galois

Cryptol: The Language of Cryptography

CASE STUDY

Domain Specific Languages

A path to high assurance solutions

DOMAIN SPECIFIC LANGUAGES

A domain-specific language (DSL) is a programming language targeted at producing solutions in a given problem domain (as opposed to general-purpose programming languages). The idea is to empower subject-matter experts by allowing them to design solutions in terms they are familiar with and at a level of abstraction that makes most sense to them. Galois has been a leader designing DSLs for a variety of domains, including cryptography, operating system security policy, communication routing policy, and more.

APPROACH

In addition, a good DSL opens the way for powerful tool support: simulations for design exploration, automatic testing and automatic generation of test harnesses, generation of highly specialized code for multiple targets, and generation of formal evidence for correctness, safety and security properties.

These kinds of tools are possible for general-purpose languages as well, but generality limits how far one can go. The fact that a DSL is restricted to a particular problem domain allows for profound and often unexpectedly superior results, mainly because one can apply the wealth of domain-specific heuristics and expert knowledge available for that domain directly in these tools.

BENEFITS

Formalizing system requirements in a DSL.

With a traditional approach, system level requirements are usually expressed in natural language documents. A well-designed DSL allows the domain expert to quickly prototype their system and explore the design space through experiments and simulation. Once the domain expert is satisfied with the design, the DSL program can serve as a formal, high-level specification of the system requirements while remaining fully executable.

The reference specification may also be used as documentation (to help other domain experts and stakeholders understand and work with the system design and as part of any certification documentation) and as input to other useful tools.

Generating code for multiple targets.

A DSL is a programming language, and most DSLs come with an interpreter based "development environment" to directly execute programs for experimentation purposes. Furthermore, as DSLs tend to be smaller than general-purpose programming languages, it is usually easier to add a new target back-ends, generating executables that directly run on the target architectures. Indeed, DSLs may also be used to describe heterogeneous systems and to generate code for each of those different targets. For example, one might have a C monitor coordinating the behavior of many FPGA-based components. The DSL program would specify how to compile each component and how the components communicate.

Often, just generating C, Java, VHDL, or native machine code is not sufficient, as certification bodies (such as the FAA) have tight restrictions on the kinds of language constructs allowed (e.g., no pointer arithmetic, no general recursion). Again, as DSLs tend to be small, it is much easier to generate code that fits within those constraints, together with formal evidence that the generated code is functionally equivalent to the original program.

Verification and validation.

Any system needs good testing. The same features that enable a system designer to experiment with her designs via simulation also enable the generation of comprehensive test harnesses, specialized to the domain in question. Specific certification requirements that impact testing can be encoded into the test harness generator.

In addition, DSLs lend themselves well to the generation of formal evidence of correctness and robustness. DSLs are smaller and more focused than general-purpose languages, thereby reducing the size of the search space significantly. In addition, the capture of important formal evidence can be factored into the design of the DSL itself, taking care to provide constructs that support the domain expert while maintaining the tractability of formal techniques.

CRYPTOL: THE LANGUAGE OF CRYPTOGRAPHY

The Cryptol specification language was designed by Galois for the NSA as a public standard for specifying cryptographic algorithms. Cryptol tool-set provides the necessary components for deploying cryptographic modules across the entire software process, from specification and implementation to verification and certification. Cryptol tools significantly reduce overall life-cycle costs by addressing the key cost drivers in the deployment of crypto.

Providing complete specifications.

Cryptol is a formal specification language tailored to the unique needs of cryptography and cryptographic implementations. It is fully executable, allowing designers to experiment with their programs incrementally as their designs evolve.

Enabling re-use.

Cryptol provides a platform-neutral specification language that generates or guides implementations on multiple platforms. The Cryptol tools can generate C, C++, and Haskell software implementations, VHDL and Verilog HDL hardware implementations, or formal models for verification from the specification

.

Accelerating certification.

