The performance of every FPGA design will improve if you follow a few simple guidelines.

Design Tips for High-Performance FPGA Design

by Stephen L. Wasson

Automatic tools do not necessarily automate success. When given less than optimal input, EDA tools often generate less than optimal output. When appropriate design tradeoffs aren't made, results for designs pushing performance limits may be disappointingly slow. Without designing to the target technology, designs pushing performance limits may not fully route. Still, with a little knowledge of the device architecture and some hand-crafted assistance, the automatic partition, place, and route tools will produce excellent results.

Overview Although the following design tips are particularly applicable to the Xilinx 3K and 4K FPGAs, these guidelines are generally adaptable to other FPGA architectures. These guidelines will significantly improve FPGA implementation results—especially where in route-resource bound architectures are concerned. Although illustrations are provided in schematic representation—where the "power hooks" currently are—these design hints are also pertinent in text-based design environments.

Some tips for designers in pursuit of higher performance designs are as follows: An intelligent pin assignment will maximize device utilization and flexibility, and a route budget analysis maximizes performance. Synchronous design eases analysis, while gated clocks complicate analysis. Pipelines, state-per-bit state machines, and linear-feedback shift registers and prescaled counters all increase circuit speeds. Combinatorial clock-enables and duplicate circuits increase routability. Customized macros improve mapping and placement, while over-constraining the design hampers routing and performance.

The design tips that follow will result in increased performance margins, fewer design iterations, reduced time to market, as well as higher performance results.

Never pre-assign pins The fastest way to get into routing trouble is to arbitrarily assign package pins before taking device architecture into consideration. Conversely, a well-planned pin assignment makes the difference between a successful or unsuccessful route, a mediocre or stellar result, a low- or high-performance margin, and the number of future FPGA modifications that can be absorbed without PCB redesigns. When considering Xilinx 3K and 4K architectures, it is important to note that bused data flow is preferentially horizontal, due to horizontal tristate long lines. Furthermore, data structures—such as registers and counters—are preferentially vertical due to the vertical clock-enable long lines in the 3K and carry chains in the 4K. Therefore, data bus I/Os are best assigned to the left and right device rails, while control signals are best placed along top and bottom rails. Temporarily assigning these I/Os while verifying the routability of the entire design ensures the greatest opportunity for implementation success. The largest margins will be obtained if the final pin assignment is left open until the end of post-route analysis. It is more likely that design schedules will demand that some pinout be turned over to PC design so board layout can proceed in parallel to FPGA design. Under these conditions, the best compromise is a wise assignment of the data buses and critical control signals; the remaining pin assignments should be based on an average of a few preliminary routes. Hand-crafted pin assignments help to avoid re-implementations, reduce design iterations, save time, and shorten the FPGA design

cycle. (Note: Optimal pin assignments can be produced by floorplanning—a procedure that considers design data flow in conjunction with device architecture. Floorplanning is beyond the scope of this article and will be the subject of a future article.)

Perform route budget analysis Route budget analysis (see Sidebar figure) provides a method for predicting which circuit implementations will need special attention, either in redesign, mapping, or placement constraints.

