PRIMARY PERFORMANCE PROBLEMS OF RECONFIGURABLE SILICON

William Bricken
November 2002

FPGAs from companies such as Xilinx and Altera offer the capability of dynamic reconfiguration of the circuit functionality performed by the silicon architecture. This is achieved by either LUT or MUX-based fine-grain computation units embedded in a matrix of routing resources. The logical functionality is placed within a collection of fine-grain resources, the resources are routed to connect the logic, and finally the placed and routed network is timed to determine the delay incurred by the placement and routing configuration.

This approach to reconfigurable computing evolved over a decade ago, equivalent to hundreds of years of evolution compared to human rates of growth. For today's logic networks of hundreds of thousands of gates, the antiquated place-and-route approach to reconfigurable logic is dysfunctional. Basically there are two very significant problems with this approach.

Problem 1: Logic functions have a shape, a structure. LUT and MUX fine-grain computational units also have a shape. Placing any logic structure into a given LUT or MUX structure is similar to attempting to place a round peg into a square hole. Reconfiguration is a misnomer, the hardware architecture of today's computational units is rigid, not flexible. A four-input LUT can accommodate any four-input Boolean function, so it places a rigid "four-input" structure on any given functional structure. The problem is that logic does not fit nicely into four-input chunks.

Problem 2: Logic functions can be deep or wide, it is a generally known technique how to convert any function into either a deep or a wide format. Wide formats (called SoP, PoS, CNF, and two-level) require few levels of logic gates (or LUTs or MUXes), in exchange they require tremendous routing resources. Deep formats (called multilevel) impose a longer delay by passing through more levels of logic gates, in exchange they require the least routing resources. The problem is that no generally know transformation techniques are known to optimize multilevel logic.

The structural problem occurs in today's FPGAs as a need to floor-plan, to know which computational resource has the best location for which chunk of logic. The multilevel problem is to know how best to trade-off routing resources for timing delays. Obviously, these two problems interact; therein lies the FPGA performance problem.

FPGA performance can be degraded in two distinct ways. The structural problem leads to areas of silicon resource that get isolated, chunks placed in computational units use up routing resources so that other computational units are unreachable. With today's abundance of computational resource, this alone is not a significant problem. The actual problem is that different computational units require different lengths of routing, and thus incur different wire propagation delays. Routing variation created by the structural problem cannot be avoided since the physical location of silicon resources is rigid, so that the placement of logic must conform to the locational structure of the silicon, rather than to the logical structure of the desired functionality.

The multilevel problem is not really a problem, it is more of a catalyst that exacerbates the structural problem by placing structural demands on the routing resources themselves that are independent of the demands placed upon routing by the structural problem. The multilevel problem is a problem because there are no efficient tools to disentangle the two demands upon routing resources.

In sum, both structural and multilevel problems show up as routing problems. In current FPGA architectures, designers encounter routing saturation and timing variation. Routing saturation means that a designer cannot place the desired logic functionality on a given FPGA chip because the resources of the chip become inaccessible. FPGA manufacturers respond to this problem by putting a lot more routing resources on the chip, making the placement problem much more complex while not addressing the essential structural variation that drives the placement problem.

Timing variation means that a designer never knows the signal delay time of a given functional layout, not until the functionality is actually laid out into the physical resource matrix. When the timing is known, a different problem immediately shows up, that of timing closure. Timing closure means that different parts of the silicon resource, as laid out, take different amounts of time to complete their functional task. The timing performance of the chip is the timing performance of the slowest path, the weakest link in the routing layout. Timing specification usually specify a particular performance requirement, a signal must be at a given point within a given amount of time. Timing closure is empirical twiddling with the worst case delay paths to bring them into conformance with the timing specifications. Unfortunately, changing a path means changing the routing of the entire chip. Timing closure is the iterative refinement of the design to reach a timing performance by redoing essentially the entire design for every small change in the critical path length.

Timing is brittle; when a design changes, even by the smallest amount, the timing usually changes, erasing the prior closure and forcing yet another entire redesign. It is often falsely believed that by manipulating the multilevel nature of the logic, that timing closure can be reached. This is because tools for changing multilevel formats are known, and tools for changing physical routing resources are not known.

The Solution

The solution to both structural and multilevel problems lies in changing the physical silicon architecture. The new architecture in turn requires new algorithms that can address structural and timing variation directly.

The structural problem is not a problem, the answer is to waste some silicon with less than totally efficient utilization. The timing problem caused by variable routing is addressed by rigidly pipelining all computational units. Pipelining is the technique of placing registers at regular intervals throughout the computational resource structure, so that signals move lockstep through the computational units, and thus remove timing variation. For the pipelining approach to work, there must be sufficient routing resources to convey signals from one register bank through the next computational unit and into the next register bank. This can be achieved by using large chunks to enact logical functionally, not 4LUTs that accommodate about six logic gates, but blocks that accommodate thousands of logic gates. Within the blocks, smaller units must be somewhat fluid, not forcing logic into computational chunks, but rather accommodating any structure. For this to work, the blocks must be maximally multilevel, since wide functional formats overwhelm routing resources. This approach is known as a sea-of-gates.

The architecture that has formed is such: small reconfigurable cells of a few logic gates richly connected by dedicated routing between cells, and arranged in multilevel layers that are pipelined consistently at a given depth. The required configuration and layout software must be able to optimize functionality into multilevel blocks.

For pipelined timing, physical restrictions dictate the cycle speed of the above architecture. Taking 300 MHz as a working example, blocks can be five-levels deep and about sixteen cells wide. To place and route arbitrary logic into these chunks requires multilevel optimization and a local way meet the depth and breadth restrictions of blocks.