Abstract

Even though step-by-step refinement has long been seen as desirable, it is hard to find compelling industrial applications of the technique. In theory, transforming a high-level specification into a high-performance implementation is an ideal means of producing a correct design, but in practice it is hard to make it work, and even harder to make it worthwhile. This talk describes an exception.

We introduce the domain-specific language, Cryptol, and work up to a design experience in which theorem-based refinement played a crucial role in producing an industrial quality FPGA encryptor and decryptor for AES. Quite simply, we are unlikely to have succeeded without the technique. The Cryptol specification language was designed by Galois for the NSA as a public standard for specifying cryptographic algorithms. A Cryptol reference specification can serve as the formal documentation for a cryptographic module, eliminating the need for separate and voluminous English descriptions. Cryptol is fully executable, allowing designers to experiment with their programs incrementally as their designs evolve. Cryptol compilers can generate C, C++, and Haskell software implementations, and VHDL or Verilog HDL hardware implementations. These generators can significantly reduce overall life-cycle costs of cryptographic solutions. For example, Cryptol allows engineers and mathematicians to program cryptographic algorithms on FPGAs as if they were writing software.

The design experience we describe runs as follows: we begin with a specification for AES written in Cryptol, and over a series of five design stages we produce an industrial grade encrypt core. In each stage, we state theorems which relate the component behaviors in one stage with the corresponding behaviors in the refinement. The resulting cores, running at 350Mhz-440Mhz depending on the FPGA part, bear little relationship to the original, except that the step-by-step theorems ensured we had not gone astray.

We then repeat the pattern in generating a circuit for AES decrypt. While there are many similarities between encrypt and decrypt in AES, there are some crucial differences with regard to high performance. First concerns the generation of key material. The AES key is used as a seed for a specific pseudo-random number generator which produces key material for use in each of the AES rounds. For encrypt, the key-generator runs in sync with the action of encryption, so may be scheduled alongside it. For decrypt, they run counter to one-another, creating a major challenge to be overcome. Second, the generated key material has an additional transformation applied to it, which occurs deep in the middle of the high performing core.

Using theorems as stepping stones along the way, we redesign the key expansion algorithm so that it will run in sync with the decryption. We then trace parallel steps to the derivation of encrypt, establishing a series of commuting diagrams along the way. Whenever we confronted bugs in the development process, we produced many theorems to isolate the bugs, using theorems as a principled kind of printf. When the bugs were found and eradicated, we elided many of the temporary theorems, leaving behind those that provided important insights into the behavior of the code.

This talk is a story of the journey with demonstrations of the tool at work. Its ultimate message is to highlight the value of including a theorem facility within purely functional domain-specific languages.

Categories and Subject Descriptors D.3.2 [Language Classifications]: Applicative (functional) languages; D.3.2 [Language Classifications]: Specialized application languages

General Terms Algorithms, Design, Languages, Performance, Verification

Keywords cryptography, functional programming, domain-specific language, FPGA, AES, theorem-based refinement

Bio

Dr. John Launchbury is Chief Scientist of Galois, Inc. John founded Galois in 1999 to address challenges in Information Assurance through the application of Functional Programming and Formal Methods. Under his leadership, formerly as CEO, the company has grown strongly, successfully winning and delivering on multiple contract awards for more than a decade. John continues to lead Galois’ growing stature for its thought leadership in high assurance technology development.

Prior to founding Galois, John was a full professor in Computer Science and Engineering at the Oregon Graduate Institute School of Science and Engineering at OHSU. His instruction style earned him several awards for outstanding teaching, and he is internationally recognized for his work on the analysis and semantics of programming languages, and on the Haskell programming language in particular. John received First Class Honors in Mathematics from Oxford University in 1985. He holds a Ph.D. in Computing Science from University of Glasgow and won the British Computer Society’s distinguished dissertation prize. In 2010, John was inducted as a Fellow of the Association for Computing Machinery (ACM)