2011 International Conference on Information and Electronics Engineering
IPCSIT vol.6 (2011) © (2011) IACSIT Press, Singapore

Soft-IP Interface Modification Methodology

Xiaozhou Meng, Benny Thörnberg and Najeem Lawal

Department of Information Technology and Media Mid Sweden University Holmgatan 10,

Sundsvall, Sweden

{xiaozhou.meng, benny.thornberg, najeem.lawal}@miun.se

Abstract. The reuse of predefined Intellectual Property (IP) can lead to great success in system design and help the designer to meet time-to-market requirements. A soft IP usually needs some customization and integration efforts rather than plug-and-play. Communication interface mismatch is one of the problems that integrators often meet. This paper suggests a soft IP interface modification methodology (SIPIMM) for systems on Field Programmable Gate Array (FPGA). SIPIMM targets an interface-based soft IP model which is introduced to ease the interface modification and interface reuse. A case study of an open-source IP is presented using SIPIMM for system integration.

Keywords: soft IP, FPGA, interface mismatch

1. Introduction

It is widely recognised that reuse and sharing IP is becoming fundamental to closing the deep sub-micron design gap for successful system-on-chip (SOC) design [1]. Digital IP is the most popular form for design reuse in today's industry, which can be divided into three categories: soft, firm and hard [2]. If an IP core is truly reusable, it must remain untouched as it moves from system to system. Hard and firm IPs might have the ability to plug-and-play within a single technology. Soft IP is more portable but usually need some customization and integration efforts [3].

The soft-IP portability issue is described in figure 1. It also presents an example case further elaborated in section 5. The IP integrator chooses the IP with the right functionality and performance and then imports it into their IP component library. IP components can be bought from IP vendors like FPGA vendors, third-party IP suppliers or internal IP providers. The integrator will get support and guarantee from the providers. Another possible cost saving choice is open source based IP. IP can be downloaded for free but with little guarantee. The users might spend a lot of efforts to make it work since they get very limited support from the provider.

Designers face difficulties when an IP needs to be integrated into the target system. Several factors such as choice of FPGA vendor, tool/library and communication interfaces should be considered. Each FPGA vendor has its own technology and fabrics. Tool/library can be differing a lot. Moreover, communication protocols between IP cores are diverse. All together, these factors generate lots of obstacles for the IP portability.

The aim of the work presented in this paper is to propose a methodology for an increased IP portability along the Communication axis shown in figure 1. We present a soft IP interface modification methodology (SIPIMM) which will result in an efficient reuse of soft IPs.

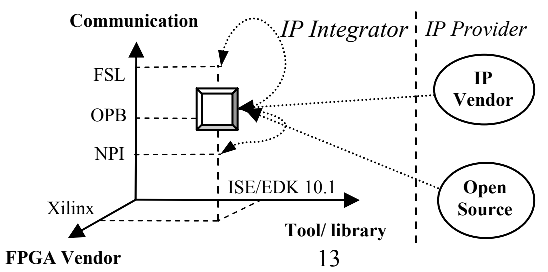

Fig. 1: Soft-IP portability issue for IP integrator

2. Related work

The WISHBONE SOC interconnection architecture [4] creates a common interface between IP cores which is used by many designs in the OpenCores [5] project. An interface-based design methodology [6] is proposed which separates communication from behavior of the core. The focus of such a method would be on how the modules interface with each other. SPIRIT consortium [7] proposed the IP-XACT description which is an XML-based SOC meta-data specification as a standard to describe the components of a SOC platform. IP-XACT components can be assembled into a SOC platform directly using IP-XACT compliant tools, which greatly improves the interoperability of SOC platform.

3. Communication interface mismatch

The idea of plug-and-play IP is the goal for IP reuse in today's industry; however, some obstacles lengthen the path to this goal. Communication interface mismatch is one of them.

Different On-Chip bus standards are defined by several leading companies and associations. There are mainly two types of communication interfaces: (1) Standard bus interface, which uses standard communication protocols. This type of interface is easy to connect because of its regularity. (2) Custom bus interface, which represents customizable communication protocols. It can be defined by the designers for some special purposes.

