Open Library of IP Module Interfaces for AMBA Bus

Laurentiu Acasandrei, and Angel Barriga, *Member, IAENG*

***Abstract*—This paper describes the design of Intellectual Property (IP) modules of the most widely used communication standard interfaces, the AMBA bus. There is described the design of masters and slaves interface modules for APB, AHB, AXI and AXI-Stream buses. The IP modules presented can be used in systems with a variety of design constraints (low speed to high speed applications, low power consumption, etc). For the slave APB bus interfaces and the master/slave AHB a development environment for design and simulation based on the GRLIB library is provided.**

***Index Terms*—buses, interconnection network, IP module design, open source hardware**

I. Introduction

When designing high performance peripherals that require high bandwidth data transfers, the selection of bus standard and the interface design is a critical step. Very often the designer doesn't have the liberty of choosing a bus standard due to the project constrains that bind him to a specific system or technology. What aggravates the circumstances is the fact that transactions on the bus are influenced by the slave/master arbitration scheme, the chaotic behavior of peripherals and software applications. Even if the designer uses high performance bus architecture, this does not guarantee that the entire system will be fast. In general designing bus interfaces for peripherals is a challenging and time consuming task that plays a major role in obtaining a good system performance.

Every electronic device contains heterogeneous hardware elements, such as processors, graphic and sound cards, input-output (I/O) interfaces, memory, hard drives and custom hardware performing specialized tasks, which are interconnected using buses. From an electronic perspective a bus is an abstract concept that attaches to normal signal lines or wires the convention that they represent the communication channel through which information flows from one component to another.

Depending on the system requirements a bus can be formed by one up to several thousand signal lines in more complex systems, e.g. the interconnect between Processing System (PS) and Programmable Logic (PL) in Xilinx Zynq [1] and reaching ten thousand connections between Super Logical Regions (SLR) in Virtex-7 FPGA's [2]. The technological advances in deep submicron (DSM) technologies have given birth to multimillion transistor chips, generically referred as system on chip (SoC), that integrate various electronic circuits and computer components onto a single, integrated circuit (IC). A SoC can contain processors, memory controllers, cache and on-chip memory, timers, digital signal processing units, analog to digital converters, digital to analog converter, encryption cores, wireless and optical communication interfaces, programmable logic, universal serial bus (USB), Ethernet, general purpose I/O interfaces and all these components, varying in numbers from tens to hundreds, are connected via an on-chip communication fabric that is referred to as on-chip bus. SoC's are broadly deployed in embedded systems where the design engineers need to optimize power consumption, size, reliability, performance and cost.

Due to the technology particulates of an IC, the system communication architectures, which were having their physical layout on Printed Circuit Boards (PCBs), were considered inappropriate for an IC layout. New on-chip, or intra-die, standards and communication architectures emerged and took advantage of new processing paradigms, increasing frequency speed and die size for CMOS IC technology (see Fig. 1).

The continuing advances in DSM technology have enabled the integration of more functionality, but in the same time it drastically increased the complexity and difficulty of the design-verification cycle. The International Technology Roadmap for Semiconductors [3] noticed that the technology progress will surpass by far the engineer ability to integrate and verify a design containing huge amount of transistors. Rapid technology change shortens product life cycles and makes time-to-market a critical issue for semiconductor customers.

The aim of this paper is to present a communication interface library included in SHORES (Software and Hardware Open Repository for Embedded Systems), under GNU GPL [4] license, providing support for the design community in standard IP connection modules. For the most used communication standard, i.e. AMBA, we designed masters and slave interfaces, for the APB, AHB, AXI, AXI-Stream protocols, that can be employed in a large variety of systems with different constraints (i.e. for low power, low speed up to high speed streaming applications). For the APB slave and AHB master/slave interfaces also a test

Manuscript received March 5, 2005; revised March 11, 2005. This work was supported in part by Spanish Ministerio de Ciencia y Tecnología under the Project TEC2011-24319, by Ministerio de Economía y Competitividad under the Projects IPT-2012-0695-390000, co-financed by FEDER, and RTC-2014-2932-8.

Laurentiu Acasandrei is with the Instituto de Microelectrónica de Sevilla (IMSE-CNM) and the University of Sevilla, Spain (e-mail: laurentiu@imse-cnm.csic.es).

Angel Barriga is with the Instituto de Microelectrónica de Sevilla (IMSE-CNM) and the University of Sevilla, Spain (e-mail: barriga@imse-cnm.csic.es)

framework, based on GRLIB library [5], is provided.

