Hector: Multi-level Paradigm in Hardware Synthesis

Ruifan Xu, Youwei Xiao, Jin Luo, Yun Liang
Peking University
Beijing, China
{xuruifan,shallwe,luo-jin,ericlyun}@pku.edu.cn

ABSTRACT

In recent years, hardware intermediate representation (IR) has become a popular topic. An appropriate IR can give enough expressivity and avoid redundant engineering efforts. Hector IR fits well in the hardware synthesis flow, which brings the opportunity of building new synthesis tools with minimal effort. Besides, the multi-level paradigm also enables automatic verification in the synthesis flow.

1 INTRODUCTION

Hardware description languages like Verilog are frequently used in the industry for hardware design and verification, which adopt a low level of abstraction known as register transfer level (RTL). To improve the productivity of hardware design, hardware synthesis approaches convert a higher level abstraction into the RTL implementation through a series of compilation techniques and optimization passes. High-level synthesis (HLS) automatically generates hardware from a behavioral description, while hardware generators often make use of domain knowledge to optimize hardware.

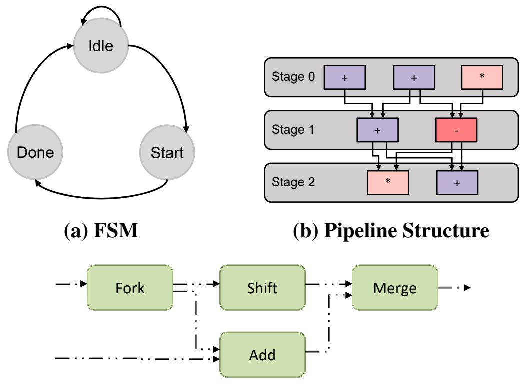
These synthesis methods adopt different synthesis flows, which increase the difficulty of building a new synthesis tool. However, these flows share some similarities like the control logic of hardware and necessary optimizations in the lowering procedure. An appropriate intermediate representation (IR) is suitable to solve this problem, which provides enough reusability and extensibility. The critical question is *what is the suitable IR for varieties of hardware synthesis methodologies*?

Hector [14][1] is a hardware synthesis framework supporting various synthesis methodologies built on MLIR infrastructure [6]. The framework proposes a unified IR with the multi-level paradigm, which can flexibly describe hardware design and build a new hardware synthesis approach with minimal effort. These benefits are mainly brought from the appropriate IR that is sufficiently expressive and provides enough optimizations.

2 BACKGROUND

Control logic of hardware. As shown in Figure 1, there are three common patterns in hardware design. Besides normal FSM approach, the pipeline structure uses a set of data processing elements connected in series, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel to improve performance. Latency-insensitive design is a dataflow circuit where data transfer only happens when predecessors and successors are all prepared. All data signals are accompanied by handshake signals, indicating the availability of the next data from the source unit and the readiness of the target unit to accept it, respectively.

[1]Hector is open source at (https://github.com/pku-liang/Hector)

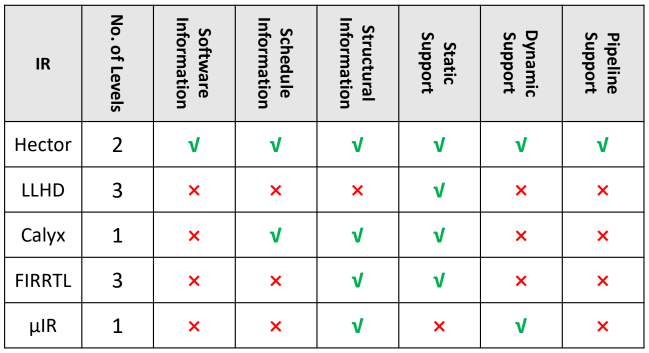


(a) FSM **(b) Pipeline Structure**

(c) Latency-insensitive Design

Figure 1: Three patterns of hardware control logic.

3 COMPARISON



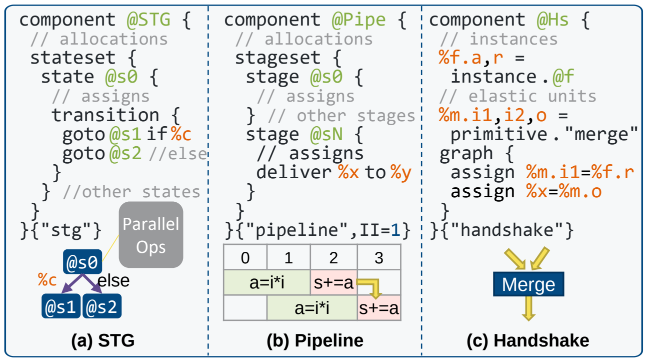
IR	No. of Levels	Software Information	Schedule Information	Structural Information	Static Support	Dynamic Support	Pipeline Support
Hector	2	√	√	√	√	√	√
LLHD	3	×	×	×	√	×	×
Calyx	1	×	√	√	√	×	×
FIRRTL	3	×	×	√	√	×	×
μIR	1	×	×	√	×	√	×

Figure 2: The comparison against other hardware IRs.

