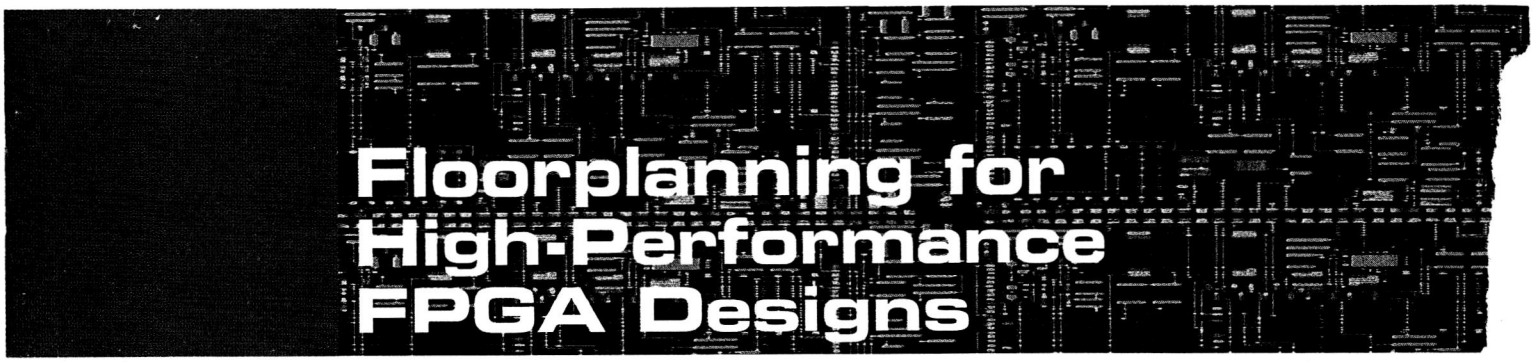
Floorplanned designs save weeks in the design schedule and typically achieve 25 to 50 percent improvements in performance.



Floorplanning for High-Performance FPGA Designs

by Stephen L. Wasson

Life is a fractal; FPGAs are not. FPGAs are two-dimensional, rigid, crystalline structures with finite route and logic resources aligned along fixed orthogonal axes. Life's most successful designs are those adapted to natural law; your most successful FPGA designs will be those adapted to target device constraints. Implementing FPGA designs irrespective of target constraints is like buying furniture before you've seen the house: a less than optimal interior design strategy. A more successful design strategy is to craft interior objects within their destination constraints. Therefore, floorplanning is as much about designing the objects within a floorplan as it is about planning the design floor.

Floorplanning is the design methodology that maps given design specifications into specific target architectures by considering device constraints to develop an optimal implementation. Optimal implementations improve design routeability and device utilization which mean higher performance at lower cost. An effective floorplanning methodology also reduces redesign risks and unnecessary iterations through the design cycle which mean greater competitive edge and less time to market. Following is an introductory treatise on the precepts of FPGA floorplanning.

In general, floorplanning is most advantageous in architectures with limited or fixed resources and is less applicable in EPLD or CPLD architectures. Moreover, it is especially well suited to software environments that provide floorplan enforcement mechanisms: mapping and placement constraint hooks. Such "human-aided-design" hooks allow designers to make the trade-offs that automated tools cannot divine. Additionally, as devices increase in size and designs increase in complexity, floorplanning is becoming ever more essential to FPGA design success.

The basic procedure The general FPGA floorplanning procedure is as follows:

I. Design to the block diagram level
II. Floorplan a best-case partitioning and placement
   A. Identify the primary data paths
   B. Identify the primary data-path objects
   C. Lay out an interconnect pattern
      1. minimize route resource consumption
      2. maximize route distribution
III. Implement the remaining hierarchy.

Specifically, design the block diagram hierarchically down to the level where blocks most nearly approximate autonomous objects (individual registers, counters, adders, RAM arrays, etc.). The most significant buses are the primary data paths, and the objects they interconnect are the primary data-path objects. Laying out the interconnect pattern between data paths and data-path objects is the step during which alternate implementation trade-offs are made to accommodate device constraints while maintaining performance requirements — the bulk of the floorplanning effort. For simple designs (Xilinx 4002s), this process takes about an hour; for more complex designs (full Xilinx 4013s), this process may require up to five days — all of which is easily regained in the design schedule by improvements in routing predictability, facilitation of performance analysis, and the subsequent reductions in design re-implementation.

