [MFAB:2] - Dummy Slave activated on the disabled link
******************************************************************************
******************************************************************************
TEST PASSED: [MFAB:2] - TAXI link shutdown successful
******************************************************************************
```

Figure 13: Test Result of Link Shutdown

5.1.3 Test Case – Link Reset

Link reset requires waiting for the outstanding transactions to complete before resetting. The sequence that Link Reset Test case follows is:

1. Hold off new A*READY to prevent new transactions entering the link.
2. Wait for outstanding transactions to complete and all outstanding credits to be returned. Check idle status bit = 1 and OutstandingWriteCount = OutstandingReadCount = 0.
3. When both ends are idle, issue a synchronous reset to the control logic at both ends of the link.
4. Return to the active state. Verify if any data is lost during reset by reading system memory for any corruptions.

Figure 14: Test case results of Link Reset

5.1.4 Test Case – Master Link Control

The T-AXI link has the facility for the control registers at either end of the link to be used for link management, however only one end of the link can be used. The software can select either the upstream or downstream end to be the link master by setting the LinkCtrlMaster bit in the TAXI_CTRL register at the desired end of the link. If this bit is not set then the enable/reset bits of the control is ignored. It is possible to transfer the link master to be the other end of the link. The swap Link test case follows the sequence below:

1. Disable the link and wait for the status register to indicate that the link is fully shut down (Used Test cases 5.2 and 5.3).
2. Clear the LinkCtrlMaster bit at the end of the link that is currently the assigned link master.
3. Set the LinkCtrlMaster bit at the other end of the link.
4. Enable the link (using the control register at the new link master).

The Link Master will typically be assigned to the upstream end of the link as the control registers at the upstream end are typically accessible to the CPU without needing the link to be enabled.

Figure 15: Test Result of Swap Master Link

Both ends of the T-AXI link contain a link control state machine that it uses to safely transition between link states such as active, disabled and reset. The state machines at the two ends of the link need to track each other to ensure safe transitions with the link master leading the state transitions and the other end following its lead. The LinkState bit from Table 4 can have the following state decoding:

0x0 LINK_DISABLED - Link is shutdown, dummy slave enabled

0x1 LINK_WAIT_DS_IDLE - Wait for dummy slave to go idle and go ready

0x2 LINK_READY - Link is ready but not enabled.

0x3 LINK_ACTIVE - Link active and ready to take transfers

0x4 LINK_WAIT_IDLE - Wait for existing transfers to clear then go to idle

0x5 LINK_IDLE - Wait for remote to go idle then go to reset

0x6 LINK_RESET - Reset issued, then go to reset_clear

0x7 LINK_RESET_CLEAR - Reset cleared, go to either disabled or ready

Figure 16: Link State Machine

5.1.5 Test Case – Clock Gating

Clock gating saves power by adding more logic to a circuit to prune the clock tree. Pruning the clock disables portions of the circuitry so that the flip-flops in them do not have to switch states. Switching states consumes power. When not being switched, the switching power consumption goes to zero, and only leakage currents are incurred. This test measures the read and write latency before and after disabling clock gating in T-AXI Link. From Figure 17, before disabling clock gating, the number of CPU ticks required to complete read and write test is 12624. After clock gating is disabled, the number of CPU ticks required to complete read and write test is 12620. This clearly shows that disabling clock gating can reduce latency for the first transfers after the clock was gated.

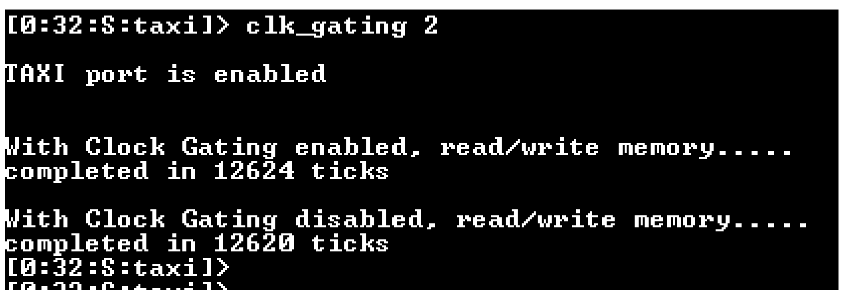


Figure 17: Test result of Clock gating.

5.1.6 Test Case – Status Register bits

The functionality of status bits was tested using various read and write cases. As shown in Figure 18, when a T-AXI Link is activated the RemoteStatusActive bit is high at the subsystem side of the T-AXI as well as at the Main Fabric side of the T-AXI. The LinkState is 0x3 which means link is active and ready to take transfers. When the write traffic is initiated, OutStandingWriteCount bits give out the number of outstanding writes. In Figure 18, there are eight outstanding writes. After the writes complete, the link comes back to the initial state. When the reads start, OutStandingReadCount bits give out the number of outstanding reads. In Figure 18, there are four outstanding reads. After the reads complete, the link comes back to the initial state.

