An Interface Methodology for Retargettable FPGA Peripherals

Tien-Lung Lee, Neil W. Bergmann
School of ITEE, The University of Queensland, Brisbane Australia

Abstract - Initially, IP cores in System-On-Chip (SOC) were interconnected through custom interface logics. The more recent use of standard on-chip buses has eased integration and eliminated inefficient glue logic, and hence boosted the production of IP functional cores. However, once an IP block is designed to target a particular on-chip bus standard, retargeting to a different bus is time consuming and tedious. As new bus standards are introduced and different interconnection methods are proposed, this problem increases. Industry standard Bus Wrappers are intended to ease the interface problem, but performance overheads make them unattractive. A new methodology is presented that can automate the connection of an IP block to a wide variety of interface architectures with low overhead through the use a special Interface Adaper Logic layer.

1. Introduction

Ever-increasing FPGA capacity and low design productivity has introduced the so-called Design Productivity Crisis [14]. IP reuse is one of the keys to close the gap between capability and productivity. In traditional ASIC design, to reuse legacy or third-party IP, designers need to understand the functionality, coding, documentary, interfacing and other properties, before any modifications can be done. The Reuse Methodology Manual for System-on-a-chip Designs [15] has been published as one possible solution. However the Reuse Methodology Manual is only a guideline, designers still have to understand the complex communication protocols in order to interface an existing IP design.

A new approach to reconfigurable System-on-Chip designs, based on FPGAs, is the use of platform technology. These platforms have libraries that contain pre-designed and pre-verified IP soft cores, so user can mix-and-match the functional IP from the library to assist in design of the required system. Problems still remain when the third party and library IP have already been designed to communicate with a particular bus protocol. It is very difficult to extract the functional core and retarget it to a different interface protocol.

The VSI Alliance (VSIA) has defined a standard interface between the IP behavioural logic and a Bus Wrapper called the Virtual Component Interface (VCI) [10]. The primary intention of VSIA is to provide a simpler and more universal protocol then standard bus interfaces. However, they introduce a considerable protocol conversion overhead that is a problem in for time-critical applications.

In this paper, we aim to extend this idea of automating the interface between the core logic of a peripheral and the particular communication network that connects the peripheral to a central processor in a reconfigurable SoC design. This allows peripheral re-use across different reconfigurable SoC manufacturers, and also between different communication strategies (such as parallel bus, serial bus, Ethernet, SPI, I2C).

2. Interface Methodology

In our proposed methodology, we aim to ease the complex process of interfacing peripheral IP blocks to a system interconnection protocol, such as a bus. The methodology should have the following functionalities:

- IP independence - Separates IP behavioural logic from interface logic to provide IP-reuse.
- Targets diverse system interconnection protocols (busses, serial networks, packet-switched networks).
- Provide low-overhead Interface Adaptor Logic – eliminate the protocol conversion issues that the Bus Wrapper approach introduces.
- Parameterisable – only generates the minimum required Interface Adaptor Logic dependent on the peripheral features.
- Generic – cater for different standard protocol specifications and different platform restrictions within a given communication style. For example, with a bus, a single Interface Adapter should be able to cope with different bus widths, number of interrupt lines, etc.
- Simple integration guidelines – it should be easy to design a peripheral that can work with the Interface Adapter methodology. Where possible, extra intelligence is added to the Interface Adapter, to ease peripheral design.

3. Reconfigurable Platforms – Integration Platforms

Since the introduction of reconfigurable SoC platforms with standard integrated bus architectures, IP functional blocks that were built to interface with a certain bus protocol can migrate to a different platform that has the same bus architecture with little or no change to the IP logic. However, reconfigurable SoC platforms from different manufacturers all use different standard bus protocols, and IP cores are not easily transferable from one manufacturer's platform to another's.

The following section will discuss some of the current available reconfigurable platforms, and their bus architectures, along with a description of the vendor specific methods and tools to design a feasible custom IP for each of the platforms. The main purpose is to show the difficulties that designers need to go through to compose a functional IP.