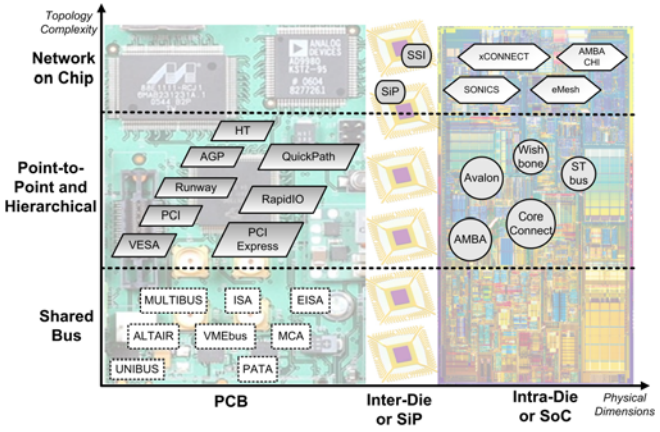

Fig. 1. Evolution of communication architectures. The evolution of the communication architectures from simple shared bus, to point-to-point and hierarchical bus, onto network on a chip topologies can be noted. Due to technological advances, their physical implementations have passed from PCB to chip-to-chip interconnects inside same package and reached deep sub-micron dimensions for the gate level interconnect inside a chip.

II. Elements and Mechanisms of Communication Architecture

The internal components of SoC communicate with each other using buses. A bus can connect several components using communication channels represented by wires in case of integrated circuit or air in case of radio devices and fiber optics in case of optical modules. IC buses are divided in serial buses (i.e the information is transmitted using a single wire) and parallel (i.e multiple wires convey the information in the same time). At physical level, data transmitted by a component travels from its output pins through wires to the receiver input pins. Depending on the direction of the data flow the buses can be unidirectional (i.e the data is transmitted only from source to source to destination) and bidirectional (i.e the data is transmitted from source to destination and vice-versa). A bidirectional bus is electrically more complex and inherently slower than a unidirectional bus. The data transmission/reception between SoC components is governed by rules and conventions that combined constitute the bus or communication protocol. Taking into consideration the targeted technology limitations and performance requirements, the bus protocol specifies the appropriate syntax, semantics and synchronization of communication. To successfully implement the communication protocol, every IP or component in SoC needs a bus interface responsible for data transfers between that component and bus. The component ports that have the specific functionality of receiving and transmitting data are referred to as bus interface ports. The connection to the bus of interface input/output ports can be made with simple wires, but in special cases where there is a crucial need for performance, it is made with FIFOs, buffers and frequency converters. Depending on the direction of data flow there are three types bus component interfaces are:

- Master interfaces are capable of initiating and controlling read/write data transfers. A typical IP or SoC component with master interface can be a processor, DSP or video accelerator.
- Slave interfaces only responds to a read/write data transfer previously initiated by a master. Typical IP or SoC component with slave interface are memories, low speed communication core like SPI, I2C, UART.
- Mixed master/slave interfaces are capable of initiating and controlling read/write data transfers but also can respond to data transfer previously initiated by a different master. A typical IP or SoC component with master/slave interface is a DMA controller.

All data transactions of the busses are supervised and controlled by specialized bus control circuitry. A typical bus controller topology consisting of two types of logical components (arbiter and decoder) that control the data flow between the master and slave interfaces using bus multiplexers.

The arbiter is responsible for solving bus congestion issues, where multiple masters try to access the bus simultaneously, by granting bus access only to one master and signaling to the rest of the masters that the bus is busy and they should wait until the bus is available. The arbitration scheme has a direct impact on system performance and several arbitration algorithms and techniques have been developed to satisfy the performance requirements for bandwidth, latency, master bus acceptance rate and master access bus waiting time.

Every master or slave interface connected to on-chip bus has assigned a range of addresses. All the master or slave bus addresses are organized in an address map. Before starting a data transfer on the bus, the master first transmits the destination address of slave interface. The decoder's responsibility is to decode the destination address of a data transfer started by master and to select the corresponding slave to receive the data. Bus architectures can contain one decoder, i.e. centralized decoder, or multiple decoders distributed at every slave interface. The centralized decoding has the advantage that new bus component can be added to the system with little changes needed to be done, resulting in design ease in adding new components to the bus.

