

Optimizing FPGA Power with ISE Design Tools

You can reduce the power dissipated in Virtex-4, Virtex-5, and Spartan-3 designs through new power-driven back-end flows.

by Subodh Gupta, Ph.D.
Staff Software Engineer, Design Software Division
Xilinx, Inc.
subodhg@xilinx.com

Jason Anderson, Ph.D.
Principal Engineer, Design Software Division
Xilinx, Inc.
janders@xilinx.com

In the more than 20 years since Xilinx invented the FPGA, research and development has produced dramatic improvements in FPGA speed and area efficiency, narrowing the gap between FPGAs and ASICs and making FPGAs the platform of choice for implementing digital circuits. Today, power consumption is a rising concern for FPGA vendors and their customers. Reducing the power of FPGAs is key to lowering packaging and cooling costs, improving device reliability, and opening the door to new markets such as mobile electronics.

Xilinx is taking the lead in offering low-power FPGA solutions. In this article, we'll illustrate the application of computer-aided design (CAD) techniques, such as those incorporated into Xilinx® ISE™ version 9.2i software, for effective power reduction.

Power dissipation in CMOS circuits comprises both static (leakage) power and dynamic power. Dynamic power is caused by transitions on the signals of a circuit and is governed by the equation:

$$P_{dynamic} = \frac{1}{2} \sum_{i \in signals} C_i \times f_i \times V^2$$

where C_i represents the capacitance of a signal, i ; f_i is referred to as “switching activity” and represents the rate of transitions on signal i ; and V is the supply voltage.

Static power, on the other hand, is consumed when a circuit is in a quiescent, idle state. Static power results from leakage currents in off transistors, primarily sub-threshold and gate-oxide leakage. An off-MOS transistor is an imperfect insulator allowing sub-threshold leakage current to flow between its drain and source terminals. Gate-oxide leakage is caused by tunneling current through the gate terminal of a transistor to its body, drain, and source.

Technology scaling, such as the recent move to 65 nm, means lower supply voltages and smaller transistor dimensions, leading to shorter wire lengths, less capacitance, and an overall reduction in dynamic power. Smaller process geometries also mean shorter transistor channel lengths and thinner gate oxides, producing an increase in static power as technology scales.

Power Dissipation in FPGAs

The programmability and flexibility of FPGAs make them less power-efficient than custom ASICs for implementing a given logic circuit. The FPGA configuration circuitry and configuration memory consume silicon area, producing longer wire lengths and higher interconnect capacitance. Programmable routing switches attach to the pre-fabricated metal wire segments in the FPGA interconnect and add to the capacitive load incurred by signals.

Most dynamic power in an FPGA is consumed in the programmable routing fabric. Likewise, static power is proportional to total transistor width. The interconnect comprises a considerable fraction of an FPGA’s transistors and therefore dominates

leakage. Consequently, the interconnect fabric should be a prime target for FPGA power optimization.

Certainly, power can be addressed through process technology, hardware architecture, or circuit-level changes. Virtex™-5 FPGAs, for example, contain “diagonal” interconnect resources, allowing connections to be made with fewer routing conductors and reduced interconnect capacitance. At the transistor level, Virtex-4 and Virtex-5 FPGAs employ triple-oxide process technology for leakage mitigation. Three possible oxide thicknesses are available for each transistor, depending on its speed, power, and reliability requirements. The proliferation and increased use of hard IP blocks such as DSPs and processors can also lower power consumption versus implementing such functionality in the standard FPGA fabric.

It is also possible to reduce power without incurring any costly hardware changes. You can tackle power issues through new power-driven CAD algorithms and design flows, such as those in ISE 9.2i software.

Power Optimization in ISE 9.2i Design Tools

ISE software version 9.2i incorporates power optimization in placement, routing, and through a post-routing technique that reduces power within logic blocks.