3.1 Commercial Reconfigurable SoC Platforms

Xilinx – Virtex II

Xilinx [1] is one of the leading innovators in reconfigurable computing; its Virtex II system has the highest density FPGAs in the industry. In the Virtex II SoC platform, the MicroBlaze soft-core processor [2] is implemented with IBM's CoreConnect bus architecture [3]. The bus is open-licensed, and IBM provides licensees with the bus arbiters and bus-bridge designs.

The CoreConnect bus architecture consists of three buses [3] for interconnecting the cores, library micros and custom logic blocks: Device Control Register Bus (DCR), Processor Local Bus (PLB) and On-chip Peripheral Bus (OPB). The aim of the DCR bus is to save PLB and OPB bandwidth. The PLB bus is a primary high-performance memory bus that interfaces directly with the processor. The OPB is designed for less time-critical connections to peripherals. Figure 1 shows MicroBlaze, or IBM's PowerPC processor, with the CoreConnect.

The general flow for creating a custom peripheral consists of the following steps: (1) the HDL code for the peripheral core is constructed as well as the code to interface with the OPB protocol. Examples of protocol signals that peripherals need to consider are OPB clock, reset, address bus, data bus, and Read/Write (2) Software device drivers for the peripheral are written; (3) A number of specification files need to be added to the VHDL code. The specification files include the user constraint files, which define the I/O connection from the peripheral for the external pins; (4) hardware and software are compiled and verified

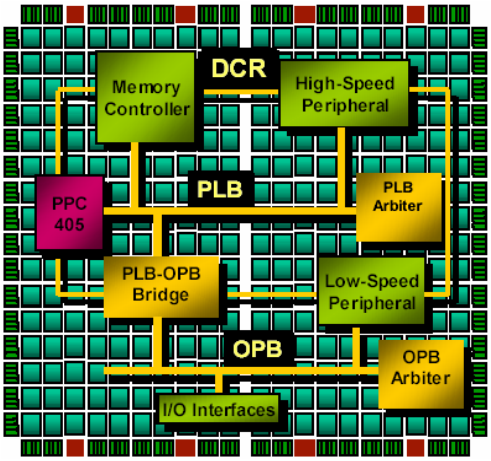

Figure 1. Foundation of MicroBlaze, or IBM PowerPC, with CoreConnect. (Source: [4])

The process of making a custom peripheral can be complicated and time-consuming task. Xilinx is aware of these problems and supports two interface CAD tools, the *OPB Intellectual Property Interface* (IPIF) [4] and *System Generator Pro* [4].

The OPB IPIF is a parameterisable glue-logic module provided though the CoreGen GUI CAD tool within Xilinx' ISE package [5]. At this stage, IPIF is only preliminary and aimed at common peripherals. More details on IPIF will be discussed in a later section.

System Generator Pro is a GUI interface tool that allows designers to automatically connect IP cores to OPB bus. It can automatically generate some hardware and software components associated with bus interfacing, such as netlist, implementation files, and device drivers.

Altera – Excalibur

When system engineers chose Altera's reconfigurable platform – Excalibur [6], they can select two different embedded processors, the ARM9 32-bit processor or the Nios processor.

The ARM processor is provided with the AMBA [7] bus, which is an open specification from ARM. In the AMBA bus architecture, there are three distinctive buses, the Advanced High-performance Bus (AHB), the Advanced System Bus (ASB) and the Advanced Peripheral Bus (APB). Figure 2 shows a general AMBA system. In the Altera Excalibur ARM-based platform, AHB only is implemented.

Altera Nios processor embeds Altera's own Avalon bus [8], as shown in figure 3. Unlike some other bus architectures, the Avalon bus focuses on simplicity that yields a shorter learning curve for designers.

