Interfacing methodologies for IP re-use in Reconfigurable System-on-Chip

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

ABSTRACT

Initially, IP cores in Systems-on-Chip were interconnected through custom interface logic. 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. Many solutions have been proposed, however these solutions either limit the IP block performance or are restricted to a particular platform. 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 Adaptor Logic layer.

Keywords: FPGAs, Reconfigurable Logic, System-on-Chip

1. INTRODUCTION

Ever-increasing FPGA capacity and low design productivity has introduced the so-called Design Productivity Crisis [1]. IP (Intellectual Property) 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 [2] 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 (rSoC) designs, based on FPGAs, is the use of platform technology. These platforms utilise a bus architecture as the backbone for system peripherals to communicate with each other. If peripherals (or IP blocks) are designed to communicate with a particular bus standard, it is possible to reuse this IP block on a different platform that has the same bus standard. However, if the platform does not use the same bus standard, the IP block will need to be modified. It can be difficult to extract the functional core of an IP block and retarget it to a different interface protocol.

In this paper, we aim to extend the 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 characteristics:

- IP independence - Separates IP behavioural logic from interface logic to provide IP-reuse.
- Targets diverse system interconnection protocols (buses, serial networks, packet-switched networks).
- Provides low-overhead Interface Adaptor Logic – eliminates 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 – caters 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.
• Vendor independent – IAL generator provides service to multiple rSoC platform vendors.

3. XILINX IPIF MODULE

Interfacing an IP block to a communication protocol has always been difficult. Xilinx developed its IPIF module [3] as a solution for end-users when interfacing custom peripherals (or IP Blocks) to the system bus architecture. On Xilinx’s Virtex II rSoC platform, the bus architecture for peripherals is IBM’s CoreConnect’s OPB and (or) PLB bus. The IPIF module only provides users with OPB and PLB Bus interfacing support; developers need to customise their own interface if targeting other bus standards.

The intention of the IPIF module is to simplify the OPB/PLB interfacing design process that an IP designer needs to go though. This is achieved by IPIF providing a set of interfacing features that common IP blocks require. Figure 1 shows the structure of IPIF module.

![Figure 1. An OPB Device Using the Full set of IPIF Features (Source: [3])](image)

3.1 Structure and Operations

The block diagram of an IPIF module has two sets of interfaces. One side is the communication interface between the IPIF module and the IP functional core, and this interface is called the IP Interconnect (IPIC). On the other side is the attachment of IPIF to the OPB or PLB bus.

Unlike VSIA’s VCI protocol [4], an IPIF module does not convert signals from one bus protocol to another. The aim is to interface the IP functional core to the OPB bus as one would without the use of IPIF. This eliminates any unnecessary conversions and overheads.
The current IPIF module has eight features: slave attachment, master attachment, address decode, interrupt control, read packet FIFOs, write packet FIFOs, DMA and scatter gather. Each of these features has its own control and some shared data signals.

Either the slave attachment or the master attachment must be used to interconnect the IP block with the OPB/PLB bus. The master and slave attachment provide separate address and data buses, single read/write and burst read/write operations (reading and writing block of data). Obviously a master IP block must be configured with a master interface, to gain access to the PLB as a master.

Features such as a DMA controller will need the IP block to be designed as a master peripheral. DMA master operations (reading and writing to memory) do not differ from normal master read and write operations, however can be parameterised to include optional “scatter/gather”, “packet support” and “interrupt coalescing” capabilities.

The register feature interface does not include an address signal. Address decoding process is completed inside the IPIF module. It will assert a “Read Chip Enable” signal to specify a request for a register read. IPIF supports IP blocks with up to 256 registers. The “Read Chip Enable” signal is a vector from 0 to 255 specifying which memory mapped register has been selected for data transactions.

The SRAM feature supports IP blocks with SRAM-like functionalities. Differently to the register interface, SRAM IP blocks need to decode the address on the address bus when the SRAM IP module is selected. Read/write operations will be similar to register read/write operations.

It sometimes is useful to be able to support IP blocks with FIFOs to buffer data. With the IPIF FIFO feature, a developer can choose either a read FIFO, a write FIFO or both. Read and write FIFOs are almost the same, except for the transfer direction of the data.

