A Structured Methodology for System-on-an-FPGA Design

P. Sedcole, P.Y.K. Cheung, G.A. Constantinides, and W. Luk

Imperial College, London SW7 2BT, UK

Abstract. Increasing logic resources coupled with a proliferation of integrated performance enhancing primitives in high-end FPGAs results in an increased design complexity which requires new methodologies to overcome. This paper presents a structured system based design methodology, centred around the concept of architecture reuse, which aims to increase productivity and exploit the reconfigurability of high-end FPGAs. The methodology is exemplified by the Sonic-on-a-Chip architecture. Preliminary experimental investigations reveal that while the proposed methodology is able to achieve the desired aims, its success would be enhanced if changes were made to existing FPGA fabrics in order to make them better suited to modular design.

1 Introduction

Field Programmable Gate Arrays are an increasingly attractive choice for highly integrated digital systems due to their significantly lower design costs when compared to semi-custom integrated circuit design. The trade-off in FPGA-based systems is an increase in unit cost and a degradation of performance (power, speed, size), which vendors mitigate by embedding into the FPGA silicon performance enhancing primitives such as memories and multipliers. Inevitably, the increasing transistor density and heterogeneity of FPGAs leads to a complexity challenge necessitating new design methodologies to increase productivity.

This paper presents a *structured system* design methodology for FPGA based system-on-a-chip development, based on a paradigm of architecture reuse; instead of designing a monolithic FPGA configuration by connecting together blocks of predesigned intellectual property (IP) the system is constructed by defining the its logical and physical structure first. Increased productivity is achieved through *(a) modularity*, *(b) abstraction*, and *(c) orthogonalisation of concerns*, such as the separation of communication and computation. Significantly, system-level timing is pre-determined avoiding iterative design and verification cycles normally necessary to achieve timing closure.

Since the methodology is based on architectural reuse, the choice of architecture is critical; the architecture must be suited to the applications which will be implemented on it as well as being well matched to the FPGA structure at which it is targeted. In this paper the methodology is exemplified with a single architectural instance described in [1] called *Sonic-on-a-Chip*.

2 Related Work

A popular area of research for improving designer productivity is the investigation of direct synthesis from existing software languages, e.g. [2], in an attempt to increase the level of abstraction. While this approach has had some measure of success, it is limited by the mismatch between logic and languages designed to abstract the execution of code for a sequential processor. An alternative is to use novel algorithm description formats, in which the algorithm designer explicitly identifies task-level parallelism [3,4]. Often the representation uses a dataflow graph format; invariably some form of modularity is used. These techniques suffer from a lack of a clear macro-architecture within the target FPGAs to which the modules may be mapped. This problem can be circumvented by using a custom modular or coarse-grained reconfigurable fabric instead of a traditional FPGA; the SCORE system [5] is one such example. The work in this paper focuses on an alternative, platform based approach, in which a hardware infrastructure is designed within an FPGA to be reused by many application instances. An FPGA based platform that employs an AMBA bus has been reported by Kalte *et al.* [6]. Rather than use a general purpose bus, our work assumes an infrastructure optimised for a specific domain. Moreover, the architecture is part of a larger methodology which includes precepts for hardware, software and operating system interactions.

3 Structured System Design

In a structured system design, a complex system is built by defining an architecture and then filling it with IP. The architecture can be envisaged as an interface between the form of the application algorithm and the unstructured resources of the FPGA. The architecture includes a modular logical layer which can be customised to the application, and a physical infrastructure layer, which describes how the logical modules are implemented and connected in the FPGA fabric.

The architecture design is exemplified by Sonic-on-a-Chip [1], an embedded video processing platform targeted at the Virtex II Pro family of FPGAs from Xilinx. The logical architecture is shown in Figure 1. In terms of the design methodology, the salient and necessary features of the logical model are that it is highly modular, extensible, and application implementation is achieved through customisation of modules. Moreover, the computation and communication are separated (via routers in this case) such that the topology of the application is independent of the intermodular communication infrastructure. This is important, since the infrastructure must remain fixed between applications.