Figure 1
State-per-bit state machines

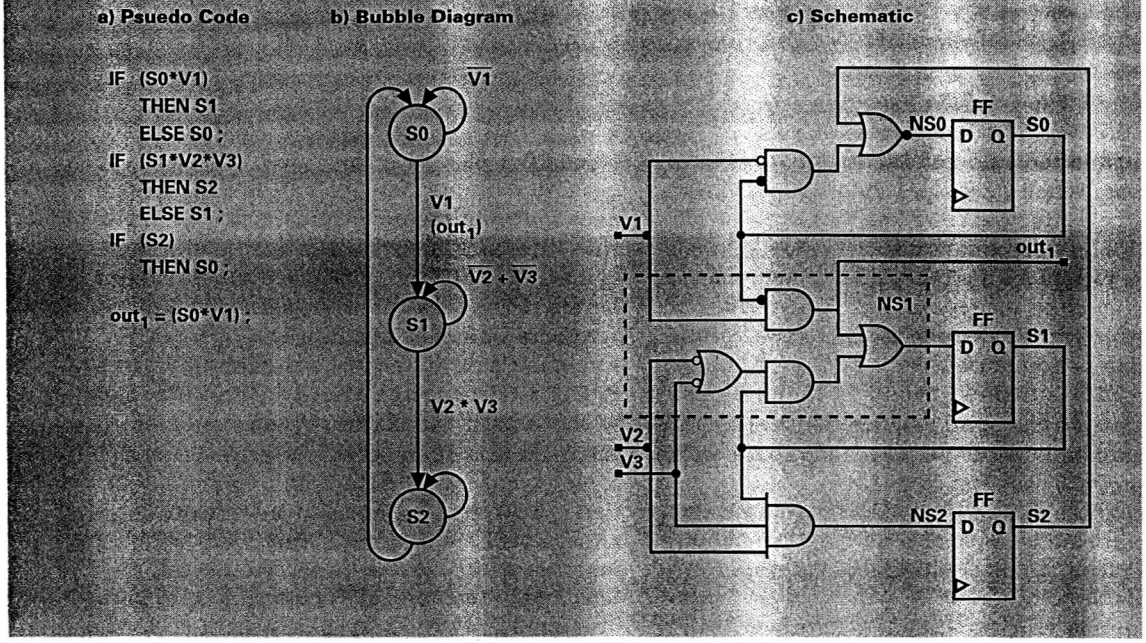


Figure 1. Three representations for the same state machine of three states S[0:2], three synchronous input variables V[1:3], and one output function out_1. Figure 1c illustrates the schematic state-per-bit implementation most conducive to "route budget" analysis and manual mapping. State 0 is low active, and asserted at power-up (darkened bubbles). The next-state function for state 1 (bounding box) is a function of five variables; therefore, in an architecture with four-input function generators, this function will have to map into multiple function generators. If output function out_1 is implemented as above, it will also have to map into its own function generator. A faster implementation for out_1 would be to duplicate the source gate. (Note: The synthesis for next-state 1, function NS1, may not be as terse as the schematic version. An alternate—and slower—implementation could be (S1 * /V2) + (S1 * /V3), in which each AND term would end up in its own function generator.)

In this analysis, logic element delays are subtracted from the target clock period to calculate the allocatable time remaining for all intermediate routes. If the average allocatable is reasonable—about equal to a typical logic cell delay—then that implementation should route without further consideration. Performing such discovery prior to compilation should eliminate most non-performance surprises, as well as determine the maximal performance obtainable from a particular implementation. (Note: Currently, text-based design entry isn't as conducive to this kind of analysis as graphic-based entry.)

Design synchronously Designing synchronously whenever possible has these distinct advantages in the FPGA environment: timing-driven routers will have fewer complicated paths to analyze and will reach completion sooner; post-route delay analysis tools will have fewer and shorter paths to trace, making it easier to extract performance results; and simulation tools will be less confused by fewer timing violations and will thus be more useful. In the Xilinx 3K and 4K devices, the asynchronous path through the configurable logic block (CLB) is the slowest of all CLB paths and degrades circuit performance. The flip-flops within the CLB share with the same asynchronous input, effectively reducing mapping and placement flexibility. In general, frivolous asynchronous design does not optimize FPGA, computing, or human resources. Consider a design that has "just-in-case" asynchronous resets. Some of these resets must be considered in the synchronous path analysis, while others can be ignored. Determining whether a reset can be ignored may take 10 minutes for every path analyzed. By comparison, in a fully synchronous design, the performance analysis can be limited to a single query—about a 10-second task.

Don't gate clocks Ideally, keep the number of unique clocks in the design to one. Although gating clocks may be suitable for simple devices, in architectures with clock enables, this introduction of multiple time domains is unnecessary. Moreover, it introduces unnecessary skew. No advantage is gained, and the risk of problems is greater when signals gate the same clock from which they originate. Instead, use the gating signals as clock-enables. It is far easier to analyze a design with fewer clocks.

