FlexDEF: Development Framework for Processor Architecture Implementation and Evaluation

Kasyab P. Subramaniyan, Erik Ryman, Magnus Själander,
Tung Thanh Hoang, Mafijul Md Islam, and Per Larsson-Edefors
VLSI Research Group, Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96 Gothenburg, Sweden
Email: {kasyab, erirym, hms, hoangt, mafijul.islam, perla}@chalmers.se

***Abstract*—Designing a processor is a complex task that uses multiple and varied tools. The complete development cycle spans software as well as hardware design and verification. More often than not, in spite of the close dependencies between hardware and software, there is no common platform for quick and accurate testing of these dependencies. Though such systems are often employed in industry, it is not common for end-to-end frameworks to be deployed in educational and research settings. We present the FlexCore Design Exploration Framework (FlexDEF), an end-to-end tool-chain used to develop the FlexCore processor and its accompanying cache system. The tool-chain is a hierarchically linked system that spans the various development phases involved in design and verification. The processor system is intended to be a model, for use in research-oriented projects where both the software and hardware are in a constant state of flux. We discuss the complete framework and the advantages in each context. Finally, we summarize the developments and discuss the future of the FlexDEF tool-chain.**

I. Introduction

It is a complex task to achieve a manufacturable processor; to do so in a time-constrained manner presents multiple challenges. In a research setting, the number of challenges is even greater due to the constant addition of ideas to the overall development process. It is therefore imperative that a framework is established where development becomes modular and simultaneous development is unimpeded. This work aims to describe a seamless development environment for such a scenario. The development environment is adopted in the fabrication of a FlexCore processor system. The FlexCore processor attempts to combine the efficiency of an ASIC and the flexibility of a reconfigurable platform [1], [2], [3].

There have been previous contributions to the area of processor design-space exploration. Azizi *et al.* use both architectural and circuit tradeoffs to enable joint design-space exploration [4]. In contrast to our contribution, however, Azizi *et al.* tradeoff the accuracy resulting from simulation for quick runtimes by employing a pre-characterized circuit library. La Rosa *et al.* explore the design space using conventional tool-chains, but employ an FPGA with custom instruction extensions in addition to a RISC processor, to achieve reconfigurability [5]. In the FlexCore architecture, reconfigurability is inherent. Karuri *et al.* present a design exploration environment based on an Architectural Description Language (ADL) to create reconfigurable application-specific processors [6]. This environment, based on the ADL LISA, generates both the software tool-chain and the hardware model of a conventional embedded processor and a coarse-grained reconfigurable architecture fabric. In this approach, while architectural features can be efficiently explored, the circuit tradeoffs cannot be efficiently explored due to the different levels of abstraction in modeling. Mehta *et al.* propose using super data flow graphs as an efficient method of processor architectural exploration [7]. The work by Mehta *et al.* is very similar in intent to the work presented here, in that reconfigurability is achieved as a by-product of the interconnectability of various datapath units.

The main contribution of this article is the FlexCore Design Exploration Framework (FlexDEF); a flexible tool-chain framework, that allows seamless exploration and development. On an architectural level the framework allows exploration for different application suites and datapath configurations. It also allows exploration for different configurations of the instruction decompressor. Flexible and numerous targets allow verification as well as performance and power evaluation to be carried out at different levels of abstraction: architectural (fast; low accuracy), RTL (speed-accuracy tradeoff) and physical models (slow; highly accurate). On an implementation level, the RTL representation is written in such a way as to allow both FPGA and ASIC implementations. When the end goal is to fabricate a processor, it is necessary to also consider the testability of the resulting silicon. This leads to system-level considerations like the cache hierarchy, a system bus, and the need for other peripherals, as shown in Fig. 1.

II. FlexDEF - The Development Methodology

FlexDEF is in the strictest sense a chain of tools; some developed within the research group, some commercially available and others publicly available. The tools are hierarchically linked and the framework utilizes scripts to complete the automation of the development process. Thus, the operational flow can be described by the headings in the remainder of this section.