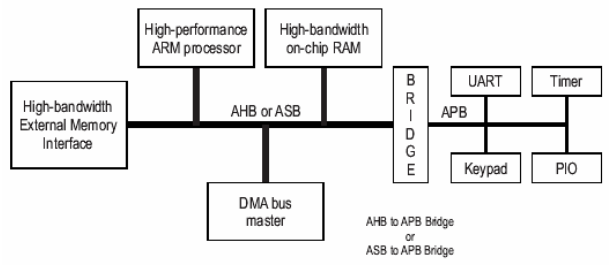

Figure 2. AMBA Bus Architecture (Source [7])

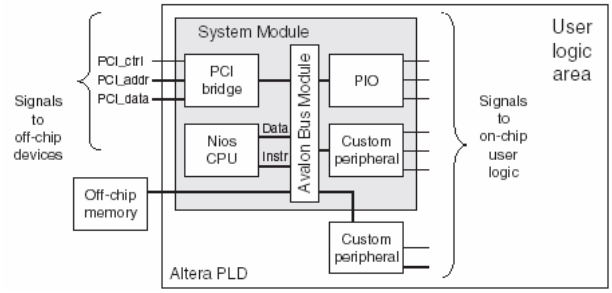

Figure 3. Nios System. (Source: [8])

The design flow to customise a peripheral for Excalibur is identical for ARM and Nios-based platforms. (1) Behaviour HDL code for the peripheral is constructed; (2) HDL code is fed through the SOPC Builder Ready system-level design tool [9]. The designer then configures the desired peripheral pins to bus connection signals, and external I/O pins. SOPC will automatically interface the peripheral behaviour core to the selected bus architecture, AHB or Avalon bus; (3) Drivers for peripheral are also auto-generated by SOPC, exporting header files and custom libraries; (4) hardware and software are compiled and verified.

3.2 Deficiencies in Current Methodologies.

Without the support of system tools provided by platform manufacturers, the processes for designing custom peripherals are time-consuming and complex.

However these system tools and libraries provided by manufacturers only target their own bus architecture. These IP blocks cannot easily be retargeted to any other busses on different platforms, or to other communication architectures and protocols. This limits the reusability of IP.

3.3 Other Interconnection architectures

Within a reconfigurable SoC platform, a standard system bus does not have to be the only mean of communication. One can take advantage of the flexibility of the reconfigurable logic to implement other interconnection networks, which may be more appropriate for a particular system design. Examples of such alternate communication network protocols are I2C, SPI, Firewire, OneWire, ATM and Ethernet.

Many other new innovative bus standards and interconnection architectures are also under investigation by the research community. However a diversity of communication methods is a problem for re-use of IP blocks if every block has to be redesigned for each new method. One of the goals of this research is to be able to quickly port IP peripheral libraries to new interconnection architectures.

4 IP Interface Methodologies

A key problem in conventional ASIC full-custom SoC is to interface IP library modules into a complete System-on-Chip. Solutions have been proposed to allow such IP cores to be re-used in multiple designs, and it is instructive to look at these approaches.

4.1 Virtual Socket Interface Alliance (VSIA) – Virtual Component Interface (VCI)

A principle of the VCI [10] is that, by separating the interface logic and the behavioural logic of the core, one could simplify the difficulty of interfacing cores to different bus architectures [11]. As shown in figure 4, the interface logic is abstracted from the core design, where traditionally it is integrated inside the design. The abstraction of the interface behaviour logic is developed into a Bus Wrapper that interfaces to standard bus architectures. The VSIA defines a standard interface between the behavioural logic and the bus wrapper called the Virtual Component Interface (VCI).

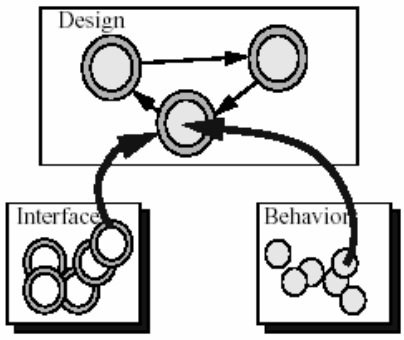

Figure 4. The Abstraction of Interface from Design (Source [11])

The advantage of this standard interface is that VCI has a simpler protocol then standard bus interfaces, and the specification of VCI remains relative stable through time. The separation of interface and behaviour logic provides flexibility because IP-designers can simply mix-and-match the appropriate bus wrapper, to a single behavioural logic design of a core. When new bus standards are introduced, the behavioural part of the logic

can be reused and a new VCI standard bus wrapper is to be designed to interface with the new bus protocol. The intention is that the reusability of IP cores can be greatly improved.

In many ways, the goals of the VCI standard are very similar to those of this research project, except that we extend the idea of a Bus Wrapper to include many other communication architectures.