Placement

The core algorithm in the Xilinx placer uses analytical (mathematical) techniques. It begins with an initial overlapped place-

ment of a design and then uses a force abstraction to move logic blocks away from highly congested regions, ultimately producing a feasible, overlap-free placement. Once analytical placement is finished, swap-based local optimization is run on the placed design to further refine the placement. The traditional cost function used in our placer considers wire length and timing in this equation:

$$Cost = a \times W + b \times T$$

where W and T are the wire length and timing costs, respectively, and a and b are scalar weights. The values of a and b can be set according to the relative priority of timing versus wire length. The placer costing scheme is illustrated in Figure 1.

As actual routes are not available during placement, the wire length cost depends on wire length estimation. Likewise, the timing cost depends on the user-supplied constraints and estimates of connection delays. To optimize power, we have extended the analytical placement and local optimizations by adding a power component to the cost function, as shown on the right-hand side of Figure 1. The modified cost function is as follows:

$$Cost = a \times W + b \times T + c \times P_{dynamic}$$

where $P_{dynamic}$ is the estimated dynamic power (as defined previously) and c is a scalar weight. You can extract signal switching activity data from simulation and supply

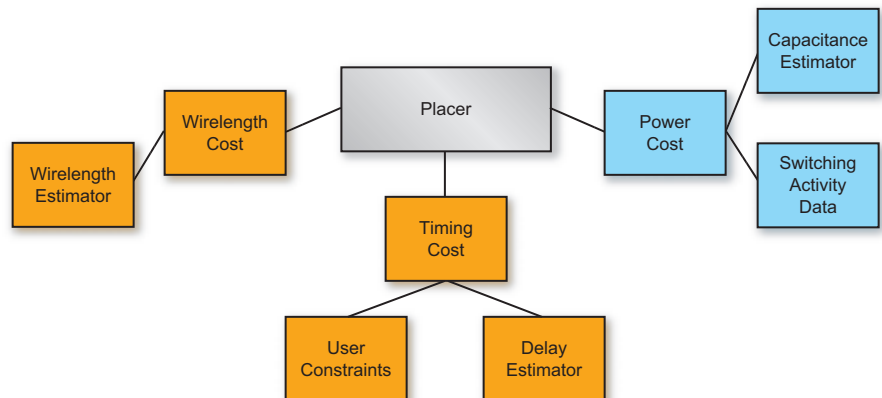


Figure 1 – Placer costing scheme incorporating power

The place and route optimizations discussed in this article aim to lower power in the interconnect fabric.

it to the tool. Alternately, if you do not provide any switching activity data, then the tool assumes a default switching activity for the primary inputs and sequential outputs and propagates activity to the rest of the signals, considering the logic functionality. For best results, user-supplied switching activity data is required.

During placement, the capacitance of a signal is not known and must therefore be estimated. We have constructed an empirically derived capacitance estimation model based on signal parameters available during placement:

$$C_i = f(FO_i, XS_i, YS_i)$$

where f represents a generic mathematical function, C_i is the capacitance of signal i , FO_i is the number of fanouts of signal i , XS_i is the X-span and YS_i is the Y-span of signal i in the placement, respectively. These parameters are architecture-independent and are readily available during placement.

To build the model, we extracted the capacitance, number of fanouts, X-span, and Y-span for each signal in a suite of designs collected from Xilinx customers. We then used least-squares regression analysis to fit the capacitance to a quadratic function of the model parameters. On average, the analytical equation gives a 30% error for a wide range of designs.

Routing

Once the logic blocks are assigned to physical locations on the FPGA, we must route the connections between the blocks. The router uses a negotiated congestion-routing algorithm, where during initial iterations shorts between signals are permitted. In later iterations, the penalty for creating shorts is increased, until no routing conductor is used by more than one signal. Timing-critical connections are routed to minimize their delay, which involves computationally intensive RC delay calculations. Most connections, however, are not timing-critical.

In the power-aware router, we choose to optimize the capacitance of such non-critical connections. To achieve this, we changed the router's cost function for non-timing critical connections to consider capacitance, as opposed to the prior approach of basing cost on other factors such as estimated delay or scarcity. To illustrate the algorithm, consider the routing graph of Figure 2.