The physical layer of the architecture must define how the logical system structure is implemented on the FPGA, including the location of inter-modular interconnect and the positioning of the modules. Existing design flows (including the Modular Design flow from Xilinx) integrate the modules and infrastructure together at design time, which we term *early integration*. In our design methodology the integration point is at run-time. This *late integration* affords the

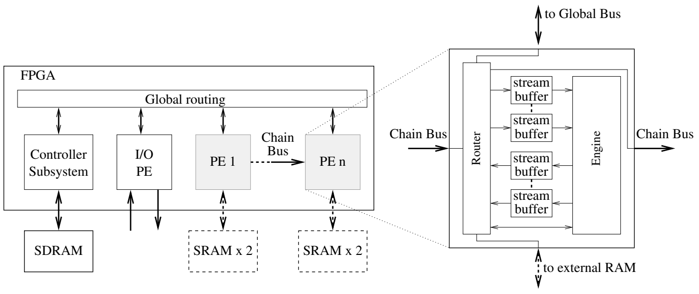

Fig. 1. General logical structure and processing element internals of Sonic-on-a-Chip.

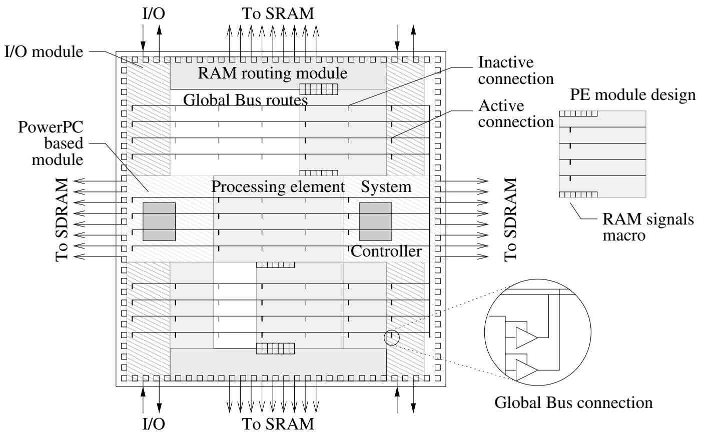

Fig. 2. Physical implementation of Sonic-on-a-Chip on a XC2VP100. Note that processing element designs must incorporate the Global Bus wiring.

maximum design productivity advantage from modularity by separating module development down to the bitstream level, since design, implementation and verification of different modules can be executed independently. In addition, different functionality can be obtained at run-time by combining fully developed modules in different permutations and combinations, allowing applications to be customised from instance to instance without re-synthesis, re-implementation and re-verification each time.

The example Sonic-on-a-Chip physical infrastructure for a Xilinx Virtex II Pro target is illustrated in Figure 2. Note that the modularity of the logical design is preserved at the physical level. In late integration the position of each module is not fixed until run-time, implying modules must be bitstream-relocatable and that both the FPGA and the infrastructure exhibit translational symmetry.

Importantly, the location of the Global Bus interconnect is fully predetermined. This is necessary for module relocatability; moreover, the overall bus timing is constant and can be characterised, enabling modules to be physically implemented independently. The interconnect and relocatability is inspired by the DISC system [7].

Application development for Sonic-on-a-Chip begins by specifying the processing algorithm to be implemented as a number of parallel tasks, connected in a dataflow graph format. Each task-node of the dataflow graph needs to be implemented in a processing element. Overall system control is handled in application software, which loads processing element module bitstreams into the configuration memory of the FPGA fabric and programs the routers within each processing element to direct dataflow appropriately.

In a stand-alone software environment, it is left to the application developer to manage the allocation of space and the positioning of modules within the fabric. In systems where two or more applications execute concurrently an operating system is usually employed to manage shared resources. In the case of Sonic-on-a-Chip, physically adjacent modules can communicate directly, so the relative positioning of modules must be considered by the operating system.

