Extensible translation for high-level synthesis

Matthew Taylor · Imperial College London
m.taylor16@imperial.ac.uk

February 22, 2018

Abstract

Contemporary high-level synthesis (HLS) tools allow the description of processes intended for execution on reconfigurable hardware, typically FPGAs, to be written at a higher level of abstraction than traditional hardware description languages. Such higher level languages support certain forms of extensibility—for instance the ability to declare named functions and methods or support for macros and templates—which may be classified broadly as mechanisms for *linguistic extensibility*. However in another sense they are relatively inextensible, specifically with respect to the process of translation from source level to register-transfer level (RTL)—which we denominate here as *translational extensibility*—the translation process is essentially fixed with respect to the static semantics of the source language.

We argue that such a lack of translational extensibility gives rise to three principal issues: (i) it limits the range of valid translations the compiler may practically explore; (ii) it limits the level of abstraction attainable in the source language description; (iii) it precludes the possibility for precise control of the RTL description. In response we propose a systematic mechanism for exposing translational extensibility in the source language, which we claim provides an effective remedy for each of these issues.

We observe regarding issue (i) that as the level of abstraction is raised the range of valid RTL translations necessarily broadens, exceeding that which may feasibly be explored automatically. Thus HLS tools commonly employ some guidance from the programmer to find an acceptably performing translation. For instance C or C++ compilers make extensive use of `#pragma` directives to guide storage layout and operation scheduling decisions. Directives in particular have significant shortcomings: viewed as a domain-specific language they are highly restrictive, only capable of eliciting those outcomes presupposed by the compiler vendor; and further their language structure is remarkably weak, offering little support for abstraction and conditional or context-sensitive parameter selection. Resulting from such deficiencies, best current practice necessitates manual in-place deabstraction of source descriptions to guide to the compiler. We argue in contrast for a guidance mechanism not requiring such in-place deabstraction, and in which guiding instructions are written separately in the source language itself.

Given that raising the level of abstraction in the source description aggravates the need for more detailed guidance, it is clear that the expressiveness of the guidance mechanism establishes a practical limit to the level of abstraction supported by the source language: issue (ii). We therefore argue for a Turing-complete guidance language, for which the source language stands as a reasonable candidate. Guidance expressivity is not sufficient however, as it is simultaneously required to remove commitment to operational details in the algorithm description—this calls for cooperation with linguistic extensibility.

Finally concerning issue (iii), any sophisticated project eventually becomes concerned with performance at the hardware level (for certain limiting components). The inability to achieve an essentially optimal RTL translation for the bottleneck is usually unacceptable, as is the prospect of describing it explicitly in RTL. We advocate an approach by which any part of the translation process can be overridden, using standard principles of object-oriented extension, such that arbitrary source terms may give rise to arbitrary RTL, without sacrificing the automation benefits the compiler is usefully affording.

We present a proof-of-concept system, *Alde*, which embodies the essential structure called for above. Alde is both a language and synthesis system, and serves as a reference design for the approach and features we describe. The language semantics derive from Kernel [3], the system architecture takes inspiration from COLA [2], and the translation approach is akin to Micros [1]. We show how Alde may be applied to common programming challenges afflicted by one or more of the issues enumerated, and how its features provide an effective remedy.

References

[1] Shriram Krishnamurthi. *Linguistic Reuse*. PhD thesis, Rice University, 2001. URL `https://dl.acm.org/citation.cfm?id=934293`.

[2] Ian Piumarta. Accessible language-based environments of recursive theories, 2006. URL `http://www.vpri.org/pdf/rn2006001a_colaswp.pdf`.

[3] John N. Shutt. *Fexprs as the basis of Lisp function application; or,* `$vau`*: the ultimate abstraction*. PhD thesis, Worcester Polytechnic Institute, 2010. URL `http://web.wpi.edu/Pubs/ETD/Available/etd-090110-124904/unrestricted/jshutt.pdf`.