```
[0:32:S:taxi]> status 2

TAXI port is enabled


Before WRITE traffic starts:
Register Name              MFAB          SUBSYSTEM
--------------------------------------------------
RemoteStatusActive:         1.             1
IdleStatus:                 0.             0
OutstandingReadCount:       0.             0
OutstandingWriteCount:      0.             0
LinkState:                  3.             3
RemoteStatusShutdown:       0.             0
WrDataMismatch:             0.             0
IllegalRemoteState:         0.             0
IllegalTaxiCmd:             0.             0
IllegalAxiCmd:              0.             0

During WRITE traffic:
Register Name              MFAB          SUBSYSTEM
--------------------------------------------------
RemoteStatusActive:         1.             1
IdleStatus:                 0.             0
OutstandingReadCount:       0.             0
OutstandingWriteCount:      0.             8
LinkState:                  3.             3
RemoteStatusShutdown:       0.             0
WrDataMismatch:             0.             0
IllegalRemoteState:         0.             0
IllegalTaxiCmd:             0.             0
IllegalAxiCmd:              0.             0

After WRITE traffic stops:
Register Name              MFAB          SUBSYSTEM
--------------------------------------------------
RemoteStatusActive:         1.             1
IdleStatus:                 0.             0
OutstandingReadCount:       0.             0
OutstandingWriteCount:      0.             0
LinkState:                  3.             3
RemoteStatusShutdown:       0.             0
WrDataMismatch:             0.             0
IllegalRemoteState:         0.             0
IllegalTaxiCmd:             0.             0
IllegalAxiCmd:              0.             0

Before READ traffic starts:
Register Name              MFAB          SUBSYSTEM
--------------------------------------------------
RemoteStatusActive:         1.             1
IdleStatus:                 0.             0
OutstandingReadCount:       0.             0
OutstandingWriteCount:      0.             0
LinkState:                  3.             3
RemoteStatusShutdown:       0.             0
WrDataMismatch:             0.             0
IllegalRemoteState:         0.             0
IllegalTaxiCmd:             0.             0
IllegalAxiCmd:              0.             0

