Architectural and Physical Design Challenges for One-Million Gate FPGAs and Beyond

Jonathan Rose+ and Dwight Hill*

Department of Electrical and Computer Engineering, University of Toronto+ and Synopsys*, Inc.

Abstract

Process technology advances tell us that the one-million gate Field-Programmable Gate Array (FPGA) will soon be here, and larger devices shortly after that. We feel that current architectures will not extend directly to this scale because: they do not handle routing delays effectively; they require excessive compile/place/route times; and because they do not exploit new opportunities are presented by the increase in available transistors and wiring. In this paper we describe several challenges that will need to be solved for these large-scale FPGAs to realize their full potential.

1. Introduction

As 0.25μm CMOS processes arrive, we anticipate that FPGAs will have gate capacities in excess of 250K gates, [1] and once process feature sizes progress below the 0.13μm mark (perhaps by the year 2001) the one-million gate FPGA will become a technical feasibility. Having the ability to make such a chip, however, is different from having the architectural and algorithmic knowledge to exploit this potential to its fullest.

Although one-million gate Mask-Programmed Gate Arrays (MPGAs) are available today, the physical design of these devices is difficult due to their sheer size and the fact that the delay of the routing is a much larger portion of the critical path delay than has been the case in the past. A one-million gate FPGA will have these problems and more, due to the programmable nature of the interconnect.

As the quantity of silicon available for FPGAs grows, it also makes sense to question if the devices should look the same or if some radical change in architecture is beneficial.

In the following sections we outline challenges that we foresee in the creation of the architecture of a one-million gate FPGA, and in the placement and routing of these large devices. We also suggest possible directions for architecture and challenges involved in those directions.

2. Core Routing Architectures for Speed and Density

As process technologies advance, gate switching speeds improve because of short channel effects, but wiring delays due to the capacitance and resistance of routing wires tend to become proportionally greater. Even if the device sizes were not shrinking, the huge increase in the number of gates per chip make the relative distances that a signal must travel much greater. This is true in mask programmed parts, and these factors are often sited as part of the "deep sub micron" problem. It is equally true in FPGA's. But in FPGA's, the intrinsic delays of the metal routing wires are joined with the delays of the programming switches in programmable interconnect. The combined effect will cause routing delay to become more significant than it is today.

The FPGA routing architecture is the manner in which metal wires and programmable switches are placed to provide connectivity between the pins of the FPGA logic blocks. The routing architecture must be designed to provide paths with greatest possible speed without sacrificing too much area. Three key architectural parameters that have strong effect on delay and density are buffering, segmentation distribution and segmentation population. We discuss each of these in the next sections.

2.1. Buffering

One key issue in creating routing architectures for low delay is to prevent the occurrence of nets in which one driver must charge a large capacitance through a large resistance. This can easily occur in an FPGA when a single output buffer drives a load through many programmable switches and/or drives many loads. One way to reduce the delay of this network is to divide networks with large RC time constants into pieces and drive them separately with active buffers.

There are two extremes that describe this spectrum of buffer usage. In one extreme every programmable switch could have a buffer in series with it (with perhaps a programmable direction). This is the case in the Altera Flex 8K and 10K architectures [3][4]. Although this prevents large RC constants from "building up" through multiple programmable switches, it makes every programmable switch pay the price of a relatively high constant delay. The alternative, in which some connections have no buffers, would make some short connections much faster. This could have a significant positive effect on the critical path.

The other extreme is to have no buffers at any programmable

[1] Here we assume that one 4-input lookup table plus one D flip-flop are equivalent to 12 "gates."

FPGA97, Monterey California USA
© 1997 ACM 0-89791-801-0/97/02 ..$3.50

switch. This is the case in the Xilinx 4000 architecture [5]. While local short connections will be faster than with the buffered case, it is also easy to create large slow RC networks in the absence of buffers. It is clear that some mixture of the buffered and non-buffered switch is most appropriate. The more recent 4000EX architecture provides a set of buffers in the routing [6].

2.2. Segmentation Distribution

A second technique to prevent buildup of large series resistance is to employ wires of differing lengths in what is known as a *segmented* routing architecture[1] [2]. Figure 1 illustrates segments of three different lengths. The segmentation distribution, which is the number of tracks in a channel with segments of each length, is a key question that has been studied to some degree [7][8][9][10]. If there are too many long segments, then the wire will be wasted when used for short connections. If there are insufficient long segments then networks will be too slow with many series programmable switches.

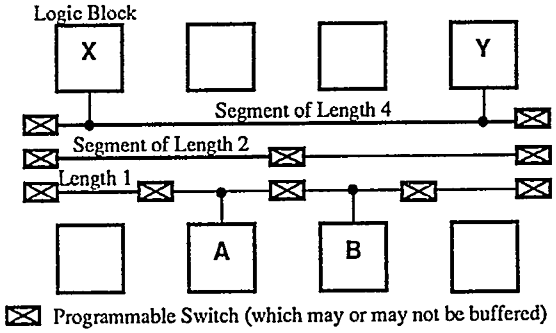

Figure 1 - Routing Segmentation

An associated issue in segmentation design occurs at the point that orthogonal channels intersect: should long segments connect to other long segments, or other short segments or both?

2.3. Internal Population of Segments

An important issue in segmentation design is whether or not there is access to the internal portion of the segment, referred to as the *internal population* of a segment. Figure 2 illustrates the two extremes of internal population with two different segments of length 4. The top segment is fully internally populated - the segment can connect to both the logic block it passes by and the orthogonal channels.The bottom segment is completely un-populated, and so can only be connected to at its ends.

