Hardware-Software Interface Specification for Verification in Accelerator-Rich Platforms

Hongce Zhang, Bo-Yuan Huang, Yue Xing, Aarti Gupta, Sharad Malik
Princeton University
USA

ABSTRACT

This paper presents the Instruction-Level Abstraction (ILA) as a hardware-software interface specification for accelerator-rich architectures. The ILA provides a common framework for formal functional specification of processors and accelerator behavior, verifying their implementations, and reasoning about software-hardware interactions of programs with accelerators. The ILA-MCM specification extends the ILA to enable reasoning about interactions of accelerators with other compute engines (processors and accelerators) through shared memory.

1 INTRODUCTION

The current trend of accelerator-rich platforms poses two distinct challenges. The first challenge is how to construct meaningful specifications for accelerators that describe the behavior exposed at the hardware-software *interface*. Such specifications are important both in design and verification, for example, driving software and firmware development even before the hardware is "taped-out," and ensuring software portability between different generations of an accelerator architecture. The second challenge is how to separate the hardware and software verification concerns. For software that runs exclusively on a general-purpose processor, its execution semantics are defined by the processor's instruction set architecture (ISA) specification. Thus, the ISA serves as a suitable abstraction of the underlying processor hardware for software verification. However, similar abstractions for reasoning about software interacting with accelerators are lacking.

To address these challenges, we have proposed a uniform and formal abstraction/specification for processors and accelerators that captures their software-visible functionality [4]. This abstraction is called an Instruction-Level Abstraction (ILA) and is based on the familiar notion of computation triggered by "instructions." For accelerators, the instructions are the commands at the interface that update software-visible (*viz.* architectural) state variables.

2 ILA FOR SPECIFICATION

Top-down an ILA provides a specification for functional verification of hardware, and bottom-up it provides an abstraction for software-hardware co-verification.

The ILA, like an ISA, provides: (a) a modular functional specification as a set of instructions; (b) a meaningful state abstraction in terms of architectural state; and (c) a specification for each instruction in the form of state update functions for architectural state variables (and potentially with child-instructions for hierarchical specifications). Figure 1 shows an example of the ILA model for an AES block encryption accelerator with its architectural state variables and the specification of instructions that describe how these architectural state variables are updated. Note that each instruction here corresponds to a memory-mapped input/output (MMIO) access from a program running on a host processor.

MMIO Access	Instruction
WR base+0xff00	START_ENCRYPT
RD base+0xff00	GET_STATUS
WR/RD base+0xff02	WR/RD_ADDR
WR/RD base+0xff10	WR/RD_KEY
WR/RD base+0xff20	WR/RD_CTR

// Architecture state variables:
{AesState, Addr, Length, Ctr, Mem, ... }

// Instruction WR/RD_ADDR:
$\delta_1 \triangleq$ (InAddr = 0xFF02)
N_1[Addr] $\triangleq$ ite(Cmd = 1, InData, Addr)
N_1[OutData] $\triangleq$ ite(Cmd = 0, Addr, 0)

Figure 1: The ILA model sketch for an AES block encryption accelerator

3 APPLICATIONS OF ILA-BASED VERIFICATION

With the ILA, one can separately verify that an ILA model is a correct abstraction (at the instruction-level) of a given hardware implementation by verifying it against the Register-Transfer Level (RTL) design and then use this sound abstraction for system-level verification (e.g., hardware/software co-verification).

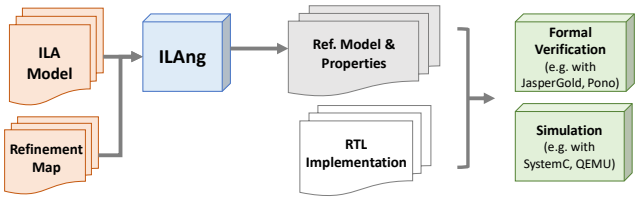

Figure 2: ILA-based implementation verification flow

3.1 ILA for Accelerator Implementation Verification

The flow of ILA vs. RTL implementation verification (both formal and simulation-based) is shown in Figure 2, where the ILAng platform supports modeling and verification using ILAs [6]. As the ILA and RTL are models at two different levels of abstraction, a *refinement map* [9, 10] is needed to connect the two levels during verification. This refinement map specifies: (1) which variables to check, i.e., a variable mapping between architectural (software-visible)

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
LATTE '21, April 15, 2021, Virtual, Earth
© 2021 Copyright held by the owner/author(s).