In SoC's where multiple components are connected to a shared bus or multiple masters request simultaneous access to the same slave, e.g. on chip memory or DMA controller, the bus arbiter selects the master that will be granted access to the bus. When a master is granted access, data can be transferred over by using one of the five most common transfer types: basic transfer, burst transfer, split transfer, out-of-order transfer and broadcast transfer. Other transfer types are specific to custom communication protocols that fulfill special purpose.

The type the bus topology employed influences the performance, power consumption and the complexity of the communication architecture. Simpler bus topologies imply simpler bus protocols ideal for communication between a small numbers of components, but the bus performance is drastically reduced when handling many heterogeneous components. Complex bus topologies efficiently handle data transfers between a large number of components with arbitrary bus behavior, but they have the disadvantage of using complex bus protocols thus increasing overall power consumption and NRE (Non-Recurring Engineering) costs.

ISBN: 978-988-19253-4-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

III. AMBA Bus

A SoC is basically an IP-centric complex world where communication between its heterogeneous components is challenging. ARM Holdings, a British fabless semiconductor and software company was first to recognize [6] the need for an open communication standard that specifies the connection and communication protocols of functional blocks in a SoC. As result, the Advanced Microcontroller Bus Architecture (AMBA) was introduced by ARM in 1996. The objective of AMBA specification is to:

- Facilitate right-first-time development of embedded microcontroller products with one or more CPUs, GPUs or signal processors.
- Be technology independent, to allow reuse of IP cores, peripheral and system macrocells across diverse IC processes.
- Encourage modular system design to improve processor independence, and the development of reusable peripheral and system IP libraries.
- Minimize silicon infrastructure while supporting high performance and low power on-chip communication.

Since their first appearance, the AMBA based IPs (e.g. Cortex processors, DSP and graphics cores (Mali)), provided by ARM, have gained tremendous popularity and reached a market [7] penetration in 2012 of 95% for mobiles phones, networking 35%, digital TVs 45% and microcontroller of 18%. In the past years AMBA standard was used for an increasingly number of non-ARM platforms, like MIPS from Infineon and LEON3 from Aeroflex-Gaisler. AMBA was adopted as the communication standard for FPGA (i.e Altera and Xilinx) and PSoC (Xilinx ZynQ SoC and Cypress PSoC4 and PSoC5).

Due to stringent time-to-market constrains, major semiconductor companies (e.g. Texas Instruments, Atmel, Analog Devices) are using ARM with AMBA based IPs in their SoCs. Nowadays, with the strategically shift of IBM, from general purpose processors, to SoCs for their servers and cloud based services, it can be easily concluded that the AMBA based devices will dominate all the consumer and industrial markets in the near feature.

Since the initial release of AMBA 2.0, the AMBA architecture has evolved and adapted to fulfill the industry needs, by incorporating new communication protocols: ACE, ACE-Lite, AXI4, AXI4-Lite, AXI4-Stream, AXI3, ATB, AHB-Lite and APB. The majority of the AMBA compliant IP cores will use separately or in combination the following architecture/protocols APB, AHB, AXI, AXI-Lite and AXI-Stream. The special purpose architecture and protocols, i.e. ACE, ACE-Lite and CHI are used for data synchronization between multiple processor and system memory. The ATB protocol is used for debugging and diagnostic.

IV. Design and Verification Methodology for AMBA Interfaces

For each protocol we have designed a basic master and slave interface. All the designs and they respective testbenches have been described in VHDL and they are technology independent.

In the design of every master/slave interface it was employed a structured “two-process” VHDL design method [8]. This method is applicable to synchronous single clock design and it purpose is to: a) Provide uniform algorithm encoding; b) Increase abstraction level; c) Improve readability; d) Clearly identify sequential logic; e) Simplify debugging; f) Improve simulation speed; g) Provide one model for both synthesis and simulation

The “two-process” design methods uses high-level sequential statements for coding, two processes per entity and record types for signals declarations. It results in a uniform way of encoding any algorithm in a VHDL entity and improved readability. One process contains all combinational (asynchronous) logic, and the second process contains all sequential logic (registers).

To verify the correct behavior of the APB slave a modified version of GRLIB AMBA Test Framework [5] was used. This framework contains packages (AHB master, AHB slave and AHB arbiter/controller core) that help in the verification of new AMBA cores. The AHB master and slave cores have debug interfaces that can be controlled using external stimuli. Based on the GRLIB AMBA Test Framework, we have designed a portable, self-contained and small size AMBA interface design and test framework. This new framework contains basic APB, AHB,, master/slave interfaces.