Different bus interface protocols are not compatible with each other. It seems hard to create a uniform bus standard for SOC interconnection. In addition, the bus standards are evolving such that a previously developed IP might have an earlier version of interface protocol. For a hard or firm IP, backward compatibility is not always guaranteed unless integrator adds an extra hardware bridge to plug it into the new system. Due to the flexibility of soft IP, the interface protocol can be modified to match the interface requirement of the new system.

4. Methodology

To make efficient reuse of IP and its interfaces, an interface-based soft IP model is introduced. Based on this model, the methodology SIPIMM is proposed to ease the communication interface mismatch problem.

4.1. IP verification

When the integrator get the IP, it is necessary to know its usage history. From a functional point of view, whether the IP has been proven in silicon or not is valuable information to know. This detailed information includes in which device or configuration it was used and what the success rate is and so on. This is useful information which will help the integrator to avoid the detours.

After this investigation and depending on its outcome, an ambitious functional verification of the IP core should be carried out. This step can for example be particularly important for an open-source IP having very limited guarantees. Any test bench or verification methodology from the IP provider will ease this work. Otherwise, this task can be tough and time consuming, especially for configurable IPs.

4.2. Interface-based soft IP model

Soft IP core has the possibility to communicate with an external system. The on-chip system includes on-chip bus, other system components or peripheral. The core can be an initiator (master) core or a target (slave) core depending on the implementation. For the interface-based soft IP model, several boundaries inside the core should be clarified, shown in figure 2.

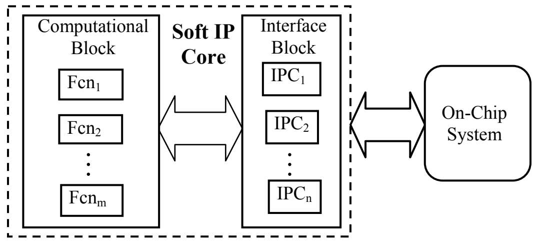

Fig. 2: A generalized interface-based soft IP model

1) *Isolate the computation logic from the interface logic:* The soft IP core can be mainly divided into two blocks: computational block and interface block. The *Computational Block* contains the main logic functions of an IP. The *Interface Block* implements all IP communication divided into one or several Interface Protocol Cells (*IPC*) as well as various ports for dataflow, control signals and debug signals. The *Computational Block* has a higher chance of being re-used since the *Interface Block* might be changed frequently due to different system configurations. This separation of computation from communication is effective for IP database management. It is a suggestion to the IP providers for creating reuse of IP in SOC Design [8].

2) *Boundaries between functions:* Different functions in the *Computational Block* should have clear boundaries (Fcn_1 to Fcn_m). It might occur that the integrators only need parts of the functions of a *Computational Block*. A reduced *Computational Block* must then be working correctly after removing some of the unnecessary functions.

3) *Boundaries between IPCs:* When using multiple interfaces, each *IPC* should be independent (*IPC_1* to *IPCn*). This also prepares the way for selective reuse in the future design.

4.3. Interface modification

1) *Interface selection:* Before designing the *Interface Block*, the integrator should already have enough investigation to make a decision on which kind of communication interface is most suitable for the design specification. When the core requires multiple interfaces, the *Interface Block* is just the assembly of different IPCs. *IPC* can be predefined and reused. This method for interface reuse can save a lot of time since the designer only needs to change some parameters without compromising the functionality of the IP core.

2) *Interface design:* One or more IPCs might be designed if they are not predefined.

- IPC design: *Computational Block* is a set of *Dummy Functions* in this case. See figure 3a. This set of dummies can be any synthesizable simple logic or test vector generator. Simulate the IP that contains the *Dummy Functions* and the *IPC* using a simple test bench. After that, synthesize the RTL code and write software application. Implement IP verification in the FPGA prototype. Because the *IPC* is aim for widespread reuse block, verification must aim for zero-defect.
- IPCs assembling (multiple interfaces): Iteratively assemble and verify the designed or predefined IPCs one by one, until the whole *Interface Block* is verified through simulation and hardware prototyping. A model of an assembled *Interface Block* with *Dummy Functions* is shown as Figure 3b.