Each node in the graph represents a routing conductor or logic block pin and

mented with built-in automatic input vector generation by attaching a linear feedback shift register-based (LFSR-based) pseudo-random vector generator to the primary inputs. This permitted us to perform board-level measurements of dynamic power without requiring a large number of externally applied waveforms.

The industrial designs were mapped into Spartan-3, Virtex-4, and Virtex-5 devices. Results showed that dynamic power was reduced as much as 14% for Spartan-3

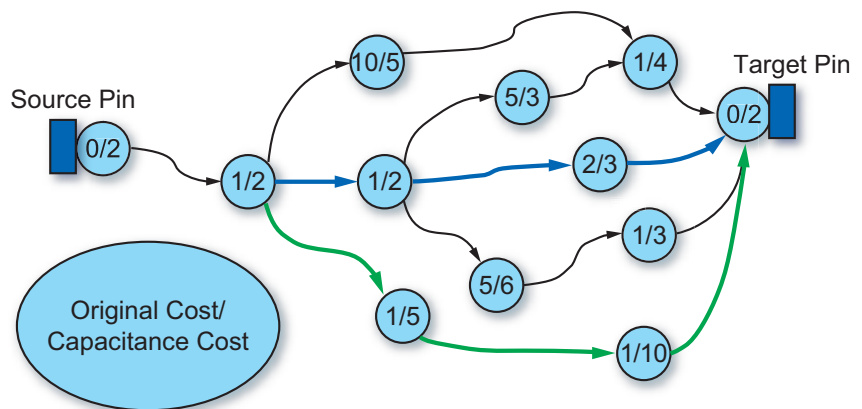


Figure 2 – Routing graph with capacitance costs on nodes

each edge represents a programmable routing switch. The router must select a path between the source and target pins. The original cost and capacitance cost of each node is shown internal to each node in the figure. To minimize the original costs, the routing between the source and target pin would have taken the path shown in blue. However, in the power-aware flow, the router will use the path shown in green, which has lower overall capacitance.

Results from Power-Aware Placement and Routing

A set of industrial designs were placed and routed using both the traditional place and route flow, as well as the power flow described earlier. The designs were aug-

FPGAs, 11% for Virtex-4 FPGAs, and 12% for Virtex-5 FPGAs. On average, across all designs, dynamic power was reduced by 12% for Spartan-3 FPGAs, 5% for Virtex-4 FPGAs, and 7% for Virtex-5 FPGAs. For all families, on average, the speed performance hit was between 3% and 4%, which is small and we believe acceptable in power-conscious designs. Considering that these are initial results coming from software changes alone, we consider the power benefits quite encouraging.

Reducing Power within Logic Blocks

The place and route optimizations discussed in this article aim to lower power in the interconnect fabric. We also devised an approach to reduce power within logic

blocks, specifically in look-up-tables (LUTs) when not all LUT inputs are used (Figure 3). A K-input LUT is a small memory capable of implementing any logic function with no more than K inputs. Figure 3 shows how a hypothetical three-input LUT (having inputs A1, A2, and A3) implements a two-input logical-AND function. The truth table for the logical-AND is shown in the LUT SRAM contents on the left side of the multiplexer tree.

Note that input A3 is unused in Figure 3. Normally, unused inputs are treated as “don’t cares” and assumed to be either 0 or 1. Consequently, to account for this in the case of Figure 3, the Xilinx software “replicates”

propagated to the LUT output.

This optimization entails detecting unused LUT inputs at the post-routing stage and setting LUT memory contents to be logic-0 so as to eliminate unnecessary internal toggling and without disrupting the logic functionality. Returning to the example of Figure 3, the bottom half of the LUT memory contents would be set to logic-0. No toggling would occur on internal nodes n1 and n2, and therefore no dynamic power would be consumed by charging or discharging n1 and n2.

We conducted board-level power measurements to evaluate this optimization on industrial designs and observed that it

Power-aware logic synthesis and activity-driven technology mapping to LUTs are well-studied in the literature and will yield considerable power reductions for Xilinx FPGAs. In placement, improved capacitance estimation accuracy will yield even higher power reductions.