Specific objectives Minimization of route resource consumption is accomplished by devising data objects that will partition in such a way as to permit the greatest number of signals to traverse the shortest distances along the fewest routing channels with the least crossovers. This most often means placing interconnected objects adjacent to each other with related elements aligned to the routing axes. Maximizing route distribution is accomplished by placing these well-partitioned objects in such a way as to evenly spread out the allocation of routing channels across the target device.

Data object partitioning and placement are primarily concerned with structured objects (objects that exhibit uniform interconnectivity patterns) such as registers and counters, and not with unstructured objects, such as decoders and state machines. An important objective in the floorplanning process is to maintain consistency of structure between related objects to ensure objects are placed with the same number of elements per logic block and sequenced in similar directions along preferred orientations. For example, in the Xilinx 3000 and 4000 FPGAs, logic blocks may contain either one or two elements, and preferred orientations are horizontal for data paths and vertical for data-path objects.

Note that the procedure outlined above proposes floorplanning on a *pre*-netlist design; that is, on an incomplete and uncompiled design. Floorplanning prior to compilation is recommended because, at this early stage, the design is malleable. More than at any other time in the design process, pre-netlist implementations are most amenable to change with least cost and greatest advantage. Waiting until the netlist is syntactically correct before considering device constraints is an invitation to considerable rework or performance degradation.

If you *have* considered device constraints in your implementation, a well-targeted netlist is an excellent candidate for the benefits of a post-netlist floorplanning tool: efficient constraints file generation. Prior to this, in the absence of a pre-netlist floorplanning CAD package (where the "A" stands for "assisted," not "automated"), I currently recommend the use of good, old-fashioned pencil and paper. Here is the abbreviated

Figure 2
An expanded circuit

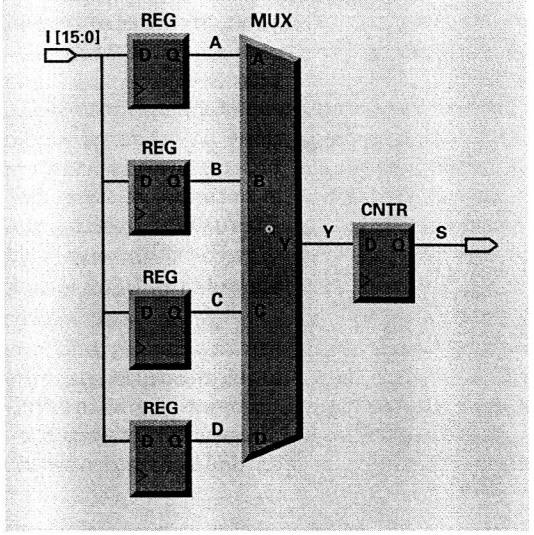


Figure 2. This is an expansion of the circuit in Figure 1A, with two more 16-bit objects. Mux becomes 4:1.