During READ traffic:
Register Name              MFAB          SUBSYSTEM
--------------------------------------------------
RemoteStatusActive:         1.             1
IdleStatus:                 0.             0
OutstandingReadCount:       0.             4
OutstandingWriteCount:      0.             0
LinkState:                  3.             3
RemoteStatusShutdown:       0.             0
WrDataMismatch:             0.             0
IllegalRemoteState:         0.             0
IllegalTaxiCmd:             0.             0
IllegalAxiCmd:              0.             0
```

Figure 18: Test results of Status bits on reads and writes

5.2 Credit and Parameters Registers of T-AXI Link

The credit registers carry a payload that gives the number of words transferred from the link on each AXI channel from the available buffer space in the receiver. Transmit can increase its available credit by the value carried. There are various credit registers in the T-AXI Link and are tested in the following sections. Also, there are four parameter registers which carry connectivity configuration parameters of the T-AXI Links.

5.2.1 Parameter Read-Only Registers

The Parameter registers are read-only and are assigned the build values of the Link. These values are used to configure credit registers.

5.2.1.1 CREDIT_DWIDTH Register

Credit_dwidth register contains the width of the buses that are used to carry credit tokens across the link. Credits are accumulated in the receivers at both ends of the link each time the corresponding AXI channel completes a phase. They are passed back to the transmitter which is only able to send new commands/data if it has sufficient credit. This bus width is set at the minimum needed to avoid an overflow in the counter.

5.2.1.2 RXFIFO_AWIDTH Register

This register contains the address width of the receive FIFO. This FIFO is sized to minimize stalls on the T-AXI link while remaining area efficient. It is currently sized at 32 words i.e RXFIFO_AWIDTH=5. The second factor in the size of this

FIFO is that it is read in much larger data widths than it is written - data is read out on the AXI clock and is shifted in much larger data quantities in this domain per clock compared to the taxi data width. For example, an AXI write involves transferring an AW command (~60-bits of data) plus 144 bits of write data and strobes on a single clock, so we need to read 12 words from the FIFO for a 3GB/s link. In addition, the receiver has the capability to read data for up to two channels per clock (e.g. a write command/data plus a read command) so the FIFO must be sized to accommodate this, which is variable with TAXI_DWIDTH.

5.2.1.3 TX_AWIDTH Register

This parameter sets the number of address bits needed for the transmit FIFO. This is a wide/shallow FIFO used to cross from the AXI clock domain to the T-AXI clock domain, so, adding more address bits is expensive (e.g. FIFO is 244-bits wide in downstream for a 128/32-bit memory/peripheral bus). A depth of 4 (i.e. TX_AWIDTH=2) has been shown to produce very few unnecessary stalls so this register is set to a value of 2.

5.2.1.4 RX_ARWIDTH Register

This register contains the size of the storage built at the receive end of the read command channel and determines the number of outstanding transactions that the receiver can take (and not pass down the bus) before the transmitter will run out of credit and stall.

5.2.2 Credit Control Registers

The link resets with a small starting credit assigned to each AXI channel in order to enable the link to function correctly, however this is typically smaller than the maximum the link can support. Setting the credit values to their maximum value will increase the performance of the link by allowing it to support more outstanding transactions.

The credit values assigned to the AR/AW/RR/BR_CREDIT_CTRL registers should match the sizes of the corresponding receive FIFOs at the far end of the link so that the transmit ends of the link can issue enough outstanding transactions to fill these FIFOs but no more. The software determines the size of these FIFOs by reading the Parameter registers for the two ends of the link. Since the values read from parameter registers are the FIFO address widths rather than the actual FIFO depths, they are adjusted to match the actual FIFO depth to determine the maximum credit allocation, for example:

AR_Credit (upstream) = 1 << RXAR_FIFO_AWIDTH (downstream)

AW_Credit (upstream) = 1 << RXAW_FIFO_AWIDTH (downstream)

WD_Credit (upstream) = 1 << RXWD_FIFO_DWIDTH (downstream)

RR_Credit (upstream) = 1 << RXRR_FIFO_DWIDTH (downstream)

BR_Credit (upstream) = 1 << RXBR_FIFO_DWIDTH (downstream)

Once the new credit values have been programmed, the link is reset to update the new credit values. To test the credit registers, a DMA subsystem was used to read and write to the memory using T-AXI Link. The maximum bandwidth supported by DMA port is 20MB/s:

- FPGA clock was running at 10MHz,
- DMA does 1 read and 1 write per transaction,
- Therefore, DMA peak bandwidth = 10MHz * 2Bytes = 20MB/s.

On the other hand, maximum bandwidth supported by memory controller is 40MB/s:

- FPGA clock was running at 10MHz,
- Memory controller lane is 32bit wide, therefore supporting 4Bytes
- So, Memory peak bandwidth = 10MHz * 4Bytes = 40MB/s.

In the test, the credit control registers were varied from their minimum to maximum credit values and at each credit value, DMA generated traffic and system bandwidth was measured. Figure 19 through to Figure 23 show the bandwidth results. Some terminology used in the tests is:

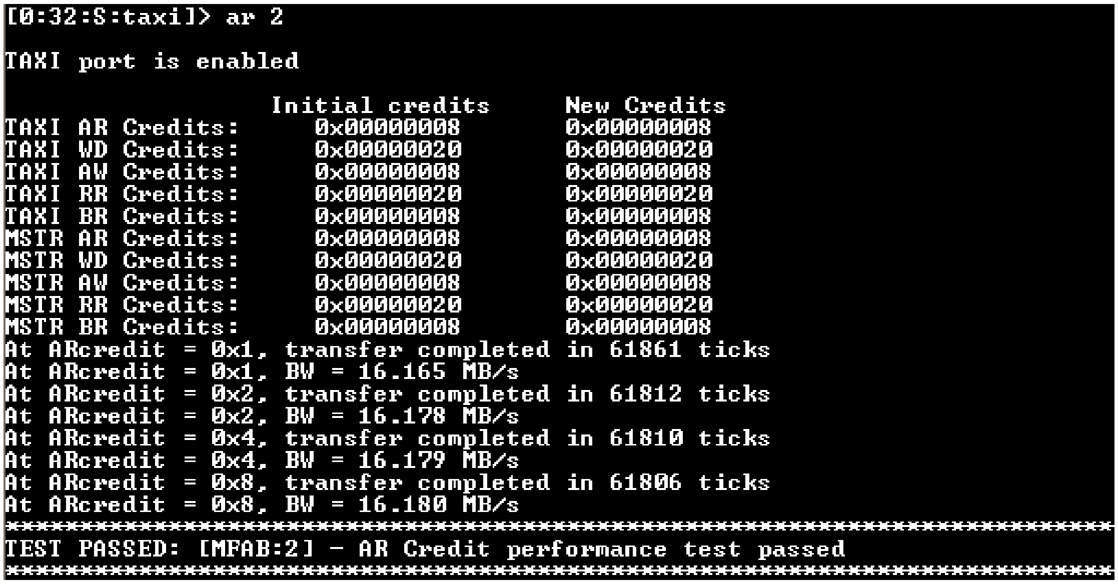
AR = Address Read

AW = Address Write

WD = Write Data

RR = Read Return, acknowledgement to the read data

BR = BResp, acknowledgement to the write data.

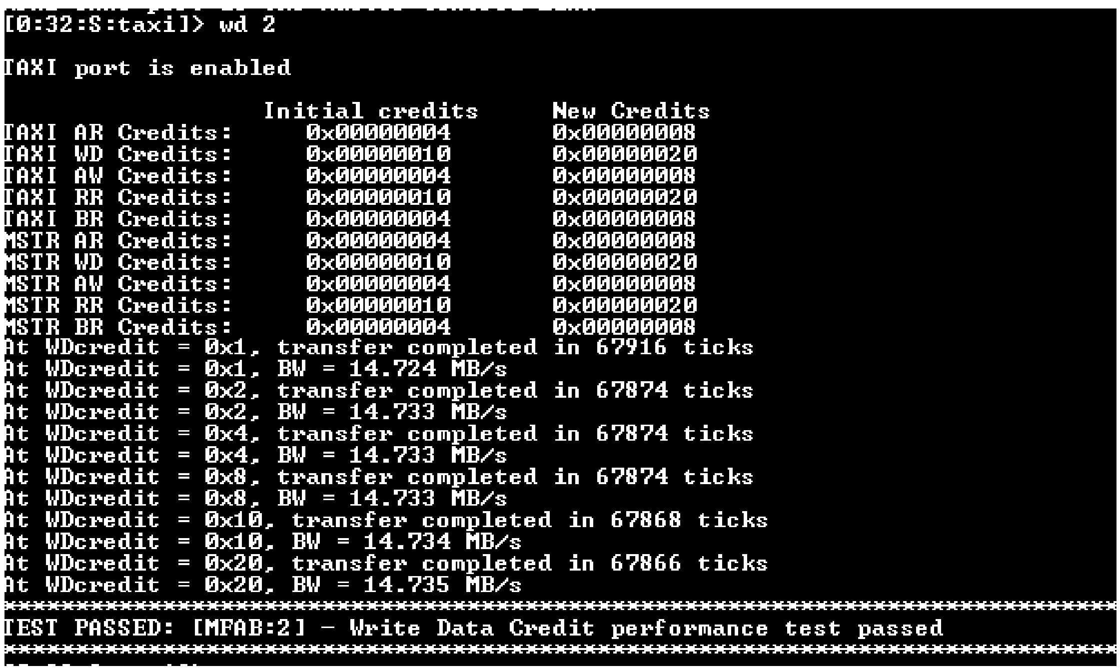