A. Tool Hierarchy

All the tools used in FlexDEF are linked in a hierarchical manner. Through the judicious use of switches and well placed generics and pragmas, most configuration options can be handled with minimal intervention. It should however be noted here that when an ASIC implementation is desired, the

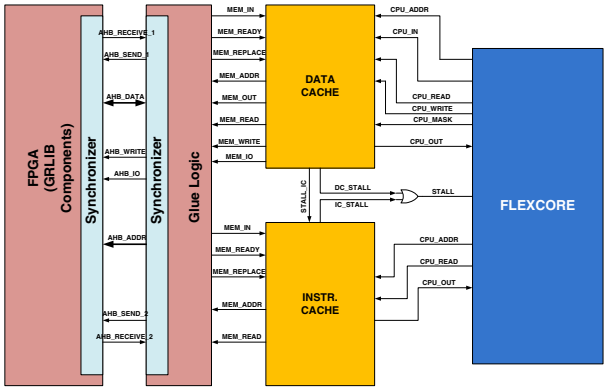


Fig. 1. Block diagram of the FlexCore system.

floorplanning portion of the physical implementation is going to be a one-time manual effort. Once a suitable floorplan has been validated, this too can be automated during the back-annotation phase of the verification.

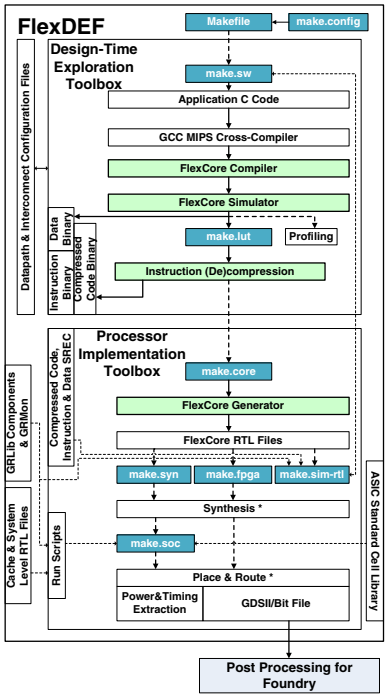
The hierarchy is driven from a *Makefile* located in the root directory of the FlexCore design and development environment. A *bin* directory contains various modules to handle different steps throughout the design, exploration and verification of the FlexCore. When targets in sequential steps are invoked, the dependency requirements and the manner in which the structure is architected ensure that targets required in previous steps are invoked automatically at the right design exploration stage. Thus, while Fig. 2 is operationally and logically representative of the FlexDEF framework, it does not reflect all the targets that are used. The dependencies in the tree have been deliberately introduced, in order to ease the burden on the end user, and consequently a number of targets are not directly visible. In Fig. 2, the blue (dark gray) boxes represent the targets used to chain the different tools together and the green (light gray) boxes represent the tools in the FlexTools suite [3]. Scripts, repository and generated files are represented by clear boxes, since they are generated and/or used by/for the application of this framework.

B. The Design-Time Exploration Toolbox

The design-time exploration toolbox consists of all the tools to handle the configuration and software verification of a complete FlexCore processor. The FlexCore processor datapath consists of the same datapath units as a MIPS R2000 processor [8] and, additionally, a multiplier. Further extensions to the datapath of the FlexCore are possible, due to the fine-grained architecture.

The tools comprising the design-time exploration toolbox have been discussed previously [3], while another publication [9] describes the instruction decompression scheme.

- **FlexComp - The Compiler**
  FlexComp, with its built-in scheduler, generates static schedules of FlexCore assembly instructions, called Register Transfer Notation (RTN) instructions, and writes these instructions to output files for use in the verification by the simulator. The scheduler expresses dependencies between instructions as *routing* constraints and employs a SAT-solver [10] to find a solution, if one exists.



* These steps are implementation dependent.

Fig. 2. Illustration of the FlexDEF framework.