4.2 Standard Bus Interface

An alternate standards-based approach is to specify a fixed bus protocol that is used in many designs. An example is the AMBA bus standard mentioned earlier. A major benefit for IP designers who design logic cores that directly interface to the standard bus-architecture is that timing between the IP core and bus are more accurate and precise. This can be a crucial factor for real-time applications. By contrast, the more flexible bus wrapper approach introduces some undesirable protocol conversion overheads [12].

Although this can be eased with CAD tool support, such as the SOPC tool described earlier, designers still have to understand the complex bus architectures and protocol. It is also these CAD tools only support their own reconfigurable platform.

4.3 IPIF

Xilinx' OPB IPIF [13] is a simplified form of bus wrapper specifically designed to ease interfacing to the IBM CoreConnect bus used by Xilinx' MicroBlaze-based reconfigurable SoC.

The OPB IPIF is a parametric soft IP core and available through Xilinx's CoreGen flow. At this stage it is still a preliminary design. The OPB IPIF reduces the repetitive process of connecting IP cores to the OPB bus by providing common components. These common components are Address Decode, Interrupt Control, Read Packet FIFOs, Write Packet FIFOs, DMA and Scatter Gather.

Shown in figure 5 is an OPB device that uses the full set of IPIF Features. The advantage of using IPIF glue logic is IP designers do not have to learn the complex OPB bus protocol, all they have to do is chose the required functionality and know how to attach their own IP core to the IPIF. Since it is a parametric soft core, the IP designer can choose only the required functions and eliminate the unwanted components to optimise the OPB peripheral. Shown in figure 6 is an example of an IP device that uses only Register and Address Decode functions.

The major disadvantage of this tool is that the OPB IPIF module targets only CoreConnect OPB and PLB bus interfaces and only Xilinx reconfigurable platforms. Core re-use with other busses, other brands of FPGAs, or other network structures would require many more additional IPIF-like tools to be written.

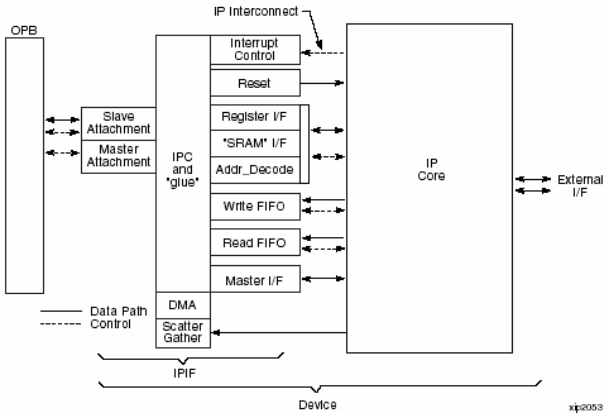

Figure 5. An OPB Device Using the Full set of IPIF Features (Source: [13])

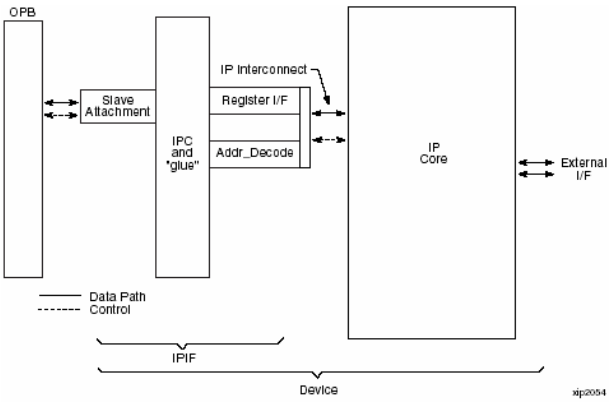

Figure 6. An OPB Device Using Only Some of the IPIF Features (Source: [13])

Our proposal builds on the fundamental idea behind the OPB IPIF approach, which is to automate the interfacing of core peripheral logic to the system bus. However, in our case we wish to extend this automated interfacing to include a larger number of possible system interconnection networks.