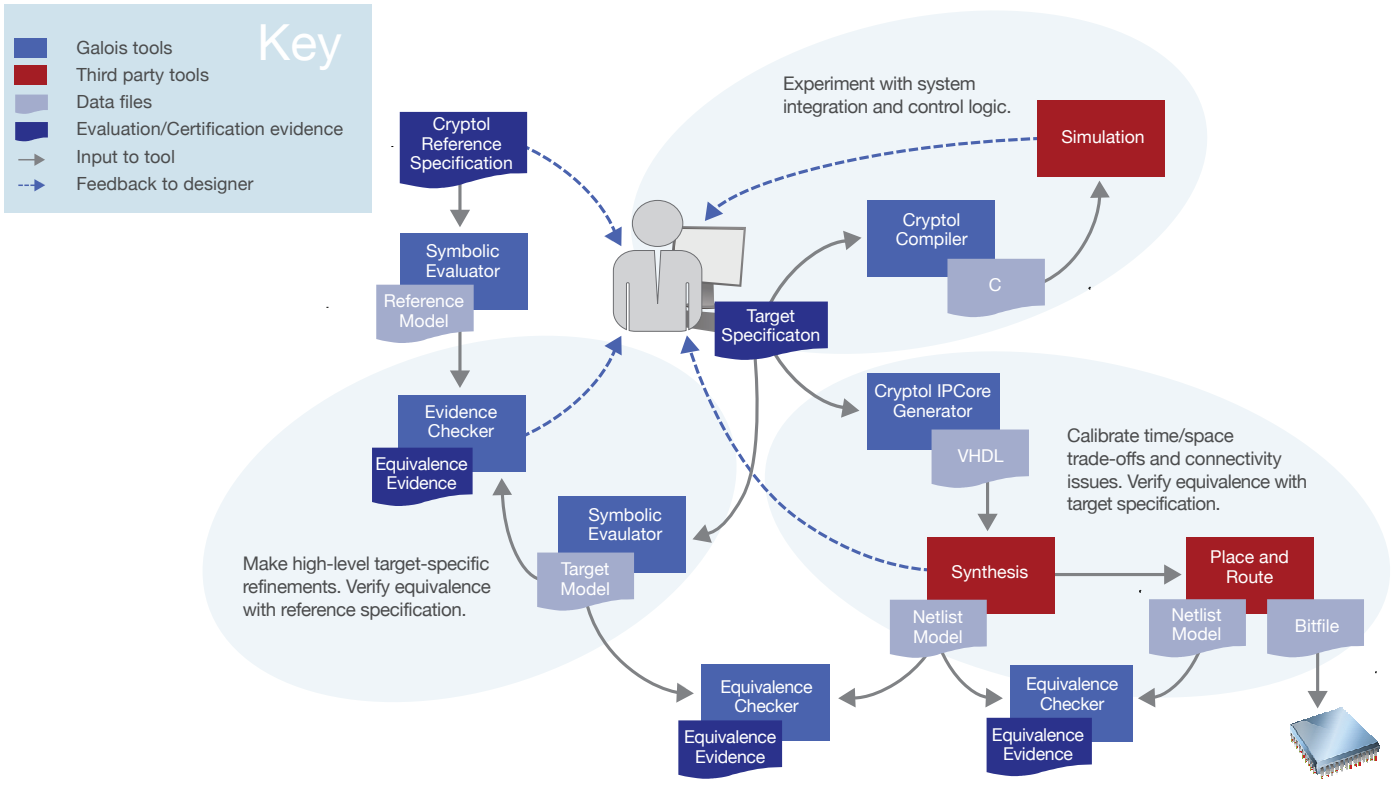
A Cryptol reference specification becomes the formal documentation for the cryptographic module, eliminating the need for separate and voluminous English descriptions. In addition, Cryptol verification tools show functional equivalence between the specification and the implementation at each stage of the toolchain.

Simplifying implementation.

Cryptol provides a refinement methodology to bridge the conceptual gap between specification and low-level implementation, thereby reducing time-to-market. For example, Cryptol allows engineers and mathematicians to program cryptographic algorithms on FPGAs as if they were writing software.

USING THE CRYPTOL TOOLS

The Cryptol development process begins with a reference specification for the cryptographic algorithm. A crypto developer refines the technology-independent, parameter-neutral specification into an implementation that targets a specific technology and associated parameter settings, resource constraints, and performance requirements.



CRYPTOL IN ACTION

A team of developers from Rockwell Collins, Inc. and Galois, Inc. has successfully produced high-speed embedded Cryptographic Equipment Applications (CEAs), automatically generated from high-level specifications. An algorithm core generated from a Cryptol specification for AES-256 and running in Electronic Codebook mode demonstrated throughput in excess of 16 Gbps. The "crypto waveform" logic uses the Model-Based Development language Simulink. These high-speed CEA implementations comprise a mixture of software and VHDL and target a compact new embedded platform designed by Rockwell Collins. Notably, almost no traditional low-level interface code was required to implement these high-performance CEAs. In addition, automated formal methods based tools provided by Cryptol proved that algorithm implementations faithfully implement their high-level specifications.

When feedback from the output stage to the input was introduced, thereby defeating the advantage gained by "unrolling" AES rounds, encryption performance for AES-256 still exceeded 1 Gbps while consuming less than 2% of the available programmable logic for the algorithm core. Most importantly, the Rockwell Collins/Galois team was able to design, implement, simulate, integrate, analyze, and test a complex CEA on the new hardware, including AES-256 and Galois Counter Mode (GCM), in less than 3 months.

SUMMARY

DSLs (domain specific languages) allow subject-matter experts to design solutions in using familiar concepts and constructs. Cryptol is an example of a DSL for expressing cryptographic algorithms. A feature-rich tool set has been developed at Galois and used with success at Rockwell Collins, Inc. The capabilities include:

- Create a reference specification and associated formal model.
- "Test" the specification against published test vectors and formal assertions about state.
- Refine the specification stepwise to one or more implementations, trading off space, time, and other performance metrics. Because these explorations are done in Cryptol, they can be done quickly.
- Compile the implementation for multiple targets: C/C++, Haskell, and VHDL/Verilog are currently supported.
- Equivalence check an implementation against the reference specification. VHDL implementations are converted into bitfiles for FPGAs using third-party FPGA vendor tools; verification tools can be utilized at several points in that vendor tool-chain.
- Equivalence check VHDL implementations not produced by Cryptol against a reference specification.

| galois |

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, Oregon 97204
T 503.626.6616 | F 503.350.0833
www.galois.com