3) *Interface mapping:* Remove the original *Interface Block*. Map the output of the *Computation Block* to the input of the newly designed *Interface Block*. Write a test bench to verify the functionality of the complete IP as depicted in figure 2.

4.4. IP integration

Synthesize the RTL code. Write the software driver and test software for functional verification of the IP core. Integrate the whole IP into the system by connecting it to a simplified SOC for functional verification. Run the test software for the entire system. Evaluate the IP function and performance to finalize the design.

5. Case study: interface modification of M-JPEG decoder

The motivation for this part of our work was to implement an M-JPEG decoder into a Video Transmit over IP (Internet Protocol) system and to analyze the usage of the methodology SIPIMM. We started this work from an M-JPEG decoder project [9] that was downloaded for free from OpenCores. This VHDL project comes with an earlier version of the bus interface OPB (On-chip Peripheral Bus) [10]. The video output directly connects to the monitor which does not meet our system specification. Figure 4 shows the

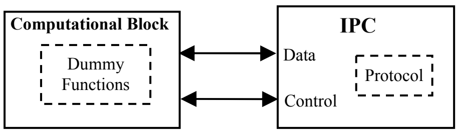

Fig. 3a: IPC design with a dummy function in the computation block.

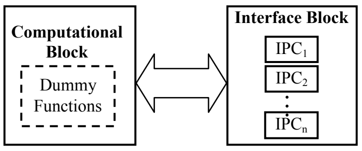

Fig. 3b: Interface design with dummy functions in the computational

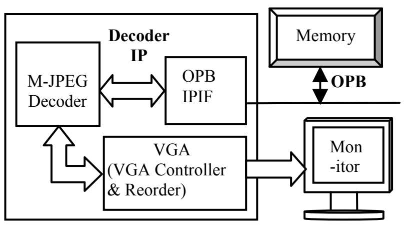

Fig. 4: Block diagram of original decoder project.

architecture of the original decoder system. *The Decoder IP* behaves as an OPB master on the bus and can fetch compressed JPEG data from *Memory*. The uncompressed RGB data is directly output to the *Monitor*. Three main blocks are involved: *OPB IPIF* is the OPB bus interface protocol. *M-JPEG Decoder* decodes the compressed JPEG data and outputs uncompressed RGB data in a sequence corresponding to a series of Minimum Coded Units (MCU). *VGA* reorders the MCU-wise data to line-wise data for output. The *VGA Controller* can be considered to support a communication protocol for the monitor interface.

For the original project from OpenCores, the FPGA vendor and the device family was fixed to Xilinx Virtex II pro. This limitation matches our initial system requirements. However, due to the change of development environment, we were facing a communication mismatch requiring modifications of the interfaces from OPB to Fast Simplex Link (FSL) [10] and from VGA monitor output to Native Port Interface (NPI) [10]. Next parts of this section describe the work of modifying these interfaces using the proposed methodology SIPIMM.

5.1. IP verification

Firstly, the functionality of the original decoder IP was verified by simulation using the test bench supplied as part of the decoder project. After that, we implemented the original system of the project on our FPGA board. The IP functionality was verified using the original development environment of the project.

5.2. Interface-based soft IP model

The computation and interface boundaries were vague in the original IP structure. *VGA Controller* (interface protocol) and *Reorder* (computational) were blended. It represented low portability and was hard to migrate to other system with different functions and different bus interfaces. In order to integrate the core into our system and increase its portability, several modifications to its structure were performed.

For this work, we defined the interface-based soft IP model shown in figure 5. The *Reorder* function was made into a part of the *Computational Block* and the *VGA* function was removed from the *Decoder IP*. The Interface Block contains two IPCs for streaming of compressed and uncompressed video. There is clearly a distinct separation of computation from communication in the developed interface-based soft IP model shown in figure 2.

5.3. Interface modification

1) *Interface selection:* According to the investigation and system specification, the FSL and NPI interfaces were selected to replace the OPB and VGA interfaces. The *On-Chip System* feeds the decoder IP with compressed video through the FSL interface at *IPC*$_1$. The decoder IP outputs data to the system through the NPI interface at *IPC*$_2$. The NPI interface was selected for its low latency and high bandwidth target to memory which is ideal for real time video communication. FSL and NPI are custom interfaces and not predefined, so the two IPCs must be designed.

