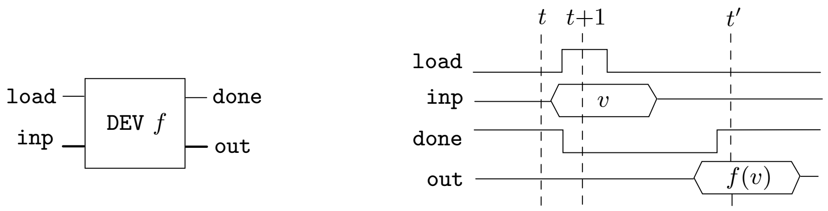
Proof producing synthesis of arithmetic and cryptographic hardware

Konrad Slind, Scott Owens, Juliano Iyoda, Mike Gordon
[Project web page: `http://www.cl.cam.ac.uk/~mjcg/dev/`]

We have implemented a novel compiler from a 'synthesisable subset' of higher order logic to clocked synchronous hardware. It is intended for components whose design *and implementation* needs very high assurance of correctness. Our compiler reliably generates working FPGAs running on Xilinx and Altera boards. We are using it to synthesise hardware implementations of arithmetic and cryptographic algorithms.

The compiler automatically translates a function f defined in higher order logic (typically using recursion) into a device `DEV` f that computes f via a four-phase handshake circuit on signals `load`, `inp`, `done` and `out`.



Compilation is by fully automatic proof in the HOL4 system, and generates a correctness theorem for each compiled function. The generated theorem has the form $imp \Rightarrow$ `DEV` f where implementation imp has been synthesised by structural refinement steps from the mathematical function f. Refinement proceeds until imp is at the level of clocked synchronous devices expressed in terms of flip-flops and combinational logic. Circuits of this form can be directly translated to Verilog, and then simulated and input to standard design automation tools (e.g. we use Quartus II for Altera FPGAs).

The theorem generated by the compiler can be combined with theorems about f proved using standard (possibly interactive) methods to derive functional correctness of imp.

The compiler is being applied to block ciphers and fast arithmetic circuits. It deals effectively with simple ciphers, such as TEA, and is being scaled to handle relatively complex ciphers such as AES. A pure hand-shaking approach leads to bulky circuits, so functions implementable by combinational logic are detected and compiled directly.

In collaboration with Joe Hurd of Oxford University, we hope eventually to synthesise devices for use in elliptic curve cryptography, but work on this is only just starting.

Users can modify the compiler's theorem proving scripts. A simple example is adding rewrites for local peephole optimisation. To fit large examples into FPGAs, proof-based whole program optimisation is needed. The synthesisable subset of higher order logic can be extended. For example, the core system can only compile tail-recursions, but separate proof tools can automatically generate tail recursive definitions to implement linear recursions, thereby extending the synthesisable subset of higher order logic to include linear recursion. This extension is being applied in the cryptographic applications.

Separate projects are using a verified formal model of the ARM instruction set architecture as a basis for software synthesis and verification. It is planned to generate systems consisting of devices synthesised by our hardware compiler linked via a formally modelled co-processor bus to ARM code generated by a proof-producing software compiler.

We think this work provides an interesting and novel perspective on how formal specification, refinement and proof can be fitted into design flows, especially when high assurance is important. In our talk we'll concentrate on the hardware compiler, but the ARM software compiler and verification projects makes the work relevant to high assurance Systems on a Chip that combine hardware and software.