5. Proposed Interface Methodology

Inspired by interface-based design, VCI's Bus Wrappers and Xilinx's IPIF, we propose an interface methodology that will separate interface design and behavioural design, to allow ease of interfacing of IP core to a wide variety of communication protocols, and make IP core significantly more reusable. The advantage of the proposed methodology is that we aim to develop an

interface logic (glue logic) that will have fewer overheads than VCI Bus Wrappers, and also vendor independent, unlike Xilinx's IPIF module that only support Xilinx platforms. Section 5 and 6 will explain how this idea is possible.

In order to separate IP interface logic from the rest of the IP core, the total peripheral can be separated into three distinct parts (Refer to figure 7), the Interface Adaptor Logic, the IP core and external I/O interface.

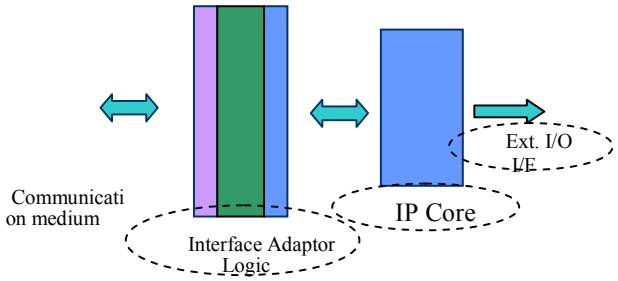

Figure 7: Methodology Separation of IP Design

Interface Adaptor Logic

The interface adaptor logic (IAL) is responsible for the correct communication between the IP core and the system interconnection network. Because it separates IP core and the communication logic, It allows the IP core to be designed without any prior knowledge of the complex communication protocol. We require the IAL not to introduce unnecessary protocol conversion operations, as is often the case with Bus Wrappers. The IAL should have a similar low overhead to that of Xilinx' OPB IPIF, ie. it provides only circuitry that is necessary to connect to the specific communication architecture.

In one aspect, the Interface Adaptor Logic provides a set of common features that IP core might use. This set of features includes anything from DMA control to memory mapped registers, and the set is expendable to support more unique features. The other aspect of the Interface Adaptor Logic interfaces with the system interconnection network protocol. If a bus architecture, like CoreConnect bus, consists of three different buses, the OPB, PLB and DCR, there will be three different Interface Adaptor Logics each targeting an individual bus type.

Therefore if there are "*n*" protocols that require to be interfaced with, there will be "*n*" different interface logic modules required. The Interface Adaptor Logic is not meant to be one very complex, very flexible circuit, but rather a number of smaller, network-specific designs. This will simplify the complexity of the adaptor and it will be easier to upgrade and edit in the future.

Once an IAL has been designed for a particular interconnection network, then all IAL-compatible cores in a library immediately become available for use with that communication structures.

This allows both the most appropriate peripheral designs and the most appropriate interconnection networks to be used at any point in the System-on-Chip. This contrasts with common practice where the availability of cores for a particular interconnection strategy often determines the type of bus or other communications architecture used.

The IP Core

The IP core is designed as an independent logic block that performs only the functionality of the IP application. For IP core to communicate with the protocol, interface logic will be integrated into the core as a component. In our first version of the IAL, it will be the designer's responsibility to choose the required IAL functions that the IP core needs, and manually map the interconnections between core and adaptor ports, as shown in figure 8.

After configuration, the adaptor logic is fully integrated within the IP core, providing the IP with the required interconnection functionality.

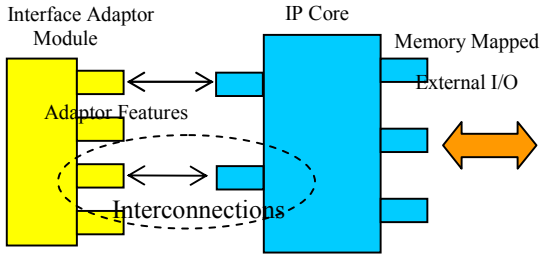

Figure 8: Communication Between IP and Adaptor Logic

External I/O

External I/O ports of the IP are the physical connection from the reconfigurable platform to the external device pins (refer to figure 8). Current FPGA CAD tools provide good support for this stage of design, and we do not yet intend to work on a vendor-independent way of automatically generating these connections.