```
[0:32:S:taxi]> ar 2

TAXI port is enabled

                   Initial credits     New Credits
TAXI AR Credits:      0x00000008       0x00000008
TAXI WD Credits:      0x00000020       0x00000020
TAXI AW Credits:      0x00000008       0x00000008
TAXI RR Credits:      0x00000020       0x00000020
TAXI BR Credits:      0x00000008       0x00000008
MSTR AR Credits:      0x00000008       0x00000008
MSTR WD Credits:      0x00000020       0x00000020
MSTR AW Credits:      0x00000008       0x00000008
MSTR RR Credits:      0x00000020       0x00000020
MSTR BR Credits:      0x00000008       0x00000008
At ARcredit = 0x1, transfer completed in 61861 ticks
At ARcredit = 0x1, BW = 16.165 MB/s
At ARcredit = 0x2, transfer completed in 61812 ticks
At ARcredit = 0x2, BW = 16.178 MB/s
At ARcredit = 0x4, transfer completed in 61810 ticks
At ARcredit = 0x4, BW = 16.179 MB/s
At ARcredit = 0x8, transfer completed in 61806 ticks
At ARcredit = 0x8, BW = 16.180 MB/s
******************************************************************************
TEST PASSED: [MFAB:2] - AR Credit performance test passed
******************************************************************************
```

Figure 19: Address Read Credit values vs bandwidth observed



```
[0:32:S:taxi]> wd 2

TAXI port is enabled

                   Initial credits     New Credits
TAXI AR Credits:      0x00000004       0x00000008
TAXI WD Credits:      0x00000010       0x00000020
TAXI AW Credits:      0x00000004       0x00000008
TAXI RR Credits:      0x00000010       0x00000020
TAXI BR Credits:      0x00000004       0x00000008
MSTR AR Credits:      0x00000004       0x00000008
MSTR WD Credits:      0x00000010       0x00000020
MSTR AW Credits:      0x00000004       0x00000008
MSTR RR Credits:      0x00000010       0x00000020
MSTR BR Credits:      0x00000004       0x00000008
At WDcredit = 0x1, transfer completed in 67916 ticks
At WDcredit = 0x1, BW = 14.724 MB/s
At WDcredit = 0x2, transfer completed in 67874 ticks
At WDcredit = 0x2, BW = 14.733 MB/s
At WDcredit = 0x4, transfer completed in 67874 ticks
At WDcredit = 0x4, BW = 14.733 MB/s
At WDcredit = 0x8, transfer completed in 67874 ticks
At WDcredit = 0x8, BW = 14.733 MB/s
At WDcredit = 0x10, transfer completed in 67868 ticks
At WDcredit = 0x10, BW = 14.734 MB/s
At WDcredit = 0x20, transfer completed in 67866 ticks
At WDcredit = 0x20, BW = 14.735 MB/s
******************************************************************************
TEST PASSED: [MFAB:2] - Write Data Credit performance test passed
******************************************************************************
```

Figure 20: Write Data Credit values vs bandwidth observed