The interrupt feature interface can support up to 32 interrupt events from the IP block. The IPIF module is capable of combining all the interrupt events inside the IP block into a single interrupt signal for connection to a system interrupt controller. This is done through Interrupt Source Controllers (ISCs).

To use these IPIF features, a designer must first analyse which features the IP block will need, then design the IP block to incorporate the associated feature signals, and feature protocols. Features that were not selected will not be generated. This will optimise the system design.

The next step is to attach the IP block to the IPIF module. In this stage, both the IP block and the IPIF module act as components inside the final OPB IP core (or PLB IP core). By “mapping” IPIF signals to the IP block signals, the IP block will gain access to the selected IPIF features as well as the interface to the OPB/PLB bus.

To complete the attachment of IPIF and IP block, the developer must initialise parameter values in the IPIF. Parameters are set to enable/disable features, select/deselect other optional capabilities, as well as setting the base address of feature control register offsets. For example, once DMA is activated with the C_DMA_PRESENT parameter, 4 packet support IPIF I/O signals will be enabled and ready for the IP block to “map” appropriate signals to it. Similar processes are used for activating other features.

The IPIF module is designed to simplify interface design for OPB/PLB buses and is not intended to improve the mobility of an IP core to different bus architecture. One obvious example is the FIFO data bus. The data bus is restricted to 32 bits, which is the same as the OPB data bus width. This restriction will affect the portability of the IP core to a system that utilises a different bus width.

In addition to bus architecture restrictions, IPIF is a vendor dependent module: rSoC platforms other than Xilinx supported platforms cannot use this service. Our research aims to provide a method that is similar to the IPIF concept, however independent of vendor platform, as well as targeting non-parallel bus architectures.

4. IAL – INTERFACE ADAPTER LOGIC

Inspired by interface-based design and Xilinx’s IPIF module, we propose an interface methodology that will separate interface design and behavioural design, to allow ease of interfacing of IP cores to a wide variety of communication protocols, and make IP cores significantly more reusable.
The aim of this research is to design an interface generator, like IPIF, that can interface a custom IP block to many different buses or communication schemes. The logic that the interface generator generates is called an IAL (Interface Adapter Logic) [5].

For every communication medium there will be a separate IAL generator developed. Although different IALs target different communication media, the interconnection between IAL and IP blocks have to be consistent. This consistency will improve the reusability, by removing the need for modifying legacy IP blocks.

Currently, we are examining different communication schemes that can be used to implement rSoC systems, and these communication schemes can be broken into several categories: parallel buses, the serial buses, and network architectures.

**Parallel Bus**

Parallel buses such as IBM’s CoreConnect [6] PLB/OPB and AMBA’s AHB/ASB/APB buses [7] provide parallel data and address bus structures. Some data buses can be configured to 16, 32, 64 up to 256 bits wide, depending on system requirements. Both CoreConnect and AMBA support burst and single read and write data operations. These two protocols support multiple master and slave devices.

The main advantage of using a parallel bus is its fast data transfer rate. Compared with serial buses, a parallel bus can transfer many bits with one clock cycle. The parallel bus architecture however is a shared medium. The problem for any shared medium type of communication architecture is that, as the number of components increases inside a system, the bandwidth resource per component are reduced.

**Serial Bus**

Generally serial bus protocols are simpler then parallel bus protocols. The number of acknowledgments and timing constraints are not as many or as strict as parallel protocols, however bandwidth is reduced compared to a serial bus.

I²C is a serial bus with only serial data (SDA) and clock (SCL) lines [8]. It supports 7 (and/or 10) bit addressing with the 8th bit as a read or write conditional bit. The data must be 8 bits wide and address and data transfers use the same serial data line. I²C is a multi-master bus, supporting multiple masters and slaves, with masters initiating read and write operations. The I²C bus protocol provides error detections, such as collision detection and arbitration to prevent corruption of data when two or more masters are trying to transfer data simultaneously. The disadvantage of I²C is the slow serial transfer rate. It is also a shared-medium architecture.

