Design Flow on a Chip - An Evolvable HW/SW Platform

Stephan Kubisch, Ronald Hecht, Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering, University of Rostock, Germany
{stephan.kubisch, ronald.hecht, dtim}@uni-rostock.de

1 Introduction

Partial dynamic reconfigurability of modern FPGAs holds the potential of realizing autonomous and highly flexible Systems-on-Chip (SoC). Current devices can be configured with several partial bitfiles and replace particular ones on demand. But these precompiled bitfiles seriously lack flexibility: they are hardly relocatable and not adjustable. In other words, they are tied to their original functional scope and location. This paper proposes an integration of the design flow onto the reconfigurable device itself to be capable of runtime reconfiguration and adaptation through modifications on the source code level. Thus, a Network-on-Chip (NoC) and a Linux operating system (OS) have been combined. The design flow is integrated into a feedback control system, which runs as an application under Linux. As a consequence, an evolvable platform emerges, which is capable of adapting autonomously to its dynamic environment and changing parameters.

We briefly introduce the architecture of the evolvable platform here but will give a more detailed view on our framework and open issues in the poster.

2 Network-on-Chip and Linux

The current trend in hardware design leads to decentralization of communication and processing [3]. The increasing use of NoCs and the availability of highly functional and fine grained platform FPGAs support this evolvement. NoCs structure the reconfigurable area in homogenous distinct parts called tiles, which can be configured independently based on partial reconfigurability. A NoC represents the hardware base of our evolvable platform.

In Figure 1, a part of our NoC is shown as proposed in [4]. A tile is linked with one switch (S). Each switch is connected to 4 neighbouring switches establishing the communication grid. A network interface (NI) connects a tile and a switch. It is divided in a fixed (*R*esource *I*ndependent-NI) and a reconfigurable part (*R*esource *D*ependent-NI). Moreover, a NoC represents a high performing communication infrastructure. This is achieved by defining an appropriate

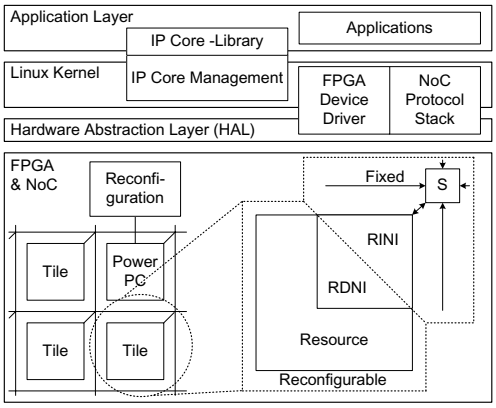

Figure 1. Fusion of NoC and Linux

protocol stack as introduced in [4]. The lower layers are implemented in hardware as part of the NoC. Arbitrary user protocols can be implemented in the reconfigurable layers or in software executed on the embedded PowerPC Core.

An OS capable of managing and abstracting the reconfigurable resources is the base of our software framework. The urgent need of such an OS has already been pointed out in [1, 2]. Thus, we refined a Linux OS. A general view of its layered architecture is given in Figure 1. The OS runs on one of the embedded PowerPC cores, which is connected to the NoC via the On-chip Peripheral Bus and a NI and to the reconfigurable hardware via the Internal Configuration Access Port. To integrate the NoC into Linux, a network device driver for the NoC and an FPGA device driver for the partial reconfiguration have already been implemented and added to the hardware abstraction layer (HAL). For the packet based communication in the NoC, the integrated TCP/IP socket API is used. The Linux kernel is extended with an IP core management extension to support instantiation, management and destruction of cores, dynamic relocation and swapping of cores, management and allocation of the FPGA area and inter-core and inter-process communication. An IP core can either be regarded as an accelerator running in hardware or as a decelerator executed in software