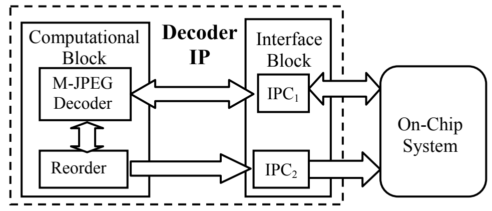

Fig. 5: Interface-based soft IP model for an M-JPEG decoder.

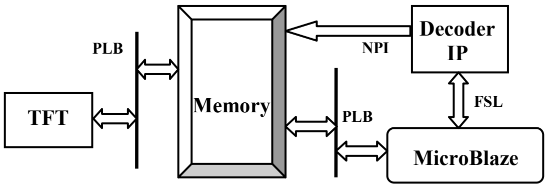

Fig. 6: Block diagram of complete SOC including the updated decoder IP.

2) *Interface design:* The FSL and NPI interfaces were designed in accordance with figure 3a. The *Dummy Function* was designed as a clock counter for verification of both interfaces. After simulation and prototyping of both FSL and NPI, these interfaces were merged into a single interface block in accordance with figure 3b. The *Dummy Functions* were now selected as a copy of input data from FSL to output on NPI.

3) *Interface mapping:* The reorder function component was connected to the M-JPEG decoder component output. The FSL interface was mapped as an input port to the decoder function. The NPI interface was mapped as an output port from the decoder function. Using a debugger, we could verify an image that was decoded in hardware and written to the system's memory through the NPI output.

5.4. IP integration

The Decoder IP shown in figure 6 was integrated with a complete SOC consisting of a *MicroBlaze* soft processor core, *Memory* controller, a *TFT* controller for video displaying and other components not shown in the figure. The *MicroBlaze* processor feeds the *Decoder IP* with compressed video through *FSL*. Decoded video is written directly to memory by the decoder through *NPI* and into the same video memory area as also read by the *TFT* controller on the Processor Local Bus (*PLB*) [10]. The graphics displaying for this SOC was visually verified running the system on a prototyping board.

6. conclusion

This paper presents a methodology for the modification of soft IP interfaces. This methodology is based on that computation is divided into functional units which have a distinct separation from communication. The benefit is clearly the increased communication portability for soft IP. In addition, the increased reuse of design works leads to a reduced work load for the IP integrator. A case study is presented to show that this methodology can be efficiently applied on a real-world design. There are lots of essential issues for IP reuse except for interface mismatch. IP design is a mixed topic with technical, financial and legal issues.

The future work includes using the SIPIMM for more IPs, as well as extended the implementation to other vendor devices.

7. References

[1] N. Bray. Designing for the IP supermarket. In: Fall VIUF Workshop, 1999. pp. 8

[2] M.Keating and P.Bricaud. Reuse Methodology Manual: For System-on-a-Chip Designs, 3rd ed. Boston, MA:Kluwer, 2002.

[3] Doh-Kyung Kim, Ki-Won Kwon, Jong-Chan Choi and Chul-Dong Lee. Reusable intellectual property cores in PC data protection ASIC design. In: AP-ASIC '99, pp. 278-281.

[4] Specification for the: WSHBONE System-on-Chip(SoC) Interconnection Architecture for Portable IP Cores Revision:B.3, Released: September 7, 2002

[5] OpenCores, http://www.opencores.org

[6] James A.R, Alberto S. Interface-based Design. In: Proc. Of 34th DAC, 1997, pp.178.

[7] The SPIRIT Consortium, IP-XACT Handoff Notes, http://www.spiritconsortium.org

[8] Pierre J.Bricaud. IP Reuse Creation for System-on-a-Chip Design. In: Proceedings of the Custom Integrated Circuits, 1999. pp.395-401.

[9] (M)JPEG Decoder, http://www.opencores.org/project,mjpeg-decoder

[10] Xilinx, http://www.xilinx.com