Pipeline Pipelining—the introduction of a sequential element into a combinatorial path—is an easy way to improve performance by trading resources and latency for speed and routability. This holds especially true for pre-decoding. Pipelining state machine next-state functions, counter decodes, and arithmetic intermediate carries are all excellent speed enhancers (see Figures 2 and 3).

Implement state-per-bit state machines In the days before PALs, state machines were implemented as timing chains, that is, as unencoded sequences of flip-flops. The advent of devices with large product terms made it easy to encode large state machines (above 32 states) without performance penalties. However, in architectures with low fan-in function blocks, these encoded state machines, although still easy to implement, are significantly slower due to the increase in intermediate logic levels. In the flip-flop rich Xilinx 3K and 4K architectures, the state-per-bit or "one-hot" implementation (see Figure 1) is advantageous for machines with as few as four states. By trading the next-state function generators for state flip-flops, intermediate logic levels are reduced for both the state-per-bit state machine and the associated output function generators. While encoded state machines with decoded output functions tend to become placed in a tight ball—thus increasing the routing snarls of the state bits—state-per-bit state machines may place and route in a serpentine fashion resulting in an increase in both routability and performance.

Use linear-feedback shift registers If a function requires a terminal count and not a binary output—such as a divide-by-n counter—(see Figure 3) then linear-feedback shift registers (LFSRs), are more space and route efficient (see Figure 2) than their binary counterparts. Since LFSRs have minimal next-state logic and no carry logic, they are even more logic-resource efficient than binary counters. They produce 2^{n-1} states for n bits. LFSRs also have the same serpentine routing advantage as state-per-bit state machines, and therefore rarely require placement. Although the count sequence may be discomforting, the LFSR operability may easily be verified during simulation. Even with their random count sequence, the non-intuitive LFSR count sequence can still be used for such functions as RAM address counters or FIFO pointers—provided that all RAM or FIFO ports use the same LFSR polynomial. (Note: LFSR polynomials can be obtained either from a math text relevant to the subject or a small executable [LFSR.zip] freely available on the HighGate BBS at 408-255-9742.)

Prescale (FAST) binary counters If a binary count sequence is required, consider implementing a prescaled counter (see Figure 3) in which the prescalar terminal count (PTC) can be used as the count-enable (CE) to the remaining counter stages. One advantage of this technique is that the remaining stages can then be implemented with a daisy-chain-style carry that will operate at a fraction of the clock frequency that is a function of the prescale. An example of this is using a 2-bit prescalar to allow the remaining stages to operate at one-fourth the clock rate. A further