V. AMBA Interfaces Design

A. APB Protocol

The Advanced Peripheral Bus (APB) [9] defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipelined and is used to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol. Typical usage of APB interface is to access the programmable control registers of peripheral devices.

The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow. Every transfer takes at least two cycles. The APB can interface with: AHB, AHB-Lite, AXI and AXI-Lite.

The APB protocol has two independent data buses, one for read data and one for write data. The data buses can be up to 32 bits wide. Because the buses do not have their own individual handshake signals, it is not possible for data transfers to occur on both buses at the same time.

In a typical AMBA based microcontroller system (see Fig. 2) the high performance components, i.e. processors or on-chip memories, are connected to a high speed bus (i.e AHB or AXI) while the lower bandwidth peripheral are connected to the low speed, low power APB bus.

The APB is connected to the AHB bus via an AHB/APB bridge. In the APB bus there is usually only one master, i.e. the APB bridge, and the rest of the APB peripherals are slaves.

ISBN: 978-988-19253-4-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

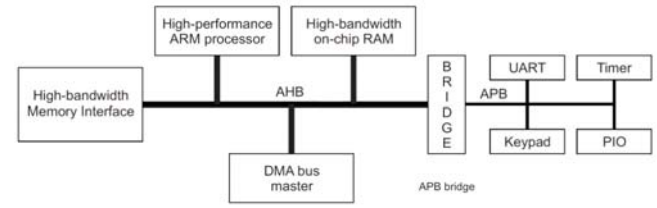

Fig. 2. APB connection in an AMBA system [10]

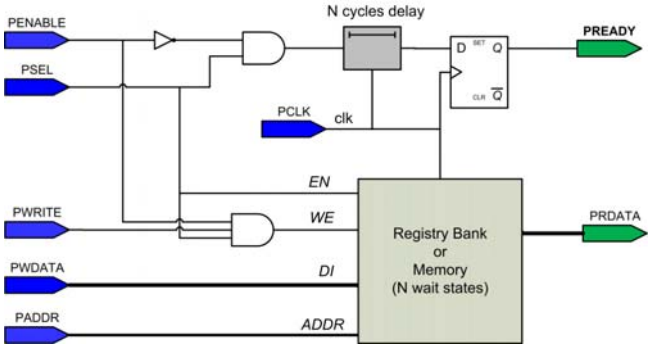

Fig. 3. Schematic of the APB slave interface with N wait states.

The AMBA APB protocol specification defines write/read transfers where the APB bridge receives slaves responses on the next clock cycle or it has to wait several clock cycle for the data. We have designed two basic APB compatible slave interfaces: an APB interface with no wait states and a slave interface that responds after several clock cycles (Fig. 3).

B. AHB Protocol

The AMBA AHB (Advanced High-performance Bus) [11] is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques. AMBA AHB has the following features: burst transfers, split transactions, single cycle bus master handover, single clock edge operation, non-tristate implementation, wider data bus configurations (64/128 bits).

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and switch multiplexer, which selects the appropriate signals from the slave that is involved in the transfer. A typical AMBA AHB system design contains the following components:

- AHB master. A bus master is able to initiate read and write operations.
- AHB slave. A bus slave responds to a read or a write operation initiates by the master.
- AHB arbiter. The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority or fair access can be implemented depending on the application requirements. An AHB would include only one arbiter, although this would be trivial in single bus master systems.
- AHB decoder. The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer. A single centralized decoder is required in all AHB implementations.

The AHB protocol has two independent data buses, one for read data and one for write data. Data transfer can be single or burst transfers where the address phase only last one cycle while data phase can last one (zero wait state) or several clock cycles(multiple wait states) .

The AHB bus master has the most complex bus interface when compared with the other components of AMBA bus. The AHB bus master is responsible for initiating and controlling read/write requests on the AHB bus. The AHB bus master controls the data transfer on the AHB bus but in return the master interface is controlled by custom IP's (e.g. processor, DMA, etc.). The IP controls and communicate with the AHB bus master using intellectual property interface (IPIF). An IPIF heavily depends on IP communication requirements and on the type of master (AHB, AXI or AXI stream). A custom IPIF for AMBA AHB master is presented in Fig. 4. The IPIF was designed to be simple and resembles the on-chip ram memory interface. The *transfer_type* signal indicates to the AHB master the type of the transfer (i.e. single, burst incremented, fixed burst, wrapped fixed burst). The *size* signal indicates the transfer data width (i.e. byte, half-word, word). When asserted the *start* signal forces the AHB master to start a transfer. The *addro* signal represents initial start address. The *dwrite* signal indicates a write or read operation. The *dataw* signal carries the data from the IP to the AHB master. The *datar* signal is used to carry the data received by the AHB master, during read transfers, to the IP.