**Communication Based Networks**

Several researchers are investigating on-chip communication schemes drawn from Local Area Network technologies such as Ethernet or ATM. Our initial investigation focuses on one example of such networks – packet-switched networks. Our first version of a packet switched network treats the system as a series of unidirectional serial lines connected to packet switch nodes and transmitter/receiver nodes. Each serial line will use a protocol similar to the serial bus format described earlier.

### 4.1 IAL to Communication Medium Interfacing Signals

Before implementing an IAL generator, we need to examine the bus characteristics and their interfacing signals. The following section analyses three bus structures, one from each category mentioned in the previous section.

**Parallel Bus - OPB**

Figure 2 shows the OPB slave interface connection for an OPB Slave peripheral and figure 3 shows the master interface signals. Not all the signals are compulsory. Depending on the functions of the slave peripheral, some optional signals can be omitted.

The OPB master and slave are synchronous devices. The OPB Bus has an OPB clock signal, generated from the system clock which is distributed to peripherals inside the system.

Each of the bus interface signals (address, data and control signals) is a unidirectional signal, hence the OPB bus architecture has separate slave and OPB data buses. Because only the master peripheral initiates transactions, there is only one address bus. The OPB bus can support two addressing and data transfer modes, 32 or 64-bit, however, it does
not mean that the address bus has to be the same width as the data bus. For example, the address bus can be 32-bit wide while the data bus can be 64 bits wide. IBM’s CoreConnect adopts big-endian operations, meaning bit 0 is the most significant bit and bit 31 or 63 is the least significant bit. This has to be taken into account when designing IAL circuits for CoreConnect and other buses.

The acknowledgement and control signals are required to ensure correct transactions between master and slave peripherals.

Serial Bus - \( \text{I}^2\text{C} \)

\( \text{I}^2\text{C} \) is a serial bus protocol. Unlike parallel buses, only two wires are used to transfer data and detect errors. Figure 4 shows a system that incorporates \( \text{I}^2\text{C} \) as the system bus [8]. As evident from figure 4, \( \text{I}^2\text{C} \) is a shared medium with one wire as the serial data bus (SDL) and the other wire as the serial clock (SCL).

Master devices must generate their own clock signal and send the clock signal on the SCL line. The SCL line will then generate the system clock’s Low period with longest “LOW” from master clocks, and the high period with shortest “HIGH” from master clocks.

Combinations of signals carried by these two wires set up the data transactions, acknowledgments and error detections. \( \text{I}^2\text{C} \) supports multi-master systems, with the peripheral that initiates transfers as the master, and the peripheral receivers as the slave. Figure 5 shows the packet structure for a master requesting write operation, and Figure 6 shows master requesting read. During the transmission of a byte of data or an address, the MSB (bit 7) is the first bit sent and LSB is the last bit sent. One special situation exists where the slave device can hold the SCL clock line LOW in the middle of a transaction, making the master “wait”, when the slave needs to perform some other functions, for example an internal interrupt routing. The slave device will release the SCL line when it is ready to receive data from the master.
Communication Based Networks – Packet-Switched Network

Network based communication architectures have not yet been widely implemented on SoC platforms. The most popular communication schemes are still parallel and serial bus type architectures. A companion project at the University of Queensland is investigating an on-chip packet switched network architecture [9]. One proposed architecture from our research group is shown in figure 7. The architecture will be based on serial connections and an ATM-like architecture.