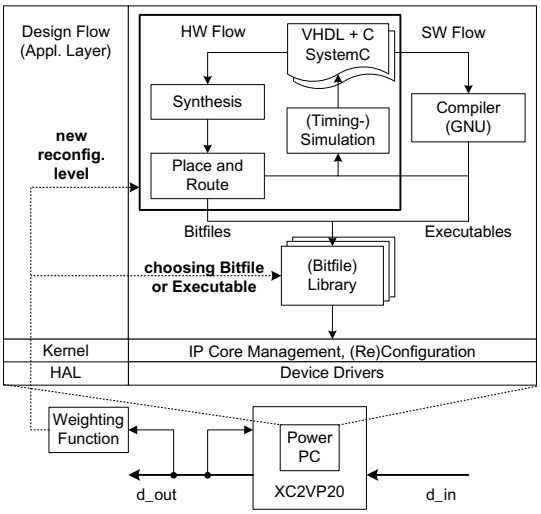

Figure 2. Design Flow Framework

on the embedded PowerPC. In the end, the main tasks comprise management of IP cores, reconfiguration and communication. But the manipulation of bitfiles is done by the design flow proposed in Section 3.

3 The Design Flow

The design flow can be understood as one possible application running under Linux. Its architecture is shown in Figure 2. At least one weighting function is used to compare observed system properties against specified ones and against minuted dynamics of the environment. When violations occur, appropriate changes on different system levels with different time horizons are stimulated within the system.

The new proposal within our framework is *reconfiguration on the source code level*. Modifying source code at runtime with subsequent online-synthesis on the target system itself offers the highest degree of abstraction. This approach is platform independent. The greater latency resulting from the high degree of abstraction can be compensated by using free processor resources on the FPGA. The free Alliance VLSI CAD System and additional Xilinx libraries can be used for prototypic implementation. For software reconfiguration, a free GNU compiler can be used.

In contrast, choosing between different precompiled bitfiles from a library is much faster than running the complete design flow. This level of adaptation relies on the systems ability to learn and improve over time. The library acts as a knowledge base offering precompiled bitfiles for various problems. Some of these bitfiles belong to the initial database and are inborn. The other group is compiled by the system itself and growing over time. Both bitfiles and software executables will be stored in the system library. The library is the basal database and represents the systems evolvement over time - its life cycle. It will be organized in a *Concurrent Version System* (CVS)-like manner. Changes are minuted over time resulting in a CVS tree containing the different system states in temporal order. In case of failure or environmental changes, the fall back capability ensures returning to stable and robust system states thus guaranteeing at least basic functionality. This ability is essential for evolvable systems and finally leads to Darwinism and evolution within the systems functional scope.

4 Results and current Work

Our NoC is implemented and tested on a Xilinx XC2VP20 development board. The NoC runs at speeds up to 90 MHz and achieves a maximum throughput of 170 MByte/s between adjacent switches. We are currently simulating the NoC together with the Linux OS. Sample AES crypto cores could already be started, stopped and relocated within the NoC. Next steps include the composition of the design flow and manually online synthesis runs.

5 Conclusion and Outlook

This paper proposes the integration of a hardware design flow onto a reconfigurable device which allows reconfiguration on the sourcecode level thus offering higher levels of in-field adaptation. An evolvable platform emerges as a new base in areas like communication and artificial intelligence where autonomy and flexibility are the most important demands. The system can evolve within its functional scope by its own stimulus and by the dynamics of the environment to equalize unbalanced or to fix faulty behaviour. But our plans are also tied to future technologies and a big leap in adaptability of hardware is still necessary. For example, the integration of a hardwired NoC in future FPGAs would be desirable. By optimally designing evolvable platforms as a conglomerate of software and hardware components, the sweet spot between performance and adaptation can be hit.

References

[1] G. Brebner. A virtual hardware operating system for the Xilinx XC6200. In *LNCS*, volume 1142, 1996.

[2] G. Brebner and O. Diessel. Chip-Based Reconfigurable Task Management. In *LNCS*, volume 2147, 2001.

[3] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. of the Design Automation Conference*, 2001.

[4] R. Hecht, D. Timmermann, S. Kubisch, and E. Zeeb. Network-on-Chip basierte Laufzeitsysteme fuer dynamisch konfigurierbare Hardware. In *ARCS 2004 - Organic and Pervasive Computing, Workshops Proceedings*, March 26 2004.