- **FlexSim - Simulator and Assembler**
  The simulator adds the capability of verifying the RTN produced by FlexComp. In addition there is also a capability to simulate MIPS assembly code. However, recent work in the group has yielded a reference MIPS implementation, deprecating this capability. The FlexSim tool also contains an assembler that is responsible for linking and producing binary outputs of code and data. This binary code/data stream is also combined into a single EPROM load file in the srec format (see Sec. III-A) along with the compressed code produced by the instruction compression scheme.
- **FlexSize - Instruction Decompressor**
  One of the implications of the FlexCore architecture is that the instruction word could be wider than conventional VLIW processors. In order to keep instruction bandwidth low, there is a need to implement compression on the instruction words generated by FlexSim. An instruction compression technique based on partitioned Look-Up Tables (LUTs) was developed in earlier work [11]. This technique was enhanced and modified for the purpose of this work. This technique described in [9], essentially groups bits from the instruction word that display a high degree of correlation and generates all possible solutions as candidates for the LUTs. In order to determine the appropriate depth for the LUTs, the profiles from the suite of representative applications are used. Cost functions related to power, area and timing bound the solution to one that can be physically implemented.

The output from the design-time exploration toolbox serves a dual purpose: the binary outputs help with software verification for the chosen application suite, while the srec file is

transferred to the processor implementation toolbox to support verification of the hardware models at different levels of abstraction and accuracy.

C. The Processor Implementation Toolbox

The processor implementation toolbox (Fig. 2) has a front–end generator that can create behavioral (testbench) and structural (datapath) VHDL code. FlexGen generates a top-level VHDL module and an associated testbench for a particular FlexCore processor based on the specified datapath and interconnect configurations. All datapath units have been implemented and verified in advance, and their RTL descriptions reside in a code repository. The generator eliminates the hassle of error-prone VHDL coding to create FlexCore processors, and it can easily be extended to accommodate new units in a modular fashion. In order to integrate the units into the FlexCore processor in a rational way, the pre-optimized RTL code of the datapath units is defined with pipeline registers on the output. Based on the configuration files for datapath units and interconnect links, the RTL generator, FlexGen, instantiates datapath units together with interconnect definitions to generate RTL code for the top, architectural level of a FlexCore processor.

III. Ancillary Tools

A. SRecord

SRecord [12] is an open source collection of tools written for EPROM load file manipulation. The srec format was a convenient choice, because the simulation models for the memory can read these files during verification and more importantly, our chosen testbed—a combination of the fabricated prototype and an FPGA board—can be loaded with the same files. Although supporting scripts are used to efficiently manage the conversion from the binary format produced by the assembler, SRecord produces the final load file, with the correct memory addresses and checksums, that is loaded into the memory.

B. Run Scripts

The CAD tools used for the physical implementation, be it FPGA or ASIC, rely heavily on scripts (usually Tcl based and including custom command extensions) when running in batch mode. Using these tools in batch mode makes it efficient at runtime, but needs an experienced user in order to obtain correct results. These run scripts reside in the repository and are used by the tool hierarchy to implement the chosen configuration.

IV. Version Control

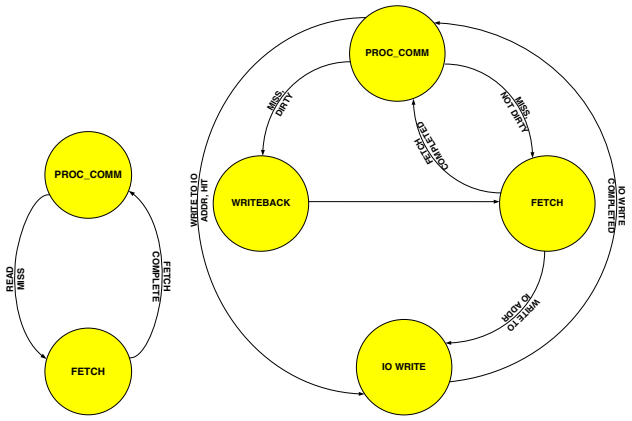
An important part of the system, yet placed outside the overall framework, is the Version-Control System (VCS) that is used during the development process. A VCS ensures that simultaneous development of tasks is possible and multiple users can contribute without being impeded by dependencies caused by simultaneous modifications to the source files.

The VCS that we chose to use is GIT [13]. It is an open source, distributed VCS.This means that anyone with a working copy of the repository also has a copy of the history. Changes to the local repository will not affect the central master until an explicit update is carried out. A powerful conflict resolution scheme ensures that collisions are minimal.