```
[0:32:S:taxi]> aw 2

TAXI port is enabled

                    Initial credits      New Credits
TAXI AR Credits:       0x00000008        0x00000008
TAXI WD Credits:       0x00000020        0x00000020
TAXI AW Credits:       0x00000008        0x00000008
TAXI RR Credits:       0x00000020        0x00000020
TAXI BR Credits:       0x00000008        0x00000008
MSTR AR Credits:       0x00000008        0x00000008
MSTR WD Credits:       0x00000020        0x00000020
MSTR AW Credits:       0x00000008        0x00000008
MSTR RR Credits:       0x00000020        0x00000020
MSTR BR Credits:       0x00000008        0x00000008
At AWcredit = 0x1, transfer completed in 67925 ticks
At AWcredit = 0x1, BW = 14.722 MB/s
At AWcredit = 0x2, transfer completed in 67868 ticks
At AWcredit = 0x2, BW = 14.734 MB/s
At AWcredit = 0x4, transfer completed in 67867 ticks
At AWcredit = 0x4, BW = 14.735 MB/s
At AWcredit = 0x8, transfer completed in 67864 ticks
At AWcredit = 0x8, BW = 14.735 MB/s
*******************************************************************
TEST PASSED: [MFAB:2] - AW Credit performance test passed
*******************************************************************
```

Figure 21: Address Write Credit values vs bandwidth observed

```
[0:32:S:taxi]> rr 2

TAXI port is enabled

                    Initial credits      New Credits
TAXI AR Credits:       0x00000008        0x00000008
TAXI WD Credits:       0x00000020        0x00000020
TAXI AW Credits:       0x00000008        0x00000008
TAXI RR Credits:       0x00000020        0x00000020
TAXI BR Credits:       0x00000008        0x00000008
MSTR AR Credits:       0x00000008        0x00000008
MSTR WD Credits:       0x00000020        0x00000020
MSTR AW Credits:       0x00000008        0x00000008
MSTR RR Credits:       0x00000020        0x00000020
MSTR BR Credits:       0x00000008        0x00000008
At RRcredit = 0x1, transfer completed in 479904 ticks
At RRcredit = 0x1, BW = 2.084 MB/s
At RRcredit = 0x2, transfer completed in 243530 ticks
At RRcredit = 0x2, BW = 4.106 MB/s
At RRcredit = 0x4, transfer completed in 125849 ticks
At RRcredit = 0x4, BW = 7.946 MB/s
At RRcredit = 0x8, transfer completed in 63651 ticks
At RRcredit = 0x8, BW = 15.711 MB/s
At RRcredit = 0x10, transfer completed in 62438 ticks
At RRcredit = 0x10, BW = 16.016 MB/s
At RRcredit = 0x20, transfer completed in 62452 ticks
At RRcredit = 0x20, BW = 16.012 MB/s
*******************************************************************
TEST PASSED: [MFAB:2] - Read Data Credit performance test passed
*******************************************************************
```

Figure 22: Read Response Credit values vs bandwidth observed

```
[0:32:S:taxi]> br 2

TAXI port is enabled

                     Initial credits      New Credits
TAXI AR Credits:        0x00000008        0x00000008
TAXI WD Credits:        0x00000020        0x00000020
TAXI AW Credits:        0x00000008        0x00000008
TAXI RR Credits:        0x00000020        0x00000020
TAXI BR Credits:        0x00000008        0x00000008
MSTR AR Credits:        0x00000008        0x00000008
MSTR WD Credits:        0x00000020        0x00000020
MSTR AW Credits:        0x00000008        0x00000008
MSTR RR Credits:        0x00000020        0x00000020
MSTR BR Credits:        0x00000008        0x00000008
At BRcredit = 0x1, transfer completed in 173225 ticks
At BRcredit = 0x1, BW = 5.773 MB/s
At BRcredit = 0x2, transfer completed in 119979 ticks
At BRcredit = 0x2, BW = 8.335 MB/s
At BRcredit = 0x4, transfer completed in 93351 ticks
At BRcredit = 0x4, BW = 10.712 MB/s
At BRcredit = 0x8, transfer completed in 67868 ticks
At BRcredit = 0x8, BW = 14.734 MB/s
*************************************************************************
TEST PASSED: [MFAB:2] - BR Credit performance test passed
*************************************************************************
```

Figure 23: BResp Credit Values vs bandwidth observed

From Figure 19 through to Figure 23, it can be noticed that the bandwidth did not get affected by the changes in the credit values of AR, AW and WD; whereas, the bandwidths experienced a significant change in the credit values of BR and RR. This is because these credit registers affect the downstream traffic (i.e. traffic generated towards the subsystem). Since RR and BR traffic is sent from memory to the subsystem, a significant change is observed. Another test was written where the processor writes data to the UART. Since the writes pass through T-AXI Link, and reach up to the UART subsystem, the bandwidth drops can be observed when credit values are decreased (see Figure 24).