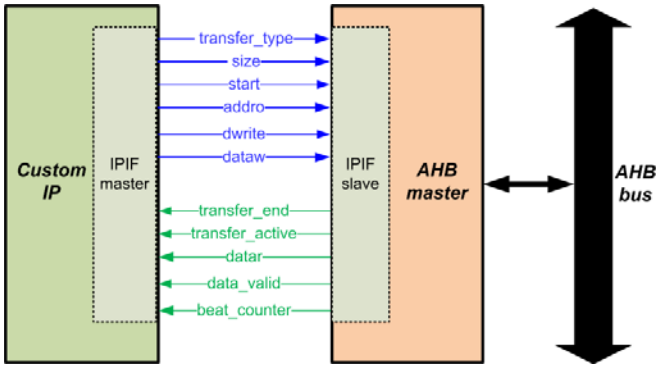

Fig. 4. Custom interface between an IP and an AHB master.

A. AXI Protocol

The AMBA AXI (Advanced Extensible Interface) protocol [12] supports high-performance, high-frequency system designs. The AMBA 4 AXI protocol is an update to AMBA 3 AXI that enhances the performance and utilization of interconnect when used by multiple masters. It includes the following enhancements: support for burst lengths up to 256 elements, quality of service signaling, support for multiple region interfaces. AMBA AXI 4 is recommended for new designs.

The bus AXI is backward compatible with existing AHB

ISBN: 978-988-19253-4-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

and APB interfaces. The AXI protocol includes the AXI4-Lite specification, a subset of AXI4 for communication with simpler control register style interfaces within components.

Figure 5 shows that a write transaction uses the write address, write data, and write response channels. The data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. The read data channel carries both the read data and the read response information from the slave to the master. The write data channel carries the write data from the master to the slave and includes a byte lane strobe signal for every eight data bits, indicating which bytes of the data are valid.

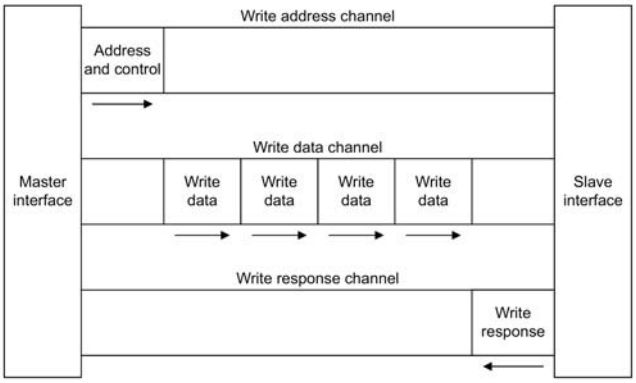

Fig. 5. AXI Channel architecture of writes [12]

The starting point, in the design of the AXI slave interfaces, is the previously designed AHB slave interface. The AXI slaves have similar behaviors to the AHB slaves, but because the AXI standard permits separate channel for read and write operations, the AXI slaves have simpler logic structure.

Similar to the design of AXI slave, the starting point in designing an AXI master is the actual AMBA AHB master (Fig. 6). Only a little logic has been added to ensure compatibility for the *HREADY*, *HRDATA* and *HGRANT*. Advanced features like separate read and write data channels, issuing multiple outstanding addresses, out-of-order transaction completion, access permission, transaction buffering, atomic accesses, low power states and quality of service signaling are not supported.

B. AXI-Stream Protocol

The AXI4-Stream protocol [13] is used as a standard interface to connect components that wish to exchange streaming data. The interface can be used to connect a single master that generates data, to a single slave, that receives data. The protocol can also be used when connecting larger numbers of master and slave components. The protocol supports multiple data streams using the same set of shared wires, allowing a generic interconnect to be constructed that can perform data upsizing, downsizing and routing operations.