V. Implementation Considerations

Implementation demands certain design decisions that influence the overall architecture of the design. These decisions are usually made early on with flexibility being a key requirement.

During the assessment phase of the FlexCore fabrication effort it became clear early on that, while the datapath was at a mature stage of development, little effort had been invested on system architecture. The instruction decompressor was another component that was still early in the development phase. Thus, in order to architect a system at minimal effort, we decided to use Intellectual Property (IP) components as far as possible. A number of the system components, like the AHB bus components, were drawn from the open source IP library GRLib [14]. In addition to this, GRLib also comes with GRMon [15], a real-time design monitor/debugger for GRLib systems, that proves to be a huge aid in testing.



(a) Instruction cache. (b) Data cache.

Fig. 3. Finite State Machines for the caches.

The instruction cache, because of the wide nature of the FlexCore instructions, needed custom architecture. The data cache, while utilizing a standard 32-bit width, had dependencies on the instruction cache. Fig. 3 shows the Finite State Machines (FSMs) used in the instruction and data cache controllers, created for this purpose. Due to the differences between the two caches, and also the diversity in configuration options, the caches reside outside the FlexTools suite but still within the FlexDEF framework. The configuration generics include, among others, width (for the instruction cache), associativity and macro block size (for an ASIC configuration). For an FPGA implementation, the FlexDEF system is configured to use a basic memory block based on a generic HDL memory description. The FPGA synthesis tool then automatically infers LUTs, configured as memories, on the FPGA.

In addition, an ASIC implementation meant that the design was pin-limited. This led to the addition of extra control logic at the interface between the ASIC and the testbed. The glue logic, controlled by the FSM in Fig. 4, was so designed that it

could be used both with the FPGA as well as the ASIC version of the design. The glue logic also implements a handshaking protocol to ensure reliable transfer of data between the testbed and the ASIC. The data bus at the interface is also responsible for data directed to both the instruction and data cache, and also outbound data from the data cache. The glue logic on either side handles resolution when necessary. Due to the restrictions imposed by the interface and dependencies between the caches, the data cache gets priority in conflict resolution and can, in fact, stall the instruction cache, as is evident from Fig. 1. In view of the LUT-based instruction

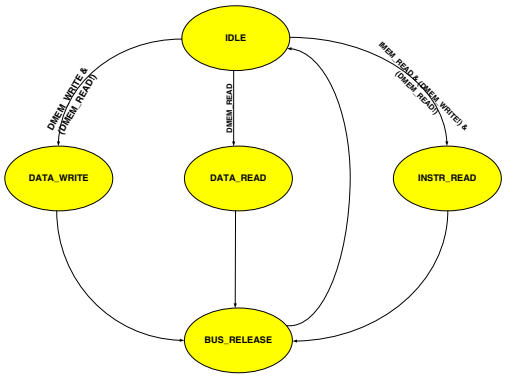


Fig. 4. FSM of glue logic.

decompressor, we initially chose memory macros that would suit both needs. However, initial estimates suggested that this would not be area efficient on the ASIC. Therefore, only the cache and tag blocks are implemented using memory macros while the instruction decompressor uses flip-flops.

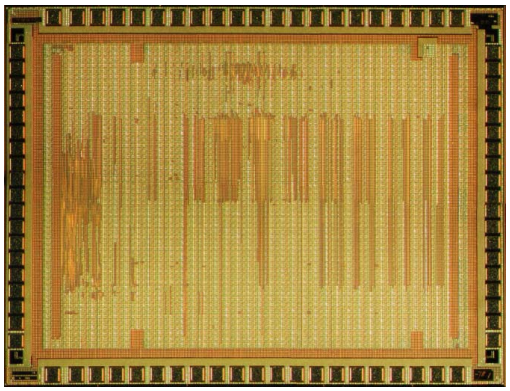


Fig. 5. Bare die of the FlexCore prototype.