```
[0:32:S:taxi]> wd 2

TAXI port is enabled

                    Initial credits     New Credits
TAXI AR Credits:       0x00000004       0x00000008
TAXI WD Credits:       0x00000010       0x00000020
TAXI AW Credits:       0x00000004       0x00000008
TAXI RR Credits:       0x00000010       0x00000020
TAXI BR Credits:       0x00000004       0x00000008
MSTR AR Credits:       0x00000004       0x00000008
MSTR WD Credits:       0x00000010       0x00000020
MSTR AW Credits:       0x00000004       0x00000008
MSTR RR Credits:       0x00000010       0x00000020
MSTR BR Credits:       0x00000004       0x00000008
At WDcredit  = 0x1, transfer completed in 173508 ticks
At WDcredit  = 0x1, BW = 5.763 MB/s
At WDcredit  = 0x2, transfer completed in 119960 ticks
At WDcredit  = 0x2, BW = 9.336 MB/s
At WDcredit  = 0x4, transfer completed in 93356 ticks
At WDcredit  = 0x4, BW = 10.775 MB/s
At WDcredit  = 0x8, transfer completed in 67868 ticks
At WDcredit  = 0x8, BW = 14.734 MB/s
**************************************************************************
TEST PASSED: [MFAB:2] - Write Data Credit performance test passed
**************************************************************************
```

Figure 24: Write Data from processor to UART vs bandwidth observed

Another test was performed where an effort was made to congest the memory controller. The processors initiated reads to the memory controller as well as the DMA. The memory controller port had to fulfil the requests of the processors and the DMA at the same time. Figure 25 through to Figure 27 show the effects of additional traffic on the bandwidth of the system.

```
[0:32:S:taxi]> rr 2

TAXI port is enabled

                    Initial credits     New Credits
TAXI AR Credits:       0x00000008       0x00000008
TAXI WD Credits:       0x00000020       0x00000020
TAXI AW Credits:       0x00000008       0x00000008
TAXI RR Credits:       0x00000020       0x00000020
TAXI BR Credits:       0x00000008       0x00000008
MSTR AR Credits:       0x00000008       0x00000008
MSTR WD Credits:       0x00000020       0x00000020
MSTR AW Credits:       0x00000008       0x00000008
MSTR RR Credits:       0x00000020       0x00000020
MSTR BR Credits:       0x00000008       0x00000008
At RRcredit = 0x1, transfer completed in 480162 ticks
At RRcredit = 0x1, BW = 2.083 MB/s
At RRcredit = 0x2, transfer completed in 243792 ticks
At RRcredit = 0x2, BW = 4.102 MB/s
At RRcredit = 0x4, transfer completed in 133953 ticks
At RRcredit = 0x4, BW = 7.465 MB/s
At RRcredit = 0x8, transfer completed in 118764 ticks
At RRcredit = 0x8, BW = 8.420 MB/s
At RRcredit = 0x10, transfer completed in 116936 ticks
At RRcredit = 0x10, BW = 8.552 MB/s
At RRcredit = 0x20, transfer completed in 115746 ticks
At RRcredit = 0x20, BW = 8.640 MB/s
**************************************************************************
TEST PASSED: [MFAB:2] - Read Data Credit performance test passed
**************************************************************************
```

Figure 25: RR Credit varying effect of increased traffic on Memory port

```
[0:32:S:taxi]> br 2

TAXI port is enabled

                     Initial credits      New Credits
TAXI AR Credits:        0x00000008        0x00000008
TAXI WD Credits:        0x00000020        0x00000020
TAXI AW Credits:        0x00000008        0x00000008
TAXI RR Credits:        0x00000020        0x00000020
TAXI BR Credits:        0x00000008        0x00000008
MSTR AR Credits:        0x00000008        0x00000008
MSTR WD Credits:        0x00000020        0x00000020
MSTR AW Credits:        0x00000008        0x00000008
MSTR RR Credits:        0x00000020        0x00000020
MSTR BR Credits:        0x00000008        0x00000008
At BRcredit = 0x1, transfer completed in 173312 ticks
At BRcredit = 0x1, BW = 5.770 MB/s
At BRcredit = 0x2, transfer completed in 120146 ticks
At BRcredit = 0x2, BW = 8.323 MB/s
At BRcredit = 0x4, transfer completed in 93386 ticks
At BRcredit = 0x4, BW = 10.708 MB/s
At BRcredit = 0x8, transfer completed in 71191 ticks
At BRcredit = 0x8, BW = 14.047 MB/s
*****************************************************************
TEST PASSED: [MFAB:2] - BR Credit performance test passed
*****************************************************************
```

Figure 26: BR Credit shmoo effect of increased traffic on Memory port