state variables in the ILA and the corresponding RTL variables, and (2) when to check, i.e., identifying when the corresponding variables should be checked in the two models. For (2), we only require users to provide the instruction *completion* condition. We believe such a condition is usually not too difficult to specify, because it is a common design practice to have a commit/finish signal for a command/operation in RTL.

```
"state_mapping": {
  "aes_address" : "RTL.aes_reg_opaddr_i.reg_out",
  "aes_length"  : "RTL.aes_reg_oplen_i.reg_out",
  … },
"instructions": [{
    "instruction"  : "WR_ADDR",
    "ready_signal" : "RTL.xram_ack_delay_1",
    "max_bound"    : 20
  }, …]
```

Figure 3: An example of refinement map

The refinement map in ILAng uses the JSON format. An example of the refinement map is shown in Figure 3, with variable mapping on the left and the instruction completion condition on the right. Variable mapping describes relations between architectural state variables in the ILA and the RTL signals. It supports complex mapping relations using conditions and auxiliary variables. The instruction completion condition accepts a commit signal or a cycle-bound. For many common design styles (e.g., pipelining, resource time-multiplexing etc.), we add constructs to make it convenient to write the refinement map. For more information about the refinement map, readers can refer to our online document [15].

3.1.1 Formal Verification. As shown in Figure 2, the ILAng framework can automatically generate formal properties in SystemVerilog Assertions (SVAs) that can be used with off-the-shelf model checkers, either commercially available (Cadence JasperGold [2]) or open-source (Pono [8]). This formal verification flow avoids the need for manually generated ad-hoc properties and ensures full coverage of the functional specification.

3.1.2 Simulation-based Validation. As an alternative to formal verification, the ILAng framework also supports simulation-based validation by automatically synthesizing an executable model from the ILA specification of an accelerator. We also support multi-level simulation, where the ILA and RTL models run side-by-side with simulation states compared at each checkpoint (e.g., after each instruction, or at certain conditions), and switching between the high-level ILA and low-level RTL simulation.

The above verification approaches have already been used in prior processor verification practices [1, 11–13]. The ILA model allows us to successfully leverage them in an automated way to support accelerator verification.

3.2 ILA for Hardware-Software Co-Verification

As the ILA is an abstraction of accelerators at their interface, it can be used to replace a low-level model in the hardware-software co-verification setting. Our previous works covered the detailed techniques and application case studies for both simulation-based validation [14] and formal hardware-software co-verification [5].

For the verification at the system-level, one also needs to take memory consistency issues into consideration. The ILA-MCM specification extends the ILA to enable reasoning about interactions of accelerators with other compute engines (processors and accelerators) through shared memory [16].

4 CASE STUDY: AN AES ENCRYPTION ACCELERATOR

This accelerator uses the encryption engine from OpenCores [3] and is available in the form of a Verilog RTL model (LoC: 1217). The ILA and RTL models and the verification setup for this case study are available in our ILA Modeling DataBase (IMDB) on Github [7]. More detailed case studies can be found in our previous work [4].

At the high-level, the accelerator receives configurations like the encryption key via MMIO commands and uses DMA to exchange data with the shared memory. These MMIO commands (and therefore, instructions) have been listed in Figure 1. Among them, the command `START_ENCRYPT` is used to trigger the accelerator to perform a sequence of steps for encryption of a block.

In this case study, the AES ILA model is manually written (C++, LoC: 433). The ILA model is significantly smaller than the final RTL implementation, making ILA an attractive entry point for verification and validation. The ILA model captures the MMIO interface of the commands together with specification of the computation performed by the accelerator. The instruction for `START_ENCRYPT` has the most complex specification and uses a hierarchical description to efficiently represent the 10-round AES-128 encryption function as well as the block-level looping through the plaintext.

With the ILA model as a specification, we can formally verify the RTL implementation of the accelerator. Here we make use of the model checker Pono and the total verification time is 17min on a machine with Intel Core i5-8300H CPU and 32GB RAM.

After passing the ILA vs. RTL verification, the ILA model becomes a reliable abstraction of RTL and therefore, can be used in system-level verification. Our experiment showed that the auto-generated simulation model from AES ILA runs at a speed of 7.3μs/instruction, which provides about 53x speed-up compared to simulating the RTL model (387μs/instruction).

5 CONCLUSION

The instruction-level abstraction (ILA) captures the behavior of accelerators at the hardware-software interface, much as the ISA has done for processors. The ILA-MCM extension enables reasoning about interactions of accelerators and processors through shared memory. This paper provides an overview of the ILA for specification and verification of accelerators. This uniform hardware-software interface also serves as a compilation target for application development that largely uses existing compilation flows.