VI. Summary

FlexDEF is a seamless design framework allowing a non-specialist user the option of realizing a specification into a processor implementation with minimal effort. A 130-nm FlexCore prototype (Fig. 5) implemented using FlexDEF is currently being measured. The ASIC testbed is an Xilinx ML-402 FPGA board that has 32 MB of onboard memory coupled to a Xilinx Virtex-4 SX-35 FPGA. This FPGA is large enough to fit the complete design and is well suited for the FPGA implementation too. Both the instruction and data caches are sized to 2 KB and generated using memory macros of dimension 64 X 8.

The future of FlexDEF includes development of light-weight units that can extend the datapath of the FlexCore and accelerate application execution. Designing a reconfigurable datapath that can contain a variable number of units presents challenges at the compiler level. Thus, research on new compiler technology to enable simple "plug-n-play" datapath unit integration and design space exploration is also underway. Finally, an alternate strategy to use tools in the public domain for the purpose of compiling FlexCore code is also being looked into. At present, the tools comprising FlexTools are written in Haskell, presenting some challenges and shortcomings. The idea behind the effort to use tools that are more conventional is to enable the FlexCore architecture to be available to a wider audience. Progress on this front will be reported on our site www.flexsoc.org.

References

[1] M. Själander, P. Larsson-Edefors, and M. Björk, "A Flexible Datapath Interconnect for Embedded Applications," in *IEEE Computer Society Annual Symp. on VLSI*, May 2007.

[2] M. Thuresson, M. Själander, M. Björk, L. Svensson, P. Larsson-Edefors, and P. Stenstrom, "FlexCore: Utilizing Exposed Datapath Control for Efficient Computing," *J. of Signal Processing Systems*, vol. 57, no. 1, pp. 5–19, 2009.

[3] T. T. Hoang, U. Jälmbrant, E. der Hagopian, K. P. Subramaniyan, M. Själander, and P. Larsson-Edefors, "Design Space Exploration for an Embedded Processor with Flexible Datapath Interconnect," in *IEEE Int. Conf. Application-Specific Systems Architectures and Processors*, 2010, pp. 55–62.

[4] O. Azizi, A. Mahesri, J. Stevenson, S. Patel, and M. Horowitz, "An Integrated Framework for Joint Design Space Exploration of Microarchitecture and Circuits," in *Design, Automation Test in Europe Conference Exhibition*, 2010, pp. 250–255.

[5] A. La Rosa, L. Lavagno, and C. Passerone, "Hardware/Software Design Space Exploration for a Reconfigurable Processor," in *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 570–575.

[6] K. Karuri, A. Chattopadhyay, X. Chen, D. Kammler, L. Hao, R. Leupers, H. Meyr, and G. Ascheid, "A Design Flow for Architecture Exploration and Implementation of Partially Reconfigurable Processors," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1281–1294, 2008.

[7] G. Mehta and A. Jones, "An Architectural Space Exploration Tool for Domain Specific Reconfigurable Computing," in *IEEE Int. Symp. Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design, The Hardware/Software Interface*, 2nd ed. Morgan Kaufman Publishers Inc., 1998.

[9] A. Bardizbanyan, M. Själander, and P. Larsson-Edefors, "Reconfigurable Instruction Decoding for a Wide-Control-Word Processor," in *IEEE Int. Symp. Parallel and Distributed Processing, RAW Workshop*, 2011, pp.317-320.

[10] N. Een and N. Sörensson, *Mini SATisfiability (MiniSAT)*. [Online]. Available: http://minisat.se

[11] M. Thuresson, M. Själander, and P. Stenstrom, "A Flexible Code Compression Scheme using Partitioned Look-Up Tables," in *Int. Conf. High Performance Embedded Architectures and Compilers*, Jan. 2009, pp. 95–109.

[12] P. Miller, *SRecord 1.56*. [Online]. Available: http://srecord.sourceforge.net/

[13] P. Baudis, *Git Fast Version Control System*. [Online]. Available: http://git.or.cz/index.html

[14] Aeroflex Gaisler, *GRLIB IP Core User's Manual, Version 1.1.0 B4104*. [Online]. Available: http://gaisler.com/products/grlib/grip.pdf

[15] Aeroflex Gaisler, *GRMON User's Manual, Version 1.1.49*. [Online]. Available: http://www.gaisler.com/doc/grmon.pdf