Figures 1A and 1B
A simple circuit

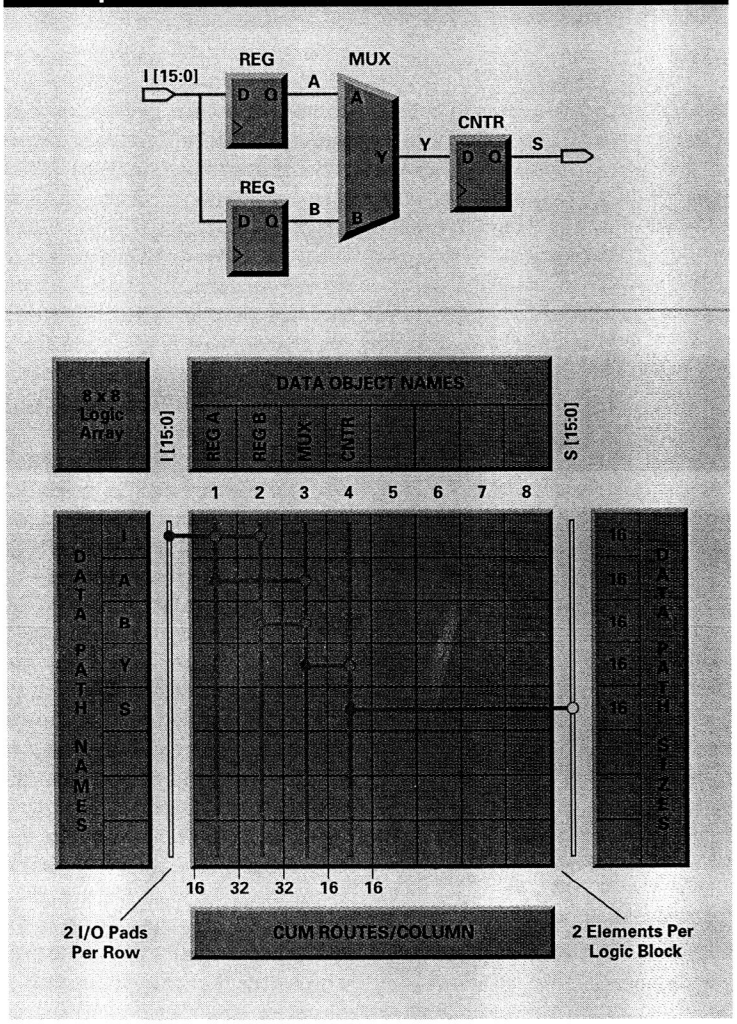


Figure 1A and 1B. This simple circuit is an 8 x 8 logic array, with four 16-bit objects and a 16-bit data path. Mux is a 2:1 combinatorial multiplexer. 1B is a floorplan for circuit 1A. Each Figure 1A data object may be partitioned 2-bits per logic block, and should be placed so that all bit-n elements are in the same resource row. Horizontal data-path lines are meant to indicate overall horizontal route requirements, not to imply the routing allocation of a specific row.

formula for the pre-netlist floorplanning recipe: transform the block diagram into suitable floorplanning representation; approximate best-case object partitioning and placement; study the interconnect distribution; and re-partition and re-place objects until an optimal interconnect pattern is obtained.