Two areas that we feel are particularly fertile are glitch and leakage optimization. Glitches are spurious transitions that occur on signals caused by unequal path delays in circuits. Such transitions are unnecessary, yet they play a significant role in dynamic power. CAD techniques to mitigate glitches include equalizing path delays or inserting registers along the most “glitchy” paths.

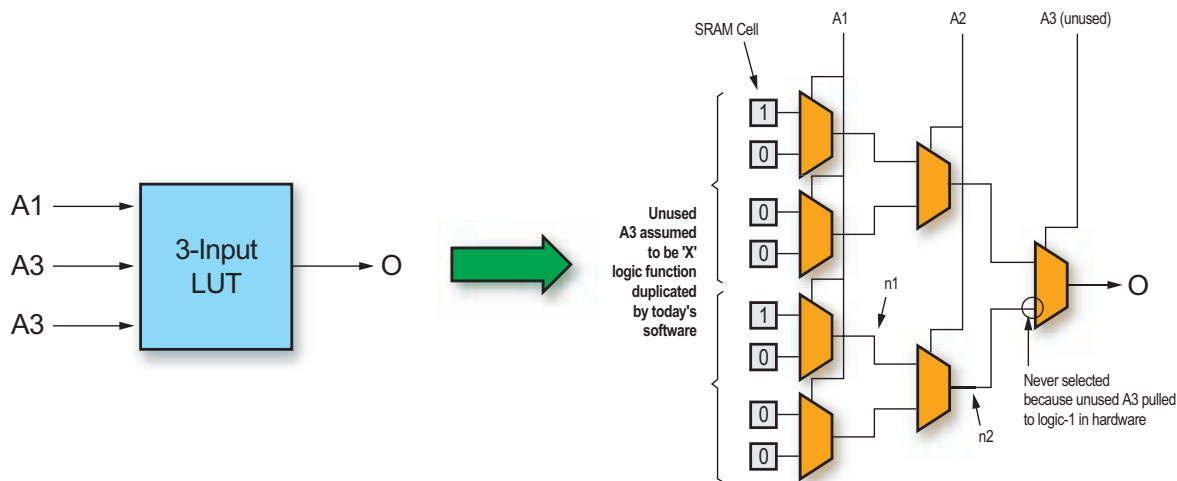


Figure 3 – Optimizing power within a LUT

the logic function in both the top and bottom half of the LUT SRAM memory contents. Unused LUT inputs occur frequently in customer designs, especially in Virtex-5 designs, where LUTs have six inputs.

In the Virtex-5 hardware, unused LUT inputs are pulled high to logic-1; this property is the root of our optimization. If A3 is pulled to logic-1, the bottom input to the deepest two-input multiplexer in the tree is never selected. However, because the logic function is replicated in both the top and bottom half of the LUT memory contents, toggling can occur on internal multiplexer nodes n1 and n2 based on changes in signal inputs A1 and A2. This toggling consumes dynamic power needlessly, as transitions on n1 and n2 are never

provides several percentage points of dynamic power savings. These results are especially promising, as this optimization is “no cost” in the sense that it can be carried out post-routing and causes no area or performance penalty.

Conclusion

Many future directions exist for further reducing power in design tools. In front-end HDL synthesis, proven optimizations from the ASIC domain can be adapted for FPGAs, such as clock gating and operand isolation. As well, FPGA-specific power optimizations can be applied, such as mapping logic into available block RAMs (which can be used as large ROMs) instead of using LUTs and the general fabric.

Leakage in a digital CMOS circuit depends strongly on the circuit’s applied input state. Therefore, one approach to leakage reduction in CAD is to automatically alter the circuit so that its signal values spend more time in low-leakage states.

The results show that significant strides have already been made in reducing power through ISE design tools. We are optimistic that the future is bright for achieving additional power reductions in software. Power-conscious solutions comprising power-aware CAD algorithms and power-optimized devices such as Virtex-5 FPGAs form a compelling story. Continued advances in low-power software and hardware will open the door for Xilinx FPGAs entering new power-sensitive markets. ●●●