TOWARDS THE CORRECT DESIGN OF MULTIPLE CLOCK DOMAIN CIRCUITS

Joe Stoy — Bluespec Inc.

Hardware designs these days typically make use of several clocks. This is partly to save power (by gating the clock to a part of the circuit temporarily not in use, and by ensuring that various parts of the design are not run unnecessarily fast), and also to allow the design to communicate with parts of the external environment running asynchronously. Hardware designs these days are, increasingly, "SoC"s, which bring together blocks (IPs) that have their own clocking requirements, either because they were designed independently, or because they are tied to various standards (external buses, video, audio etc.) It is important, therefore, to have mechanisms and techniques for designing correct circuits spanning many clock domains.

A language or notation which supports these techniques should have the following characteristics:

1. The aim should be to make the simplest situations trivial, other simple situations easy, and all situations expressible in a sensible way.
2. Thus in a design, or part of a design, with just one clock domain, clock handling should be completely implicit. Each instantiated module needs to be connected to "the" clock, and having to say so explicitly adds unnecessary clutter.
3. *Clock* should be a datatype of the language, and the type system should ensure that clocks are never confused with level-sampled signals.
4. The system should keep track of which signals are clocked by which clocks, and ensure that no signal crosses a clock-domain boundary without the use of appropriate synchronizing logic.
5. For efficiency's sake, the system should be able to recognize when two clocks are driven by the same oscillator (that is, they differ only by gating); this should be exploited to simplify domain-crossing logic between them.

The presentation will address, by way of example, how these features are realised with Bluespec SystemVerilog.

Bluespec SystemVerilog is a high-level behavioural hardware description language. As with Verilog, designs are structured into modules. The internal behaviour of a module is specified by *rules* (instead of always-blocks), which have a condition part and an action part. Modules communicate with their environment by the *methods* of their interfaces, which may be invoked by rules in other modules. Taken together, the rules of a design have the semantics of a state-transition system, or term-rewriting system (TRS): that is, they execute atomically. The Bluespec compiler generates logic which schedules as many rules as possible in each clock cycle; but the overall effect of each cycle is exactly the same as if the rules executed one at a time in some order—this greatly simplifies the analysis of a design's correctness. The language is strongly typed.

Clock is a data type of the language, and its values are treated as first-class citizens except where this is clearly inappropriate. A clock consists of two signals: an oscillator and a gating signal: if the gating signal is high, the clock is assumed to be clocking. If two clocks have the same oscillator, they are said to be in the same family, or "related". For communication between domains clocked by related clocks, no special domain-crossing logic is necessary: Bluespec's normal interface protocol can be exploited to provide all that is necessary.

If two domains are clocked by unrelated clocks, communication between them must go through special synchronising logic. The compiler ensures that all methods invoked by any one rule (or other method) are clocked by related clocks. The synchronising logic is provided by special-purpose modules written in Verilog. The wrappers written for these modules specify which of their methods are associated with which clock; so, provided the primitives are correctly written, the compiler enforces the requirement for appropriate logic between unrelated domains. The Bluespec library provides many such primitives, and users are encouraged to write others to deal with special situations and requirements. This provides an extensible framework: although Bluespec cannot of course check the correctness of the primitives, the framework ensures that accidental misuse of clocks is impossible.

The machinery used by the compiler for keeping track of all this is also made available to the designer, so writers of library routines can ensure that they take account of the clocking of their dynamic arguments.