Figure 2
Linear feedback shift register

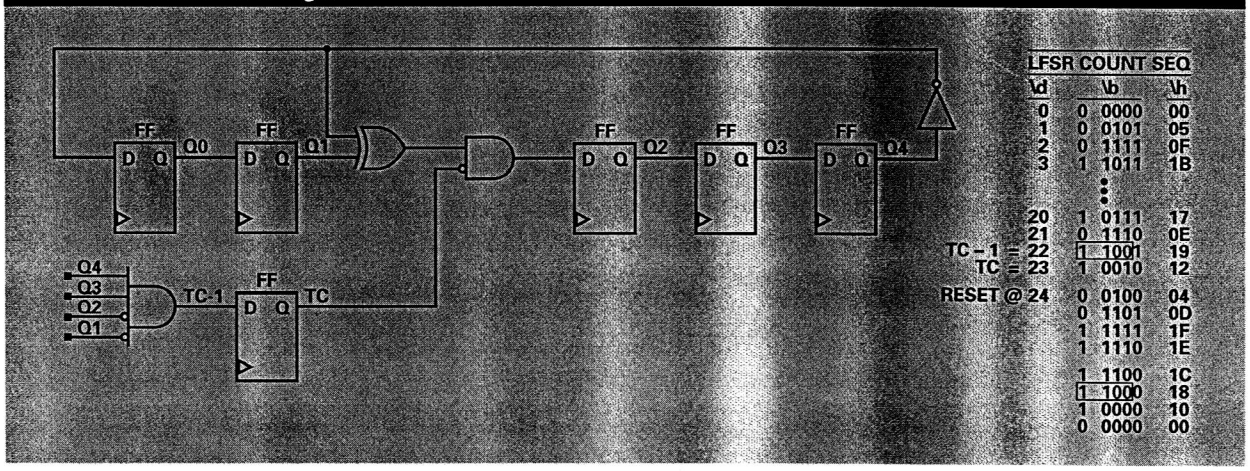


Figure 2. If a binary count sequence is not needed, a linear feedback shift register is more routable than its binary counterpart. Above, a divide-by-24 counter is implemented as an LFSR with pre-decoded and pipelined TC. Additional performance is gained by optimizing the TC decode to a function of four variables (in this example, Q[4:1] are sufficiently unique), and by minimizing the reset net (by resetting only the counter bits which would set on the next clock).

Figure 3a
Prescaling

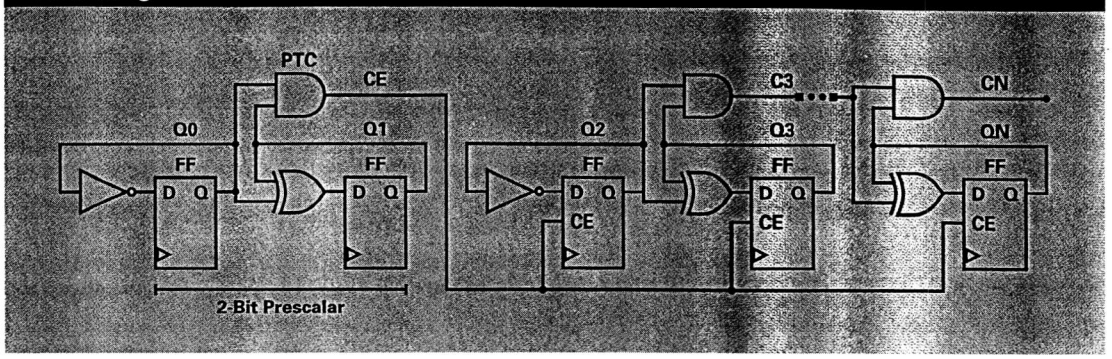


Figure 3a. Counter speeds and routability can be improved by prescaling. The N-bit counter above has a two-bit prescalar that allows the Q[2:N] stages to operate at one-fourth the clock frequency. This reduces the number of critical paths to three: Q0, Q1, and CE.

Figure 3b
Predecoded and pipelined prescaling

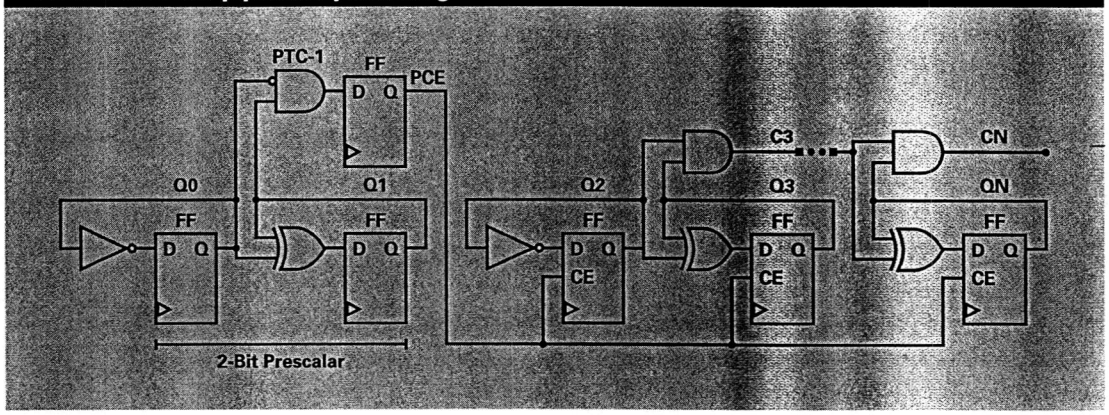


Figure 3b. Greater counter performance can be obtained with a pre-decoded and pipelined prescalar. By eliminating the intermediate combinatorial delay in the critical CE path, the entire route budget can be allocated to the PCE route.