There have been some hardware IRs like FIRRTL [5], LLHD [9], μIR [10] and Calyx[7]. However, most IRs focus on a lower abstraction like structural and netlist. Only a few IRs support software or schedule information, and none of them can describe various hardware behaviors. Hardware synthesis involves multiple abstractions between behavioral description and RTL implementation, for which a single abstraction cannot handle the entire procedure. Describing hardware behaviors like sequential, pipeline and dynamic at the software level provides enough productivity and simplifies the hardware design as well. Hector is the only IR for different synthesis methods and provides a unified representation for different hardware behaviors. Besides that, CIRCT [3] project aims at constructing a reusable and modular infrastructure for the entire hardware generation including high-level synthesis and logic synthesis. This project is still in progress and absorbs existing IR designs like Calyx and LLHD. Hector can be regarded as the front segment of CIRCT infrastructure.

4 OVERVIEW

Hector contains a two-level IR system, which is built on MLIR infrastructure for reusability and extensibility. In Hector, ToR is the high-level IR and HEC is the low-level IR. Both IRs provide a uniform representation of the control logic with various scheduling manners including static, pipeline, and dynamic. The main difference between the two IRs is that ToR describes when the operation begins, while HEC describes where it takes place.



```
component @STG {
 // allocations
 stateset {
  state @s0 {
   // assigns
   transition {
    goto @s1 if %c
    goto @s2 //else
   }
  } //other states
 }
}{"stg"}
```

```
component @Pipe {
 // allocations
 stageset {
  stage @s0 {
   // assigns
  } // other stages
  stage @sN {
   // assigns
   deliver %x to %y
  }
 }
}{"pipeline",II=1}
```

```
component @Hs {
 // instances
 %f.a,r =
  instance.@f
 // elastic units
 %m.i1,i2,o =
  primitive."merge"
 graph {
  assign %m.i1=%f.r
  assign %x=%m.o
 }
}{"handshake"}
```

Figure 3: Three styles of components described in HEC, which has the same functionality with three manners of ToR.

4.1 Hector IR

The idea of ToR is to make it closer to hardware by providing a time graph and binding software operations to elements of the graph. All the operations are bind on the time graph, which is a directed graph that carries control flow and timing information. Therefore, ToR is capable of providing a high-level abstraction of the scheduling information. Three scheduling manners: **static**, **pipeline**, and **dynamic**, are all supported in ToR as edge attributes. This unified representation makes it easier to transform among different behaviors.

HEC describes hardware with different manners in a unified allocate-assign mechanism depicted in Figure 3. Compared with ToR, HEC works at a level much closer to hardware. It explicitly describes the resource usage (including registers, memory, and compute units). Corresponding to the different behaviors in ToR, a HEC design is composed of three types of components matching their manners: **STG** (state transition graph), **Pipeline** and **Handshake**.

4.2 Synthesis flow

The two-level representation makes it easy to implement different synthesis methods. ToR IR provides a high-level abstraction of schedule information, and different scheduling approaches can be easily implemented by transformation on the time graph. The explicit representation of allocation in HEC IR brings the opportunity for resource sharing, which significantly reduces resource consumption. The allocation of sub-modules forms a hierarchical representation of hardware, which is capable of describing the architectural design. The allocate-assign mechanism also simplifies the definition of interconnection between different modules. Therefore, both HLS and hardware generators can be easily implemented in Hector IR.

SCF Dialect in MLIR describes static control flow in a higher level abstraction than jumping between different blocks, which is similar with C program. This dialect is chosen as the input for HLS and is automatically lowered to ToR through a SDC scheduling algorithm. In addition, the unified representation makes it easy to transform between different manners. Hybrid scheduling like DASS [2] is naturally supported as a hybrid time graph in ToR. The generation procedure to HEC IR makes up an HLS tool supporting various manners including static, dynamic and hybrid scheduling.

5 VERIFICATION FLOW

With the development of hardware design, hardware verification is still a big challenge. Traditional verification tools often take SystemVerilog as input, and adopt SystemVerilog Assertion (SVA) to describe the properties that need to be verified. This technique highly relies on the quality of these properties, and is not scalable as well due to the low-level abstraction. Recent works mostly focus on the generation of hardware properties [1, 8, 11], and several works provide a higher level abstraction to reduce the scale of verification tasks [4]. However, these verification approaches are not suitable for hardware synthesis flows.