In order to represent the parallel processing and the topology of the hardware modules, in our methodology the application spawns a new software process for each hardware module. These processes are termed *ghost processes*, since they do not perform any processing but are a representation of the hardware. The communication between hardware modules is represented by redirecting the inputs and outputs of the ghost processes to named pipes (FIFOs). This provides a means to encode the logical topology of the algorithm dataflow as well as a mechanism for software to interact seamlessly with hardware entities.

4 Design Evaluation

A functional simulation model has been constructed and a prototype is under development based on the Xilinx ML300 evaluation board. The characteristics of two computational element designs is given in Table 1. The approximately 8x difference in resource usage is a simple but clear justification for the necessity of variable-sized PEs.

Table 1. Processing element characteristics (designs run at 50MHz).

Type	Slices	Block	MOPS	Throughput	Estimated no. per device	
		RAMs		Msamples/s	XC2VP30	XC2VP100
Image difference	345	3	50	50 peak	39	127
3x3 convolution	2450	32	528	59.3 peak	5	18

The physical implementation of processing element modules and RAM routing modules has been investigated using the partial reconfigurable design flow [8], which uses hard macros to ensure signals entering and exiting a reconfigurable module are always routed on the same wires. The standard bus macro in this flow is unsuitable for our implementation since it is only designed to pass signals across vertical module boundaries. New hard macro objects have been developed, including one to constrain Global Bus routing to specific tristate lines.

These investigations exposed routing issues with both the FPGA fabric and the routing tool (Xilinx PAR F.31). The FPGA interconnect is optimised for use as a global resource and therefore includes features such as global clock trees and longlines, which are not well suited to partial modular reconfiguration. In particular, horizontal longlines are not used when implementing a reconfigurable module. In addition, the place and route tool currently recognises vertical module boundaries only, and routing violations were observed for horizontal boundaries.

5 Conclusion

This paper introduced an architecture reuse based methodology for FPGA SoC design, which uses modularity and abstractions to enhance productivity. The architecture, which includes physical and logical constructs, is exemplified by the Sonic-on-a-Chip instance. Applications comprise hardware IP modules and control-level software; in an OS-based software environment the hardware processing is represented by spawned software processes connected via IPC FIFOs.

Investigations into the implementation of the methodology have exposed a dissonance between the globally optimised FPGA interconnect design and the requirements for modular reconfigurable routing; this is an area we plan to address in future work.

Acknowledgements. The authors thank the Commonwealth Scholarship Commission in the UK, the New Zealand Vice Chancellors' Committee and Xilinx Inc. for financial support.

References

1. Sedcole, N.P., Cheung, P.Y.K., Constantinides, G.A., Luk, W.: A reconfigurable platform for real-time embedded video image processing. In: Proc. Field–Programmable Logic and Applications. (2003)
2. Babb, J., Rinard, M., Moritz, C.A., Lee, W., Frank, M., Barua, R., Amarasinghe, S.: Parallelizing applications into silicon. In: Proc. IEEE Symposium on Field–Programmable Custom Computing Machines. (1999)
3. Diessel, O., Milne, G.: Hardware compiler realising concurrent processes in reconfigurable logic. IEE Proc. Computers and Digital Techniques **148** (2001) 152–162
4. Weinhardt, M., Luk, W.: Task parallel programming of reconfigurable systems. In: Proc. Field–Programmable Logic and Applications. (2001)
5. Caspi, E., Chu, M., Huang, R., Yeh, J., Wawrzynek, J., DeHon, A.: Stream computations organized for reconfigurable execution (SCORE). In: Proc. Field–Programmable Logic and Applications. (2000)
6. Kalte, H., Langen, D., Vonnahme, E., Brinkmann, A., Rückert, U.: Dynamically reconfigurable system-on-programmable-chip. In: Proc. Euromicro Workshop on Parallel, Distributed and Network-based Processing. (2002)
7. Wirthlin, M.J., Hutchings, B.L.: A dynamic instruction set computer. In: Proc. IEEE Symposium on FPGAs for Custom Computing Machines. (1995)
8. Lim, D., Peattie, M.: Two flows for partial reconfiguration: module based or small bit manipulation. Application Note 290, Xilinx (2002)