RICH-IP: An Interactive System for Configurable High-Level IP Synthesis

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis, Hardware description languages*

General Terms

Design, Languages

Keywords

IP design, Hardware synthesis

1. INTRODUCTION

Hardware description language (HDL) based IP synthesis has been the industry standard in recent years. However, it has a number of problems. First, Verilog and VHDL are so low level that they are often compared to assembly languages in terms of programability. The absense of advanced data types and control mechanisms makes programming in these languages tedious and error-prone. For example, to program complex circuits, one often has to write massive amount of boilerplate code. Some designers go as far as using Perl to generate repeated Verilog code fragments. Second, the simulation and verification is the most time-consuming step in IP design cycle. For industry-strength design, this loop can go on for weeks and even months. This lengthy process is in part due to the lack of static checks at compile time in most HDLs and the difficulty in formal verification. Third, while most HDLs offer design libraries of basic building blocks, these are largely limited to the harware circuit level, and are often inadequate for programming large, complex but common algorithms such as those used in cryptography and image processing. Today, designs for these algorithms require thousands of lines of Verilog code which is extremely expensive to produce, debug and maintain. Last, because HDLs do not offer the capability of high level abstraction, it is not easy to reconfigure the functionality of an existing design. For example, for a given design of an AES algorithm, if the user prefers to trade die space for speed, a common approach is to unroll a loop a number of times and execute it in parallel within a clock cycle. Such unrolling cannot be achieved in Verilog without substantial code change.

To address some of these problems, a range of new solutions have been proposed. SystemC gives software engineers access to hardware design by introducing an event-driven simulation kernel and some ability in describing hardware in C. SystemVerilog, on the other hand, aids hardware designers by raising the abstract level of Verilog with convenient programming constructs and some object-oriented concepts. None of these languages allows the specification of a design at the algorithmic level. Property Specification Language (PSL) let the designers add assertions in their HDL code about certain properties. It is nonetheless the user's responsiblity to add correct assertions at the appropriate places. Another approach to the verification problem is equipping the HDL with a type system and providing static checks. A number of HDLs, most notably Bluespec [1], are designed as sublanguages or libraries of strongly typed functional programming languages like Haskell and ML. While the type system in these languages eliminates some design errors at early stage, it does not make the coding of a design any easier. To the contrary, hardware engineers have to learn another potentially unfamilar language such as Haskell. This is perhaps why Bluespec now uses a SystemVerilog syntax. Probably the most relevant work to this paper is the Spiral project [2] which developed systems that translate high level mathematical representations (in the SPL language) of certain DSP transforms into hardware designs. The work identifies a number of one-to-one correspondence between DFT formulas to combinatorial data paths, and hence makes the abstraction of functional components possible.

In this paper, we propose an architecture for IP design which leverages the programability of SystemVerilog but extends it with a high level of mathematical abstraction. It allows the hardware designer to specify algorithms and constraints in their designs directly, without concerns about the unnecessary details of hardware connections when developing larger IPs. Our two-level compiler automatically translates a high level abstraction into a paramterized template, which, together with user input on the choices of the parameters, synthesizes to a custom RTL. This architecture generalizes the Spiral system to handle more diverse design problems than DSP and uses a frontend familiar to most hardware engineers. More importantly, the interactive parameter instantiation approach conveniently exposes areas of algorithmic optimization to the designer and automatically generates IPs that best cater to individual's requirements.