advantage is the reduction of critical routing, since only the prescale bits and the PTC-to-CE line are in critical paths. In some cases, the PTC-to-CE line can be sped up by pre-decoding and pipelining PTC, because that critical path suffers no intermediate combinatorial delay. (Note: If the counter is loadable, the pipelined pre-decoded PTC flip-flop must also be set when loading the counter with a modulo-pre-decode value. When using the daisy-chain style carry, large counters will not likely be "load-and-go" because of the extra time required after loading to permit the daisy-chain carry to propagate to the last stage.)

Consider combinatorial clock-enables In the Xilinx 3K and 4K architectures, each CLB has two flip-flops with a common, dedicated clock-enable. For flip-flops with different CEs to share the same CLB, at least one of the two CEs must be changed to a combinatorial-style CE. This is easily done by inserting a 2-to-1 mux inline to the flip-flop's D input. The mux select is steered by the clock-enable signal, the mux 1-input is connected to the original D-input signal, and the mux 0-input is sourced by the flip-flop Q feedback (Note: Some library flip-flop macros are built using the dedicated-style, and some using the combinatorial-style, CE. You should check on this before using them.) Combinatorial CEs are more routable since a CLB has more function generator inputs than dedicated CE inputs. The disadvantage of the combinatorial CE is that it consumes two of the function generator inputs. This may force the next-state function into an intermediate level of logic that could adversely affect performance. Therefore, status and control registers with simple D-input functions are better candidates for this treatment than accumulators and other arithmetics with complex D-input functions.

Duplicate Duplicating critical pins, gates, flip-flops, and macros is another simple performance enhancing technique. When a critical input signal needs to reach several distant destinations, consider assigning multiple input pins. If a highly loaded net can be resourced from signals already globally distriuted, consider using several source gates. If the clocked version of a synchronous signal is used by several functions, then design in duplicate flip-flops that will place near their destinations. If the Q outputs or terminal count (TC) of a small counter is required in multiple or disparate areas, consider duplicating the entire counter macro. (Note: In most cases, mapping and placement constraints will be necessary to obtain the desired results.)

Customize macros Library macros are a double-edged aid. As generalized functions, they expedite design entry, but they will likely obscure mapping, placement, and naming processes. The same veil that clocks target technology also inhibits needed insight. Generalized macros are a hindrance when routing can be improved by interleaved datapath structures. For performances that could benefit from specific object placement, parameterized macros are less advantageous. Using off-the-shelf macros makes recognizing signal names difficult when generating post-route simulation vectors. Use customized macros—which can be tailored copies of the library versions—to facilitate custom mapping, placement, and naming. They cost less in compile time to expand, and have fewer deletions to verify. Customized macros are easy to create by using cut-and-paste or auto-

Route budget analysis

The goal of the route budget exercise is to determine whether a particular circuit implementation will meet desired performance criteria. If calculations yield reasonable numbers, then that implementation can be expected to place and route without additional mapping or placement constraints. Marginal numbers suggest that, from route to route, a particular implementation may or may not meet requirements. If the calculated numbers are unreasonable, an alternate implementation will have to be found before synthesis or schematic entry.

For the given implementation of a given path of concern—usually the longest path—the route budget exercise is as follows:

1. **Identify the synchronous source**
2. **Identify the synchronous destination**
3. **Determine the number of intermediate combinatorial logic levels**
4. **Subtract path delays from the target clock period**
5. **Calculate the delay-per-route**

In Sidebar figure a, for example, between the synchronous source A and the synchronous destination B, there is a single intermediate level of combinatorial logic C. Given clock period t(p), clock-to-Q delay t(c), setup t(s), and combinatorial delay t(d), the route budget for routes X and Y is given by:

t(x) + t(y) = t(p) - t(c) - t(d) - t(s)