Figures 3A and 3B
Floorplanning the circuit in Figure 2

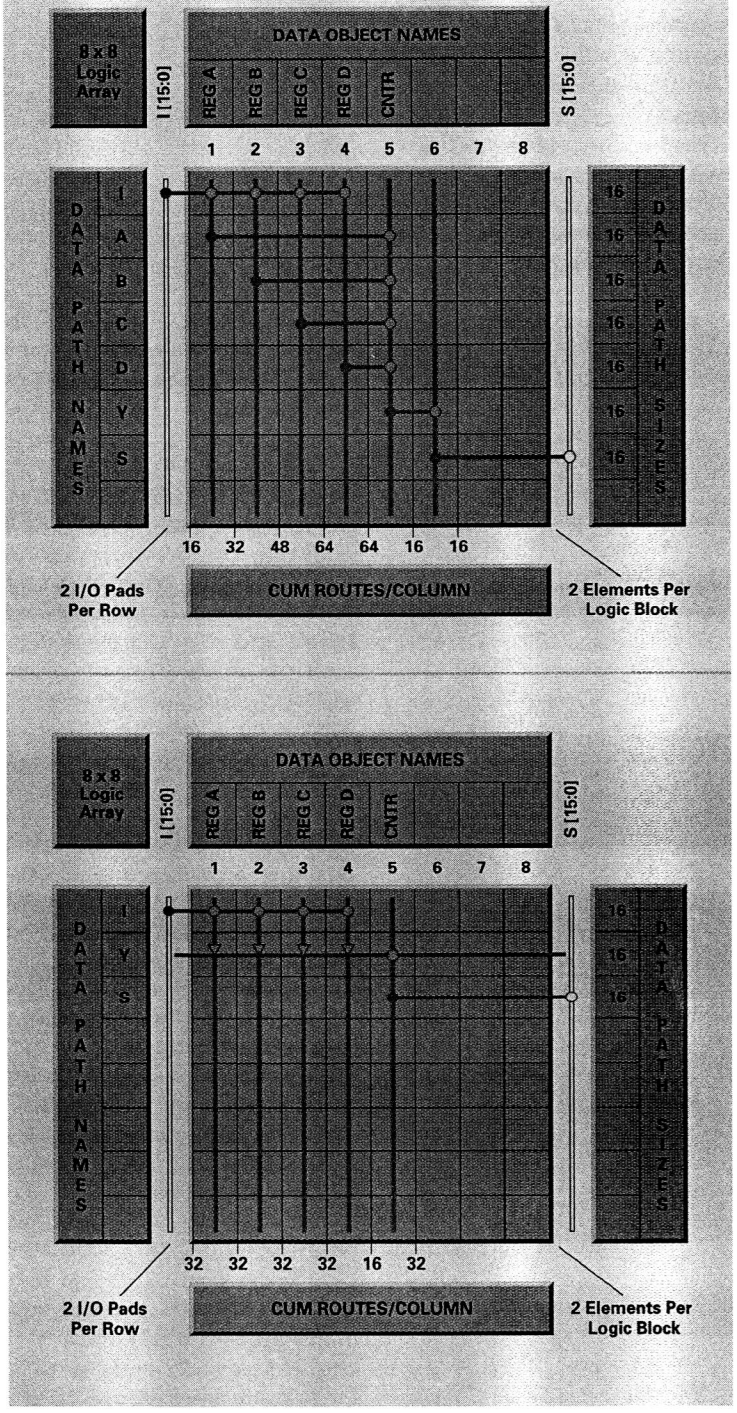


Figure 3A and 3B. The cumulative routes per column indicate that, with floorplan 3A, maximum route consumption will occur across columns three, four, and five. 3B is the improved floorplan for the circuit in Figure 2, achieved by re-implementing mux with tri-state drivers (two per row). Note the reductions in the cumulative routes per column compared to figure 3A.

The paper exercise One suitable paper floorplanning representation simply depicts the device array as a grid, the data objects as generic rectilinear symbols, and the interconnecting data paths as heavy, bus lines. The array grid connotes routing axes and delineates available logic resources. Within the grid, data object symbols should be drawn to illustrate both object placement and logic consumption. Data-path interconnect should be drawn to show route resource allocation from source(s) to destination(s). Best-case partitioning and placement are approximated by iterative rearrangement of the data object symbols and their interconnect lines. An optimal interconnect pattern is obtained after such iterative rearrangements have maximized data-path alignment to the routing axes and minimized route resource consumption (quantity and length) across all routing channels. Keep in mind that, for pre-netlist floorplanning, the data objects are still malleable and should be (re)implemented to accommodate these goals.

In the accompanying floorplan figures, data flow is horizontal and data structures are vertical. Device I/O rails are illustrated by vertical rectangles and data-path objects are drawn as solid vertical lines. Data-path sources are shown originating from solid "solder dots" and data-path destinations are shown terminating at clear solder dots. The solid horizontal bus lines are data paths. All objects, including pin assignments along the rails, have identical bit ordering (either ascending or descending) and identical bit pairing (either 1-bit or 2-bits per logic block). Mnemonic notations on the left margins are data-path names; numeric notations on the right margin are data-path sizes (in

bits). Labels along the top are object names. Numerics along the bottom indicate the cumulative number of routes per-column. The cumulative per-column route numbers are the ones to track, minimize, and distribute.

Example one For example, given that the circuit in Figure 1A consists of four 16-bit structured objects (two registers, one two-to-one combinatorial mux, and one loadable counter), Figure 1B depicts a preliminary floorplan with topology that easily mimics the circuit data-path flow. The input bus (I), assigned along the left rail, immediately connects to its two destinations (reg A and reg B), adjacently placed in the two left-most columns. The two register outputs (bus A and bus B) then horizontally traverse right two and one columns respectively to their destination (mux Y) in column three. Next, the mux output (bus Y) traverses right a single column to its destination (cntr S), which finally sources its output bus (S) to the right device rail. Note that the maximum horizontal routing demand occurs between columns one and two and columns two and three, as indicated by the per-column route numbers along the bottom.