The AXI Stream protocol was design with the purpose of reducing signal routing between a master and slave. This protocol is ideal for FPGA implementation. Data streaming interfaces are needed in DMA transfers and DSP applications, especially in video and image processing where the transfer speed and bandwidth is a critical performance factor.

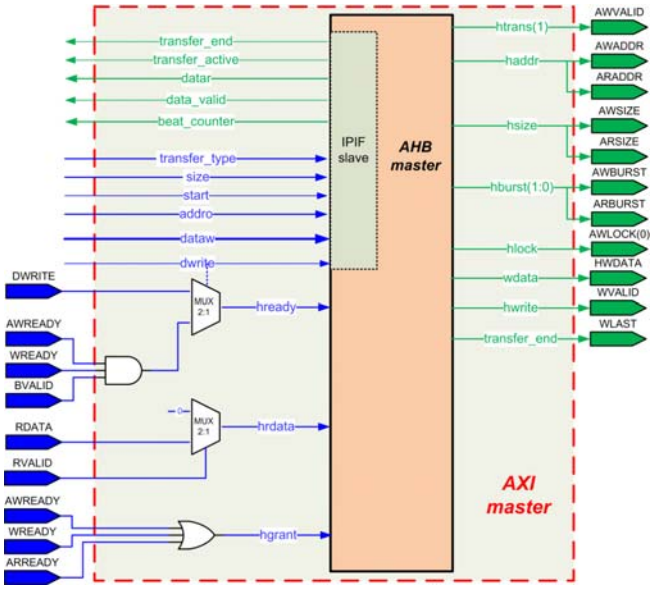

Fig. 6. AXI master interface

From a point of view on how the data flows from the input to output ports of an IP, two types of IP's can be distinguished (see Fig 7):

- Direct data transfer. For these types of IP's, the output is a function only of the current input data. The most known IP's are DMA controllers and binary image thresholding circuits.
- Feedback data transfer. For this type of IP's the output is a function of current input data and internal intermediate data. This is a vast category that includes image filters (Sobel, Median, Gaussian, FIR, IIR), image scaling, object detectors, etc.

The AXI stream master or slave interface for direct data transfer is simple and consists only of delaying the interface signal with N cycles. On the other hand, the AXI stream master or slave interface for feedback data transfers is more complex and must use input FIFOs for data synchronization. Also the handshake signals are delayed with (M+N) cycles.

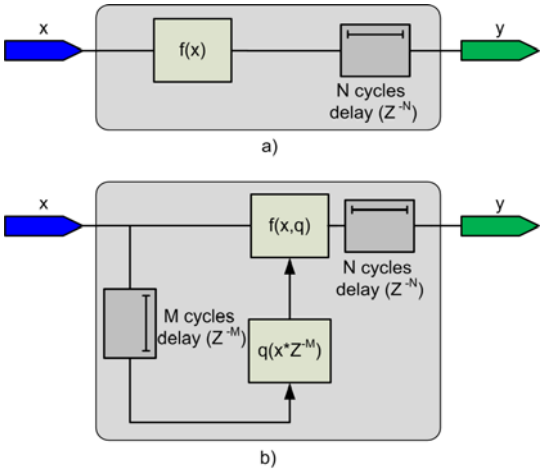

Fig. 7. Data flow in IP's : a) direct data transfers, b) feedback data transfers

VI. IP Modules Verification

To verify the correct behavior of the designed interfaces a modified version of GRLIB AMBA Test Framework [5] was used. This framework contains packages (AHB master, AHB slave and AHB arbiter/controller core) that help in the

ISBN: 978-988-19253-4-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

verification of new AMBA cores. The AHB master and slave cores have debug interfaces that can be controlled using external stimuli. Based on the GRLIB AMBA Test Framework, we have designed a portable, self-contained and small size AMBA interface design and test framework. This new framework contains basic APB, AHB, AXI, master/slave interfaces and corresponding testbenches.

The APB testbench architecture used for verification is presented in Fig. 8. The AT_AHB master and slave components are non-synthesizable and contain a debug interface that can be controlled by the stimulus generated by the testbench. The AHB arbiter/controller and AHB/APB bridge are synthesizable cores instantiated from the GRLIB library. The AHB/APB bridge is connected to the APB bus and two units under test. The APB slave0 handles zero wait state data transfers while the APB slave1 has transfer latency of 5 wait states. The APB slaves units are synthesizable.