6. Implementation

To make the above methodology possible, we will now discuss how the Interface Adaptor Logic should be implemented, the setting up of the IP core to interconnect properly with interface logic, and how to integrate adaptor logic to the IP core.

6.1 Designing Interface Adaptor Logic

As shown in figure 9, the Interface Adaptor Logic can be internally divided into to three parts: the interface to communication medium, the feature interface to the IP

core, and the internal logic that converts between these two interfaces.

The interface to the communication protocol will contain logic and signals that connect to all the protocol interface signals. These signals could be address bus, data bus interrupt signals etc.

The feature interface to the IP core will contain signals that allow access to commonly used features, such as Data Registers, FIFOs, DMA, and Interrupts. Depending on the requirement of the IP, some will need more features then others. To control and synthesise only the required features, the IAL will need to be parameterisable, as described in the next section.

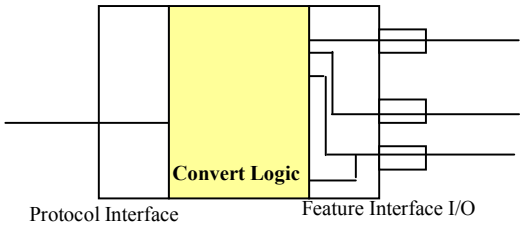

Figure 9: Distinction of Interface Adaptor Logic

Our current research effort is going into understanding the set of features that need to be provided by the IAL, so that a large number of different peripherals can efficiently interface via a single IAL module. As a starting point, we are aligning our IAL interface with the feature interface of Xilinx' OPB IPIF interface.

The internals of the interface adaptor logic will perform data formatting and conversion between IP core data signals and communication protocol signals. If a One-Wire serial communication architecture is used with a parallel-data peripheral, then the IAL will need to do parallel-serial conversion. In packet-based architectures, such as ATM, the IAL will need to de-fragment packets and analyse them before redirecting to the IP core.

6.2 Integration of Interface Adaptor Logic and IP core

In order for an IAL for a particular communication protocol to be able to effectively connect to a large number of different types of peripherals, the IAL must provide a full set of possible interconnection features. These include data registers, status registers, interrupt control, DMA control, busy/ready flags, address decoding, variable register word sizes, and FIFO controls.

However, many peripherals will require only a subset of these features. A fixed logic IAL which always provides all these features will be inefficient and introduce significant overhead. Avoiding overheads was the reason for avoiding the VCI Bus Wrapper approach.

The IAL is then required to be parameterisable, to efficiently accommodate different peripherals. Each instantiated IAL will only have the features required to communicate most effectively between peripherals and networks. Register word widths need to be variable, and the IAL will need to deal with any necessary word width changes, serial/parallel conversions, etc. Information about peripheral addresses (either on address busses, or in packet headers) needs to be decoded by the IAL

At least initially, users will manually provide parameters to an IAL Generator, which will provide the necessary logic netlists that describe the customised IAL. Currently C programming language is chosen to implement a parameterisable IAL.

The C program takes as input a specification of the required features for an IAL, and outputs a simple, structural VHDL program. If features are not needed, then the VHDL structures corresponding to those features are simply not output. Figure 10 provides an example.

```
…….
if DMA is true then
        printf " dma_1: DMA ";
        printf " port map (";
        ……
else
        printf "……..";
```

Figure 10: C Programme Printing VHDL Code

The advantage of this approach is that, it allows generation of required logics only, and hence maximise the performance of the IP core.

6.3 Connecting to the IP Core

In our first version of the IAL generator, the IAL will be a separate, customised VHDL module. This will be manually incorporated into a design as part of the normal, vendor-supported CAD flow. The designer may need to do port renaming to ensure correct connection of the IP core to the IAL, particularly if there are several identical copies of an IP core in the design.

7. Discussion

Current methods of interfacing IP cores to system buses have their own problems. The VCI introduces unwanted overheads, and although IPIF module has fewer overheads, it only targets their own platforms. An interface methodology has been proposed aiming to ease the interface process and resolve the problems that current methods are having.

The interface methodology attempts to separate the interface and the function of the IP, in order to eliminate the complex interfacing process for custom and third-

party IP cores on to SoC integration platforms. The methodology particular addresses reconfigurable System-on-Chip. The interface adaptor logic provides an automated interconnection between the IP core and different communication architectures.