```
[0:32:S:taxi]> ar 2

TAXI port is enabled

                     Initial credits      New Credits
TAXI AR Credits:        0x00000008        0x00000008
TAXI WD Credits:        0x00000020        0x00000020
TAXI AW Credits:        0x00000008        0x00000008
TAXI RR Credits:        0x00000020        0x00000020
TAXI BR Credits:        0x00000008        0x00000008
MSTR AR Credits:        0x00000008        0x00000008
MSTR WD Credits:        0x00000020        0x00000020
MSTR AW Credits:        0x00000008        0x00000008
MSTR RR Credits:        0x00000020        0x00000020
MSTR BR Credits:        0x00000008        0x00000008
At ARcredit = 0x1, transfer completed in 117728 ticks
At ARcredit = 0x1, BW = 8.494 MB/s
At ARcredit = 0x2, transfer completed in 117265 ticks
At ARcredit = 0x2, BW = 8.528 MB/s
At ARcredit = 0x4, transfer completed in 117072 ticks
At ARcredit = 0x4, BW = 8.542 MB/s
At ARcredit = 0x8, transfer completed in 116925 ticks
At ARcredit = 0x8, BW = 8.552 MB/s
*****************************************************************
TEST PASSED: [MFAB:2] - AR Credit performance test passed
*****************************************************************
```

Figure 27: AR Credit shmoo effect of increased traffic on Memory port

A comparison was made between the system bandwidths before adding processors' traffic to the DMA traffic, as shown in Figure 28 through to Figure 30.

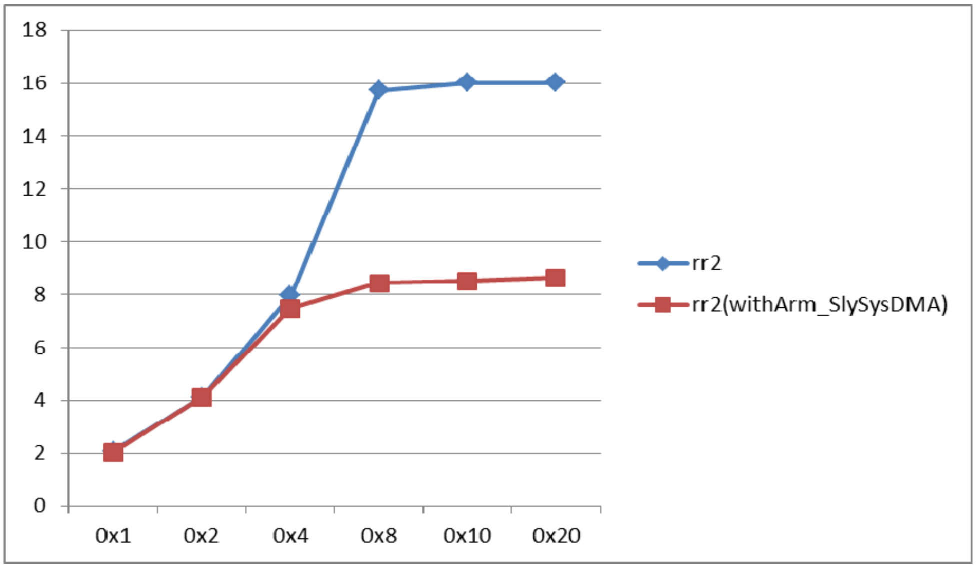


Figure 28: A comparison of RR bandwidth with ARM processor and DMA traffic and without ARM processor traffic

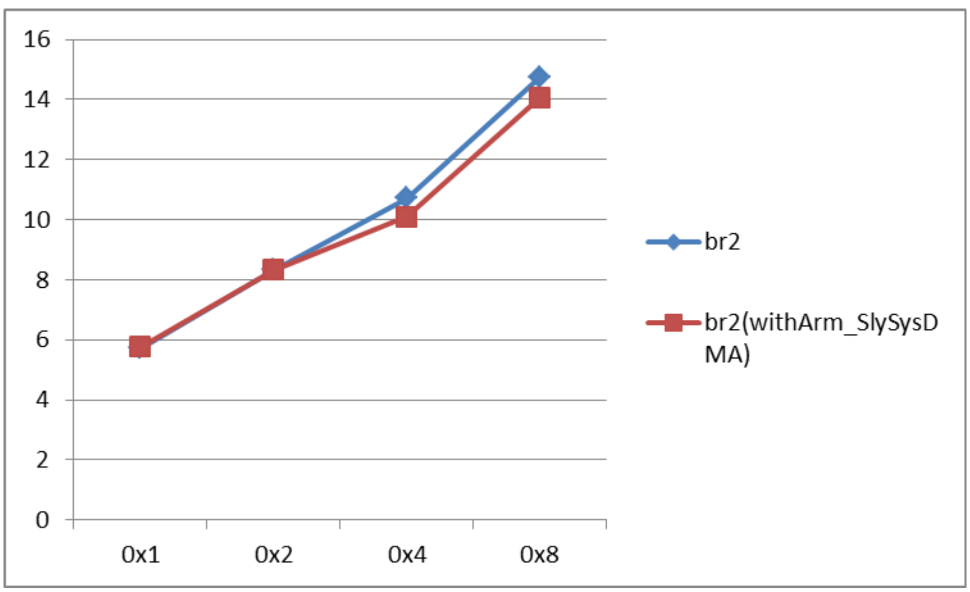


Figure 29: A comparison of BR bandwidth with ARM processor and DMA traffic and without ARM processor traffic

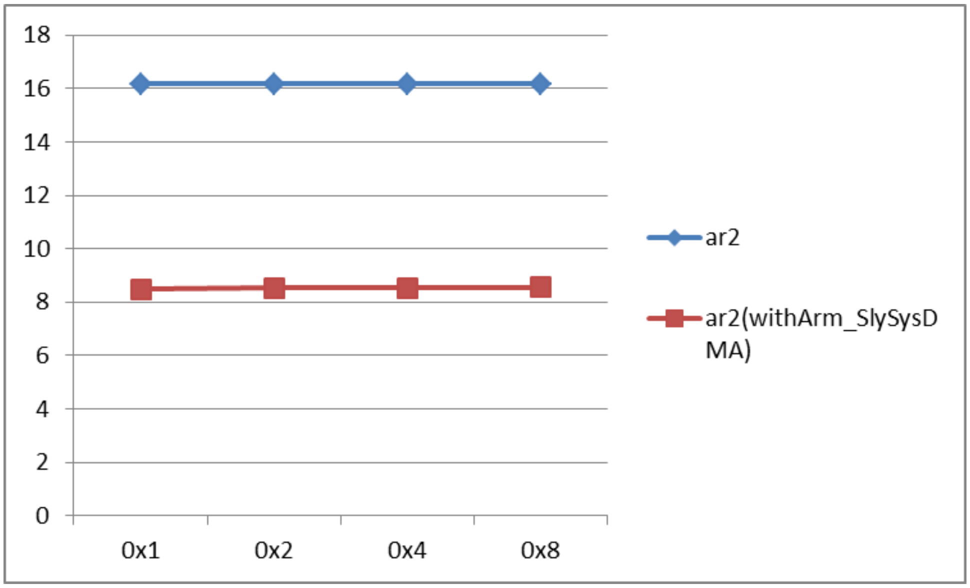


Figure 30: A comparison of AR bandwidth with ARM processor and DMA traffic and without ARM processor traffic

As seen from Figure 28 through to Figure 30, the bandwidth of the T-AXI Link decreased with an increase in traffic on memory port. This is due to the sharing of memory port by the DMA and the ARM processor to fulfill their transaction needs.

6.0 Conclusion