4.2 IAL Supported Features and Required Signals

Similar to Xilinx’s IPIF module, the IAL will provide predesigned features that a designer can use depending on the functionality of the IP block. The IAL will be targetted for a reconfigurable system on chip (rSoC) platform. We take advantage of the platform’s reconfigurability to generate optimal IALs for different IP blocks. Table 1 shows the possible features that could be implemented on the IAL, figure 9 shows the architecture of the IAL and figure 10 shows the Adaptor Logic Connection (ALC).
<table>
<thead>
<tr>
<th>Grouping</th>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td>Address Decode [i]</td>
<td>The Address Decoder decodes whether the master IP block is sending a request to this IP slave block. It further decodes the register address and asserts the corresponding “Register_Select[i]” signal to the IP block.</td>
</tr>
<tr>
<td><strong>FIFOs</strong></td>
<td></td>
<td>User can select “Read FIFO”, “Write FIFO” or both for data buffering.</td>
</tr>
<tr>
<td><strong>Interrupt_In, Interrupt_Out</strong></td>
<td></td>
<td>Determines whether interrupts are asserted for signals “In_Data_Available” and “Out_Data_Sent” from the IP Block.</td>
</tr>
<tr>
<td><strong>AutoSend</strong></td>
<td></td>
<td>Determines whether data is automatically sent as soon as it is ready, or whether transfer waits for a request from a Master.</td>
</tr>
<tr>
<td><strong>Read Action</strong></td>
<td></td>
<td>Determines actions that are taken when data is read from the IP block (eg. interrupts deasserted, data cleared from register)</td>
</tr>
<tr>
<td><strong>Serial Communication Medium (Compulsory)</strong></td>
<td>Serialiser/Deserialiser</td>
<td>Serialiser and Deserialiser are compulsory features for Serial Bus Architectures, to convert from parallel register data transfers from the IP block.</td>
</tr>
<tr>
<td><strong>Packet Switch Network (Compulsory)</strong></td>
<td>Packet Size/Format</td>
<td>The format of the packet needs to be specified for generating Packet Assembler and Disassembler logic.</td>
</tr>
<tr>
<td><strong>Packet Assembly/Disassembly</strong></td>
<td></td>
<td>Because packets contain both address and data information, the packet disassembler needs to extract the address and data from the packet and forward to address decoder and data port respectively. The Packet Assembler assembles information together into correct packet format, and transmits to the switches.</td>
</tr>
<tr>
<td><strong>System Signal</strong></td>
<td>BUS_Clock</td>
<td>Provides synchronisation and clock signals for peripheral blocks, decoded from bus or system clocks.</td>
</tr>
</tbody>
</table>

Table 1: Preliminary set of IAL features.

Figure 9: IAL architecture.
4.3 IAL to IP Block Interface – Adaptor Logic Connection (ALC)

The Adaptor Logic Connection (ALC) is the name given to the connection between the user-designed IP block, and the automatically generated IAL block. Data transfer is parallel, chosen for its higher data throughput. The following sections describe the individual ALC signals, as shown in figure 10.

![Adaptor Logic Connection (ALC) signals](image)

**DATA_IN, DATA_OUT**

The IAL has separate data input and data output buses to support high data throughput. The data width can be specified by the IP block developer when they design their IP functional block. Data width will be set as one of the parameters to generate the IAL. Width of the data buses can initially be selected from 8, 16, 32 or 64-bit width, and in big-endian format.

**REGISTER_SELECT[i]**

Register_Select is the signal from address decoder to indicate which register has selected. If a particular IP block requires only five registers, IAL will only generate Register_Select[0] to [4]. At this stage, we are not considering peripherals with RAM-based data transfer mechanisms.

**RNW**

ReadNotWrite is a single bit indicating whether the selected register is being read or written to.

**In_Data_Available**

An IAL signal indicates that received data is available inside the IP Block and is ready for the IAL to read.

**Out_Data_Sent**

Is an acknowledgment signal sent by IP block to IAL indicating that previous data has been transmitted or otherwise processed.

Implementation of several different IP block peripherals in the near future will allow us to further refine this ALC specification.
5. DISCUSSION

Current methods of interfacing IP cores to system buses have their own problems. The Xilinx IPIF module only supports Xilinx platforms and OPB/PLB bus standards. An interface methodology has been proposed aiming to ease the interface process and resolve some of these problems.

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 onto SoC integration platforms. The methodology particularly 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 than an ordinary IP core with interface incorporated. If a communication protocol has been updated, the adaptor module will need to be modified.

6. 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 SoC 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.

A preliminary IAL specification has been proposed in this paper. The next major task is to implement our first version of the IAL generator software. We plan to update this specification, and have a small set of IAL generators, and generic IP cores available in the near future.

REFERENCES