DAC '10 Anaheim, CA, USA

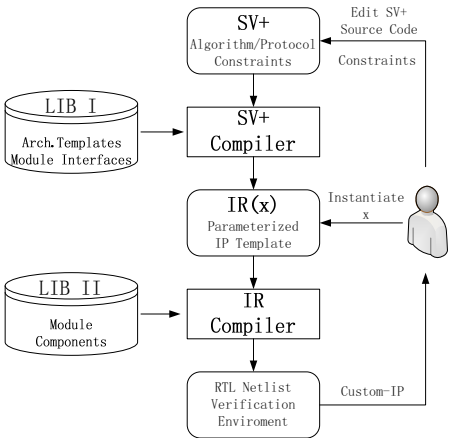

Figure 1: The architecture

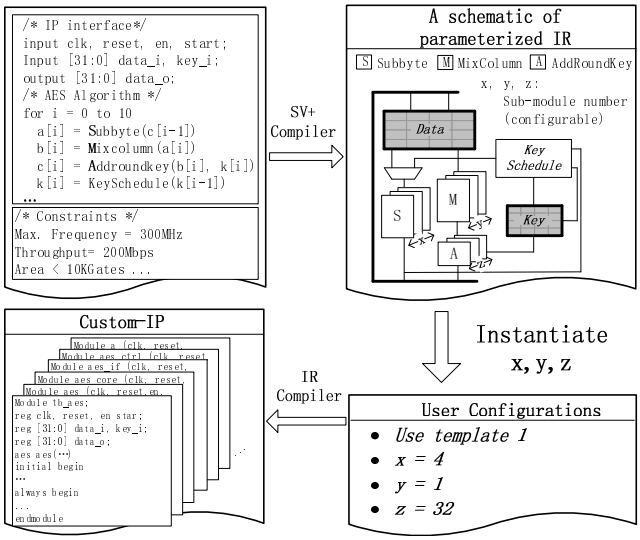

Figure 2: The synthesis of AES IP core

2. THE MAIN IDEA

Figure 1 depicts the architecture of the proposed the RICH system. The IP designer describes the algorithm, IP interfaces as well as constraints in SV+, a language extension to SystemVerilog. The constrains define the physical properties of IP, such as maximum frequency, area cost, or minimum throughput. The SV+ code then gets preprocessed by an SV+ preprocessor. This is a coarse-grained IP generation step in which the preprocessor searches through the *template library* to locate most suitable hardware templates that satisfies the IP constraints and module interfaces. The result of the preprocessing is a parameterized intermediate representation of the algorithm, IR(x), where x is a set of variable parameters. At this point, the designer instantiate these variables using a configuration file. Finally, in a fine-grained IP generation step, the fully instantiated IR is compiled into the RTL and verification environment with the help of the module library. The end result of the entire flow is a custom IP core, and it is passed to the designer for verification. In case the generated custom IP core does not match the specification, the designer can modify his or her design by either fine tuning the IR with a different set of variable instantiations, or by coarse adjustment in the SV+ code. The template library is designed for mapping algorithms to specific hardware architectures. It is co-designed by hardware engineers and algorithm designers, and it is highly optimized for hardware implementation. The module library contains many frequently used module components. The circuit-level optimizations are accomplished both in this library and during RTL code generation. Both libraries are open and extendable, which makes the system highly flexible.

3. AN EXAMPLE

As an example, Figure 2 illustrates of the synthesis of an AES IP core. AES [3] is a new encryption standard which is widely used in both software and hardware designs. Different applications call for AES IP cores with different speed, area, and frequency. There are four main functions in the AES algorithm: `Subbyte(S)`, `Mixcolumn(M)`, `Addroundkey(A)`, and `KeySchedule`. All of these functions can be represented by arithmetic transformation modules.

To automatically generate a specific type of AES core, the designer first describes the AES algorithm, the IP interface and constraints in SV+. The SV+ code is then preprocessed into a parameterized IR with five components: *Register*, S, M, A, and *KeySchedule*. Each component corresponds to a function in the AES algorithm, and all of them are pre-designed in libraries. The template library contains the abstract view of these components, such as the interface, timing and area information. The preprocessor generates a number of possible templates that implements this algorithm, and the schematic of one such template (template 1) is shown in the figure. The number of copies of S, M and A are configurable by variable x, y, and z. The module library contains the detailed hardware implementation of these modules. Once instantiated with the user configuration file, a custom AES IP core is automatically generated by the IR compiler.

4. CONCLUSION

A rapid, interactive, configurable and high-level IP synthesis system (RICH-IP) was proposed. It offers a friendly front-end (based on SystemVerilog) and high levels of abstraction when designing complex IP cores. Its two-level compiler allows users to interactively configure important design parameters in order to generate optimized cores that best suit the user's requirements. We believe the system significantly simplifies IP design and performance tuning, and dramatically decreases time-to-market.

5. REFERENCES

[1] Arvind. Bluespec: A language for hardware design, simulation, synthesis and verification. In *MEMOCODE*, pages 249–, 2003.

[2] G. Nordin, P. A. Milder, J. C. Hoe, and M. Püschel. Automatic generation of customized discrete fourier transform IPs. In *DAC*, pages 471–474, 2005.

[3] N. I. of Standards and Technology(U.S.). *Advanced Encryption Standards (AES)*. FIPS Publication.