Existing verification frameworks [4, 12] need sophisticated procedures in which manual efforts are indispensable. These frameworks either need a golden model along with the equivalence mapping between implementation, or rely on verification experts to properly design properties. Other works [1, 8, 11, 13] focus on verifying a specific hardware model, which ignores the architectural information and cannot support arbitrary design either. These approaches only focus on a specific abstraction, which makes it hard to verify a conversion procedure like hardware synthesis.

A suitable IR with the multi-level paradigm like Hector makes it easier to store necessary information and to avoid extra analysis in the synthesis flow. For example, the scheduling information is kept during the synthesis procedure, so the mapping between computations and compute units is easily obtained. The equivalence between the functionality and implementation further improve the reliability and transparency of synthesis tools, and also brings the opportunity of debugging at different levels. Furthermore, due to the explicit representation of control logic, it's possible to automatically verify some necessary properties such as the liveness property in FSM structures and the connection validation between elastic modules.

6 CONCLUSION

In this paper, we present the benefits of multi-level paradigm for hardware synthesis. With enough expressivity and flexibility, Hector provides a promising way to build a new synthesis tool. The open-source framework also provides flexibility to customize synthesis approaches and allows users to explore advanced techniques. Furthermore, an suitable IR with multi-level paradigm like Hector makes it easier to store necessary information and to avoid extra analysis in the synthesis flow, which helps to make up an automatic verification approach with no human efforts.

REFERENCES

[1] Saranyu Chattopadhyay, Florian Lonsing, Luca Piccolboni, Deepraj Soni, Peng Wei, Xiaofan Zhang, Yuan Zhou, Luca Carloni, Deming Chen, Jason Cong, Ramesh Karri, Zhiru Zhang, Caroline Trippel, Clark Barrett, and Subhasish Mitra. Scaling Up Hardware Accelerator Verification using A-QED with Functional Decomposition. In *FMCAD*, 2021.

[2] Jianyi Cheng, Lana Josipovic, George A. Constantinides, Paolo Ienne, and John Wickerson. Combining dynamic & static scheduling in high-level synthesis. In *FPGA*, 2020.

[3] CIRCT Community. Circt: Circuit ir compilers and tools, 2021.

[4] Bo-Yuan Huang, Hongce Zhang, Pramod Subramanyan, Yakir Vizel, Aarti Gupta, and Sharad Malik. Instruction-Level Abstraction (ILA): A Uniform Specification for System-on-Chip (SoC) Verification. *ACM Trans. Des. Autom. Electron. Syst.*, 2019.

[5] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. Reusability is firrtl ground: Hardware construction languages, compiler frameworks, and transformations. In *ICCAD*, 2017.

[6] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling compiler infrastructure for domain specific computation. In *CGO*, 2021.

[7] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson. A compiler infrastructure for accelerator generators. In *ASPLOS*, 2021.

[8] Marcelo Orenes-Vera, Aninda Manocha, David Wentzlaff, and Margaret Martonosi. Autosva: Democratizing formal verification of RTL module interactions. In *DAC*, 2021.

[9] Fabian Schuiki, Andreas Kurth, Tobias Grosser, and Luca Benini. Llhd: A multi-level intermediate representation for hardware description languages. In *PLDI*, 2020.

[10] Amirali Sharifian and et al. μir -an intermediate representation for transforming and optimizing the microarchitecture of application accelerators. In *MICRO*, 2019.

[11] Eshan Singh, Florian Lonsing, Saranyu Chattopadhyay, Maxwell Strange, Peng Wei, Xiaofan Zhang, Yuan Zhou, Deming Chen, Jason Cong, Priyanka Raina, Zhiru Zhang, Clark Barrett, and Subhasish Mitra. A-QED Verification of Hardware Accelerators. In *DAC*, 2020.

[12] Lenny Truong, Steven Herbst, Rajsekhar Setaluri, Makai Mann, Ross Daly, Keyi Zhang, Caleb Donovick, Daniel Stanley, Mark Horowitz, Clark Barrett, and Pat Hanrahan. fault: A Python Embedded Domain-Specific Language for Metaprogramming Portable Hardware Verification Components. In *CAV*, 2020.

[13] Yue Xing, Huaixi Lu, Aarti Gupta, and Sharad Malik. Compositional verification using a formal component and interface specification. In *ICCAD*, 2022.

[14] Ruifan Xu, Youwei Xiao, Jin Luo, and Yun Liang. Hector: A multi-level intermediate representation for hardware synthesis methodologies. In *ICCAD*, 2022.