Complex calculation of the most appropriate interfacing structures for a particular parameterised IAL design will greatly reduce protocol conversions and overheads that are part of a simple Bus Wrapper approach. Time critical peripheral functions will benefit since only minimal overheads will be introduced. The parameterised capability of the Interface Adaptor Logic will optimise the logic circuit by synthesising only the required components.

In the future, once the IAL methodology has been investigated and proven, it can be incorporated into a more automated CAD tool. This CAD tool will allow designers to mix-and-match different IP core functions with the most appropriate system interconnection structure.

There are still significant issues to be resolved concerning the methodology. An Interface Adaptor Logic and a generic IP core will take longer to develop then an ordinary IP core with interface incorporated. If a communication protocol has been updated, the adaptor module will need to be modified.

8. Conclusion

This paper has proposed a new methodology that will improve the reusability of IP cores, and promote more complete explorations of the most appropriate system interconnection architectures for individual chips

There are several integration platforms that are available on the market today, each with their unique architecture, and each with their own system-bus protocol. Interfacing IP cores to a particular bus protocol has often been a difficult process, requiring significant time to understand the complex bus protocol. Platform vendors recognise this problem and supply system tools to ease the interfacing process. These tools however only support their own platform system-bus protocol. This new approach, of a more general set of IALs has been proposed to address this situation.

This research is still at an early stage. Currently we are constructing peripherals to interface with the OPB bus. The next major task is to fully specify the feature set that an IAL needs to support, and then compare its efficiency with current techniques. We plan to complete this specification, and have a small set of IAL generators, and generic IP cores available by the end of 2003.

References

[1] Xilinx, "Xilinx Inc," 2003, <http://www.xilinx.com> [accessed 10th Feb., 2003].

[2] Xilinx, "MicroBlaze Soft Processor," 2003, <http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=microblaze> [accessed 10th Feb., 2003].

[3] IBM, "CoreConnect Bus Architecture," 2003, <http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture> [accessed 10th Feb., 2003].

[4] Xilinx, "CoreConnect – A Simplistic Overview," 2003, <http://www.xilinx.com/esp/optical/collateral/CoreConnect.pdf> [accessed 10th Feb., 2003].

[5] Xilinx, "Xilinx ISE Foundation," 2003, <http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+Foundation> [accessed 10th Feb., 2003].

[6] Altera, "Altera Corporation," 2003, <http://www.altera.com> [accessed 10th Feb., 2003].

[7] ARM, "AMBA - The de facto Standard for On-Chip Bus," 2003, <http://www.arm.com/armtech.nsf/html/AMBA?OpenDocument&style=AMBA> [accessed 10th Feb., 2003].

[8] Altera, "Avelon Bus Specification – Reference Manual," 2003, <http://www.altera.com/literature/manual/mnl_avalon_bus.pdf> [accessed 10th Feb., 2003].

[9] Altera, "SOPC Builder," 2003, <http://www.altera.com/products/software/system/products/sopc/sop-index.html> [accessed 10th Feb., 2003].

[10] Virtual Socket Interface Association, Architecture Document, <http://www.vsi.org> [accessed 10th Feb., 2003].

[11] J. Rowson and A. Sangiovanni-Vincentelli, "Interface-Base Design," Proceedings of the 34th annual conference on Design automation conference, California, United States, 1997.

[12] R. Lysecky, F. Vahid, T. Givargis, "Experiments with the Peripheral Virtual component Interface," ISSS 2000, Madrid, Spain, 2000.

[13] Xilinx, "Designing Custom OPB Slave Peripherals for MicroBlaze," 2003, <http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/opb_tutorial.pdf> [accessed 10th Feb., 2003].

[14] RSVP, "Standards Corner – The Design Productivity Crisis and Other Contributing Factors," Spring 1997, <http://www.synopsys.com/news/pubs/rsvp/spr97/rsvp_spr97_7.html> [accessed 10th Feb., 2003].

[15] M. Keating, P. Bricaud, "Reuse Methodology Manual", Kluwer Academic Press, 2nd Edition, 2002