If the circuit in Figure 1A is then expanded from two to four registers, as shown in Figure 2, and then if the registers are all placed contiguously, as shown in Figure 3A, an increase in route congestion will occur across columns two through five. Furthermore, if the target is a Xilinx 3000 or 4000 device, and if the mux is implemented as a combinatorial 4:1 mux, then object mux will actually require two resource columns of 1-bit per logic block elements. These two columns for object mux could be placed amongst the reg objects so as to reduce the per-column route allocations, but rather than explore that option, re-implementing the mux with

Figure 4
A more complex circuit

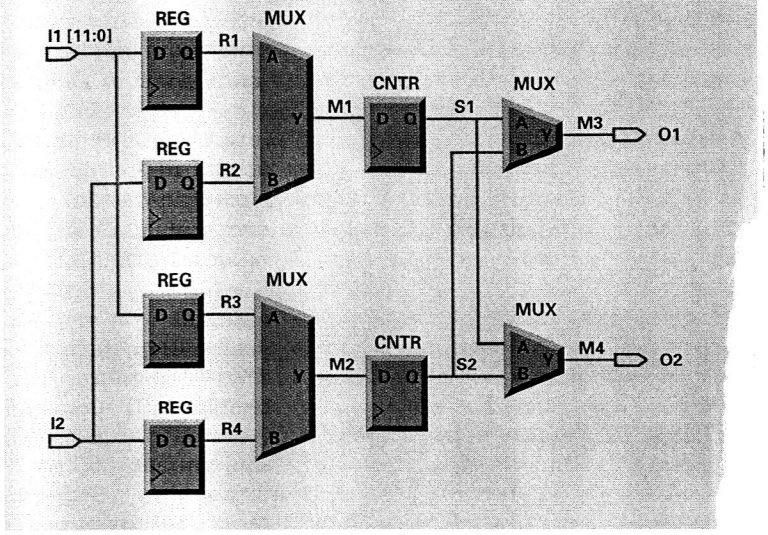


Figure 4. This is a 12-bit circuit with ten 12-bit objects.

Figure 5
Preliminary floorplan for the circuit in Figure 4

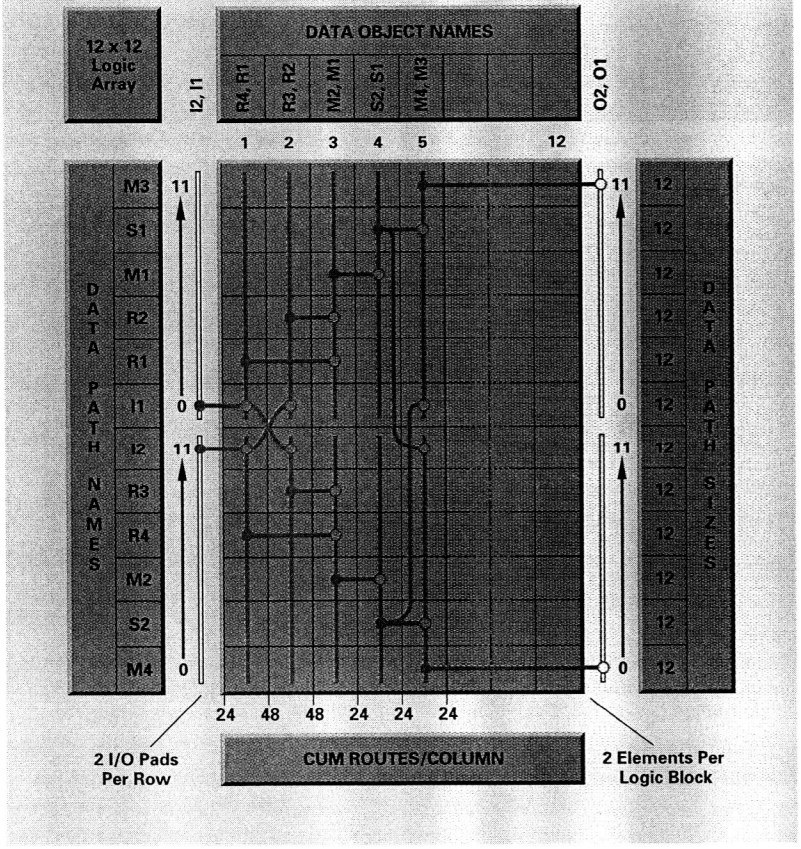


Figure 5. All objects are partitioned 2-bits per logic block, bit ascending. I/O buses I1 and O1 are sequentially assigned to the left and right upper device rails; I2 and O2 are sequentially assigned to the lower device rails. Objects R1, R2, M1, S1, and M3 are placed in the upper half of the array. Note the diagonal routing required for buses R1, R2, S1, and S2.

tri-state logic will yield better results, as shown in Figure 3B.