Thin-AXI was developed in-house at Broadcom as a method to reduce the pain of routing wide AXI buses across silicon chips, by collapsing transactions on wide buses down to packetized transactions on a narrower but faster bus. In this project, design verification of a T-AXI link was performed. From the test results, it was verified that the T-AXI link is fully functional according to the design specifications. Performance analysis was conducted on the T-AXI Links. In the analysis, various credit registers were varied from their minimum to maximum values to determine its effect on the Link bandwidth. It was observed that the lower credit values caused fewer outstanding transactions to be stored in the buffers and thus caused high latency in transaction completion. Higher latency caused the link bandwidth to drop, which was observed from the test cases. Overall, the link bring-up, shutdown, reset and link states were verified to be fully functional.

REFERENCES

[1] ARM (2013). *AMBA Trademark License* [online]. Available: http://www.arm.com/files/pdf/ AMBA_trademark_license.pdf (URL)

[2] Xilinx (2013, June 21). *Virtex-6 FPGA - SelectIO Resources*, User Guide, UG361 (v1.4) [online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug361.pdf (URL)

[3] Broadcom (2012, Jan 13). *Thin AXI Interface and Design Specifications* (Internal Document)

[4] ARM (1997, April). *AMBA Advanced Microcontroller Bus Architecture Specification* (Document Number: ARM IHI 0001D) [online]. Available: http://larc.ee.nthu.edu.tw/~sjtsai/current_research/paper_review/Advanced%20Microcontroller%20Bus_Architecture_Specification.pdf (URL)

[5] Xilinx (2012). *AXI4 Technical Seminar, Xilinx* [online]. Available: http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf (URL)

[6] Saad Z. Asif (2001), *Next Generation Mobile Communications Ecosystem, Technology Management for Mobile Communication*, First Edition.

[7] Pasricha S., Dutt N. (2008), Chapter 3 in *On-Chip Communication Architectures: System on Chip Interconnect*, First Edition.

[8] Coppola M., Grammatikakis M. D., Locatelli R., Maruccia G., Pieralisi L., (2008). *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC,* First Edition.