ACKNOWLEDGMENTS

This work was supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA; by the DARPA POSH and DARPA SSITH programs; and by NSF XPS Grant No. 1628926.

REFERENCES

[1] Jerry R Burch and David L Dill. 1994. Automated Verification of Pipelined Microprocessor Control. In *Proceedings of the International Conference on Computer Aided Verification*. London, 68–84. https://dl.acm.org/citation.cfm?id=735662

[2] Cadence Design Systems, Inc. 2018. JasperGold: Formal Property Verification App. http://www.jasper-da.com/products/jaspergold-apps/ Accessed: 2018-01-02.

[3] Homer Hsing. 2014. OpenCores.org: Tiny AES. https://opencores.org/project,tiny_aes. Accessed: 2017-11-17.

[4] Bo-Yuan Huang, Hongce Zhang, Pramod Subramanyan, Yakir Vizel, Aarti Gupta, and Sharad Malik. 2019. Instruction-Level Abstraction (ILA): A Uniform Specification for System-on-Chip (SoC) Verification. *ACM Trans. Design Autom. Electr. Syst.* 24, 1 (2019), 10:1–10:24. https://doi.org/10.1145/3282444

[5] Bo-Yuan Huang, Sayak Ray, Aarti Gupta, Jason M. Fung, and Sharad Malik. 2018. Formal Security Verification of Concurrent Firmware in SoCs using Instruction-Level Abstraction for Hardware. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465794

[6] Bo-Yuan Huang, Hongce Zhang, Aarti Gupta, and Sharad Malik. 2019. ILAng: a modeling and verification platform for SoCs using instruction-level abstractions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 351–357. https://doi.org/10.1007/978-3-030-17462-0_21

[7] Bo-Yuan Huang, Hongce Zhang, Yue Xing, Huaixi Lu, Yi Li, and Yu Zeng. 2021. ILA Model Database. https://github.com/PrincetonUniversity/IMDb/tree/master/accls/AES/AES-ILA-RTL Accessed: 2021-03-31.

[8] Makai Mann, Ahmed Irfan, Florian Lonsing, Hongce Zhang, Leonard Truong, Yahan Yang, Keyi Zhang, and Yoni Steckel. 2021. Pono: an SMT-based model checker built on SMT-switch. https://github.com/upscale-project/pono Accessed: 2021-02-04.

[9] Panagiotis Manolios and Sudarshan K Srinivasan. 2005. A complete compositional reasoning framework for the efficient verification of pipelined machines. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.* IEEE, 863–870.

[10] Panagiotis Manolios and Sudarshan K Srinivasan. 2008. A Refinement-based Compositional Reasoning Framework for Pipelined Machine Verification. *IEEE Trans. VLSI Syst.* 16, 4 (2008), 353–364.

[11] Daniel Petrisko, Farzam Gilani, Mark Wyse, Tommy Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, and Bandhav Veluri. 2020. BlackParrot: An Agile Open Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* (2020).

[12] Alastair Reid. 2017. Trustworthy Specifications of ARM® v8-A and v8-M System Level Architecture. In *Proceedings of the Conference on Formal Methods in Computer-Aided Design*. 161–168. https://doi.org/10.1109/FMCAD.2016.7886675

[13] Alastair Reid, Rick Chen, Anastasios Deligiannis, David Gilday, David Hoyes, Will Keen, Ashan Pathirane, Owen Shepherd, Peter Vrabel, and Ali Zaidi. 2016. End-to-End Verification of ARM® Processors with ISA-Formal. In *Proceedings of the International Conference on Computer Aided Verification*, Vol. 9780. 42–58. https://doi.org/10.1007/978-3-319-41540-6_3

[14] Yue Xing. 2021. Demo: Co-Simulation Model from AES ILA. http://upscale.stanford.edu/materials/ila.m4v Accessed: 2021-02-21.

[15] Hongce Zhang. 2021. Refinement Relation Specification in ILAng. https://bo-yuan-huang.gitbook.io/ilang/verification/refinement Accessed: 2021-02-21.

[16] Hongce Zhang, Caroline Trippel, Yatin A. Manerkar, Aarti Gupta, Margaret Martonosi, and Sharad Malik. 2018. ILA-MCM: Integrating Memory Consistency Models with Instruction-Level Abstractions for Heterogeneous System-on-Chip Verification. In *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 1–10. https://doi.org/10.23919/FMCAD.2018.8603015