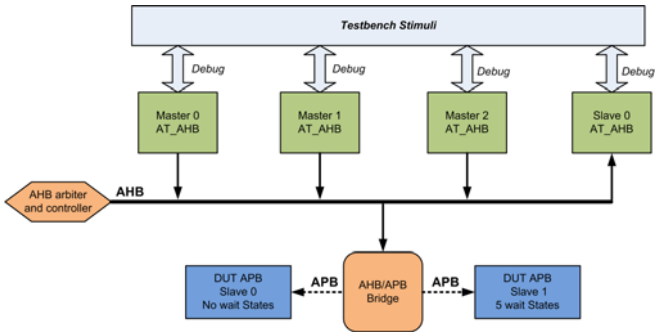

Fig. 8. AMBA testbench setup for APB slaves

To verify the correct behavior of the AHB master, the AMBA testbench from Fig. 9 was used for verification. The testbench contains three non-synthesizable AT_AHB master and three non-synthesizable AT_AHB slave with different wait states. The testbench also contains a synthesizable AHB arbiter/decoder and one AHB custom master.

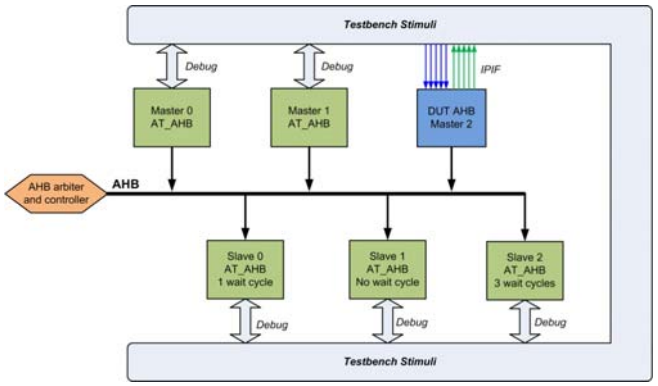

Fig. 9. AMBA testbench setup for AHB masters

VII. Conclusions

The interface design of the AMBA bus (APB, AHB, AXI and AXI-Stream) has been described. These interfaces can be used in systems with different restrictions (low-power, low speed up to high-speed transmission applications). For the APB slave and AHB master/slave has developed a design and testing environment based on the library GRLIB. Unfortunately, due to the limitation imposed by the license of all functional models AXI and AXI-Stream no test framework is provided for these master/slave interfaces. The AMBA architecture, being an open standard, has the advantage of having many bridges to other communication architectures (Core Connect, Wishbone, Avalon, etc). This means that the interfaces presented in this communication can be integrated with minimal effort in embedded systems based on different bus architectures.

The VHDL source codes of the interfaces described in this communication are included in the Software and Hardware Open Repository for Embedded Systems (SHORES) [4]. Its main goal is to make available to the public, in an open-source style, the designs and results from academia/research community. SHORES hosts the source code of various software and hardware design projects that combined with the newest algorithms proposed by academia give birth to embedded solutions to the most challenging obstacles in the fields of vision, bio-cryptography, signal processing, etc. The IP modules can be used and modified by the user to implement their own specific application. The repository aim is to give support to the embedded system community during the design process in order to increase productivity and help the development of complex systems.

References

[1] Zynq-7000 All Programmable SoC Overview (Xilinx DS190), http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
[2] 7 Series FPGAs Overview (Xilinx DS180), http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
[3] Design, International Technology Roadmap for Semiconductors (2012), http://www.itrs.net
[4] SHORES, Software and Hardware Open Repository for Embedded Systems, http://www.imse-cnm.csic.es/shores/
[5] LEON/GRLIB Configuration and Development Guide, pp. 21-27, www.gaisler.com/products/grlib/guide.pdf, 2012.
[6] AMBA Open Specifications (2013), http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
[7] ARM annual report 2012, pages 16-17, http://www.arm.com
[8] Gaisler, J.: A structured VHDL design method. Fault-tolerant microprocessors for space applications (white paper), pp. 41-50 (2010).
[9] AMBA APB Protocol Version: 2.0 (2010), http://www.arm.com
[10] ARM AMBA Specification (Rev 2.0), pp. 46-48, 69-70, 161-180, http://www.arm.com
[11] AMBA 2 Specifications (1999), http://www.arm.com
[12] AMBA AXI and ACE Protocol Specification (2013), http://www.arm.com
[13] AMBA 4 AXI4-Stream Protocol Specification (2011), http://www.arm.com

ISBN: 978-988-19253-4-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)