If the target device is a Xilinx 4005-5 and the target frequency is 50MHz, then for a combinatorial delay through the F function and a setup through the G function, the route budget (using numbers given on pp. 2-53 of the 1994 Xilinx data book) is:

t(x) + t(y) = 20.0 - 3.0 - 4.5 - 4.5 = 8.0 => 4.0 ns/route

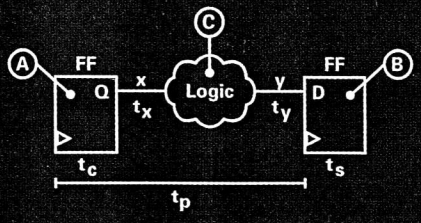
From experience, a route delay approximately equal to a single function generator delay is reasonable, and can be expected to route without alteration.

If, however, the same circuit were expanded in such a way that with optimal mapping, a second intermediate combinatorial logic level were to be introduced into the calculations (see Sidebar figure b), the route budget would yield the following less favorable numbers:

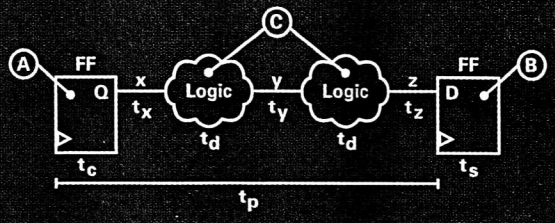
t(x) + t(y) + t(z) = t(p) - t(c) - 2t(d) - t(s)

= 20.0 - 3.0 - 2(4.5) - 4.5 = 3.5 => 1.2 ns/route

Subsequently, it should come as no surprise that without special considerations, the route results of implementation 1b will not, in all probability, meet 50MHz timing.



Sidebar figure a. Route budget analysis for a synchronous circuit with one intermediate level of combinatorial logic.



Sidebar figure b. Route budget analysis for a synchronous circuit with two intermediate levels of combinatorial logic.

Legend
- **A Synchronous source**
- **B Synchronous destination**
- **C Intermediate combinatorial logic**
- **t_p Clock period**
- **t_c Clock-to-Q delay**
- **t_s Setup**
- **t_d Combinatorial delay**
- **t_x Route delay for x**
- **t_y Route delay for y**
- **t_z Route delay for z**

array commands, and they can be copied quickly. In the long run, customized macros save time and prevent frustration.

Don't over-constrain Some final words of caution: Don't over-constrain. After following the advice on how to interfere with the automatic tools, leave the tools to do what they do well. That is, don't constrain "blobular," combinatorial, serpentine, or random logic just because it looks good. Arbitrarily doing so will probably hamper routing efforts and decrease design performance. For example, columnizing a set of 4-to-1 muxes may be good for the the (1x) output, but will be bad for the (4x) inputs.

For your tool kit Until interactive floorplanners come of age, intelligently hand-crafted pin assignments provide the greatest margins for implementation success. Conversely, arbitrary pin pre-assignment that diminishes flexibility is the best invitation to routing trouble. When pre-implementation route budget analyses uncover disappointing performance calculations, re-implemented solutions can be substituted early in the design process. Synchronous design will save time by simplifying analysis, while gated clocks waste resources and complicate calculations. Pipelines—trading resources for latency—provide significant gains in throughput. State-per-bit state machines—trading next-state function generators for flip-flops—provide significant gains in speed. Linear feedback shift registers and prescaled counters increase circuit performance by simplifying intermediate logic and reducing route congestion. Combinatorial clock-enables will make their respective flip-flops easier to place, and duplicate functions will make their signals more accessible. Customized macros will improve mapping, placing, and naming—all of which simplify routing, analysis, and simulation. Finally, remember that over-constraining may offset these gains.

Put these tips in the designer tool kit, and your FPGA designs can be minimized for design iterations and time to market, and maximized for utilization and performance. ●

Stephen L. Wasson is a principal of HighGate Design Inc., Saratoga, Calif., consultants specializing in FPGA implementations.

Please indicate the value of this article on the reader service card.

77	78	79
High	Medium	Low