Example two Figure 4 shows a more involved circuit of ten 12-bit objects and the topologically similar floorplan in Figure 5. However, in this preliminary plan, input buses I1 and I2 and counter buses S1 and S2 have to make less desirable diagonal traverses. One way to "orthogonalize" this layout is to bit-interleave both the I/O pins and the data-path objects. Figure 6 depicts one such layout in which the I1 and I2 input pins are interleaved along the left rail; the two register destinations for input bus I1 are interleaved in column one; the two register destinations for bus I2 are interleaved in column two; the M1 and M2 muxes are interleaved with their respective counter destinations S1 and S2 in columns three and four; the M3 and M4 muxes are interleaved with each other in column five; and the O1 and O2 output pins are interleaved on the right rail. This second plan achieves both data path orthogonality and route resource reduction certain to increase overall performance.

But wait, there's more... These two generic examples illustrate the *rectilinear* models of floorplanning covering about 90 percent of the design cases. (The other 10 percent are better floorplanned using a *circular* model.) Example one demonstrates the time-saving advantage of floorplanning: by determining the most route-efficient structures prior to final implementation, redesign, and therefore time-to-market, are reduced. Example two demonstrates the performance-enhancing advantage of floorplanning: by crafting the most route-efficient mapping and placement prior to initial route, routeability, and therefore performance, are improved (typically more than 25 percent, often 50 percent.) Example two also makes the case for hand-crafted floorplanning since an interleaved scheme—applicable to about half the design cases—is unlikely to be generated without human-aided design assistance. ●

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

An in-depth case study on interleave techniques will be presented in the tutorial "Floorplanning Xilinx FPGA Designs for High Performance," scheduled for presentation on February 28 in the On-Chip Systems Design portion of Design SuperCon '95.

Figure 6
Improved floorplan for the circuit in Figure 4

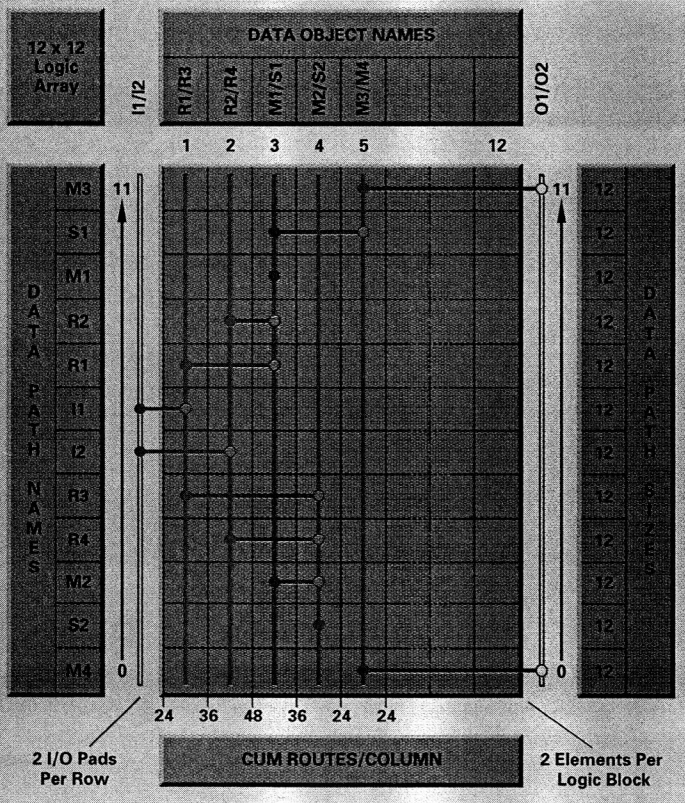


Figure 6. Objects are here partitioned 1-bit per logic block, bit ascending, and bit interleaved. That is, within a column, bits from two objects alternate resources. For example, in column one, starting from the bottom row, the ascending bit-sequence is R1 bit 0, R3 bit 0, R1 bit 1, R3 bit 1, etc. The I1, I2 and O1, O2 I/O buses are also interleaved along the full heights of the left and right device rails. Note that all diagonal routing has been eliminated with the bit-interleaved scheme.

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

109	110	111
High	Medium	Low