The population of a segment is important with respect to both speed and density: the more programmable switches connected to a wire, the greater the parasitic capacitance (and hence delay) they add to the wire. From a density perspective, more switches provide greater flexibility and perhaps fewer routing tracks will be needed overall. However, a completely populated long segment requires vastly more area than a unpopulated one. As processes with more metal layers become common, it will become more desirable to de-populate segments because the wires themselves will be cheaper.

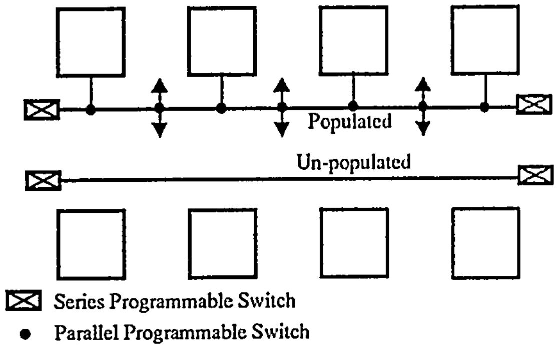

Figure 2 - Internal Population of a Segment

2.4. Putting These Together - The Challenge

These three design spaces (buffering, segmentation distribution and internal population) interact with each other in very complicated ways. For example, a buffer may be needed to simply drive the capacitance of a long segment, depending on how long the segment is. It may also be unnecessary to use a buffer because a long segment reduces series resistance by removing series programmable switches.

The determination of appropriate choices in these architecture spaces are affected by the "goodness" measures of a routing architecture: its delay, density, predictability, "ease" of routing, power, and noise properties. These must typically be measured by experimentation [1] or by fast back-of-the envelope estimates.

We believe a key challenge for the one-million gate FPGA and beyond is to make the correct choices in this space of architectures.

In addition, these choices cannot be made in isolation from the placement routing algorithms that are used to map circuits into the architecture. A routing architecture is only half the solution without a routing algorithm. Does an iterative timing-driven maze router do an adequate job of both routing completion and timing? Is there some more clever interaction between the routing architecture and algorithm that can produce significantly faster and more dense FPGAs?

3. Compile Time

The largest FPGAs available today have 5000 LUT/flip-flop pairs, and the next generation will contain in excess of 12,000. The 1 million-gate FPGA will contain (by today's gate counting standards) approximately 83,000 LUT-flip-flop pairs. There is evidence to suggest that the complexity of most placement and routing algorithms is non-linear. With these large sized FPGAs, the placement and routing time could easily exceed 1 day, extrapolating from current tools.

Fast turnaround devices like FPGAs lose much of their value if users have to wait more than a day for the synthesis, place and route tools to finish one design. In fact, many customers would like to have all that happen in the same time as it takes to compile and link a typical C-language program.

This issue can and should be dealt with both in the architecture and CAD tools. For the architecture, one question is how much silicon can we afford to give up to make the compile time quicker? It is clear that by adding more routing resources (typically the number of tracks per channel) the routing problem becomes easier, and therefore quicker, because the place and route don't have to fight so hard to achieve routability.

On the CAD side, the challenge is to find placement and routing algorithms that give up the least quality for the most gain in time.

Only through synergy of the architecture, placement and routing systems can significant gains in compile time be achieved. One approach to this type of synergy is discussed in the next section.

4. Partitioned Architectures

It is extremely unlikely that the million gate FPGA's will be structured as a flat, homogeneous sea of logic blocks and switch boxes. Rather, they will almost certainly have large scale structure. This will add flexibility to their use, but challenge to the architects and CAD developers.

4.1. Higher-Functionality Blocks

Based on industry experience with standard ASICs and embedded arrays, we believe that typical 1Mgate FPGA will almost certainly contain reconfigurable RAM blocks [4] [15]. The design and use of these will be a major challenge because of the performance/flexibility trade-offs involved.

Going beyond RAM, as FPGAs are used for more computation-oriented applications, such as digital signal processing and FPGA-based compute engines, it is tempting to suggest the use of adders, multipliers, dividers as the basic logic block. Are these practical? They give up generality/flexibility to get more speed and density. Can synthesis be made to handle them? We believe the challenge is to architect blocks like these so that they retain some flexibility, so that they can be used in the widest spectrum of applications possible.

4.2. Partitioned FPGAs for Speeding up Compilation

As mentioned above, many designers would like their FPGA compilations to be as fast as their C compilations. In order to make the software processing faster, FPGA compiling will probably have to look more like software compiling. For example, a large C program is usually organized as many smaller files, and it is generally only necessary to recompile only the files that have changed. A fast linking process stitches them all together. To carry this into FPGA's, one approach would be to assume that the design is organized as a set of HDL files, stitched together at the block level with a system level netlist. The content of individual HDL files may change frequently, but the overall system organization should be relatively stable. If the FPGA architecture can support this type of hierarchy efficiently, it will be easier for designers to work incrementally.

In ASICs, large chips are typically floorplanned, where the floorplan matches the highest level netlist structure. One approach to a million gate FPGA would be to use a partitioned FPGA architecture to support the same methodology. The essence of a partitioned one million gate FPGA would be to break the surface of the die into about a dozen to two dozen large sections, each capable of supporting about 100K gates. If there were fewer sections than this, the processing time for each would be too high. If there were many more than this, they would be too small to take a significant block of logic. Each physical block would be matched to one or more logical blocks. Naturally, if the die is heterogeneous, there may be some restrictions (e.g. RAM blocks from the logical hierarchy would have to map onto RAM blocks on the die). To simplify the compilation, it would be best not to force a high utilization: the use of an extra 50% to 100% of silicon over what might otherwise be needed would greatly simplify processing time.

Software linking is fast, in part, because there is essentially infinite bandwidth among the program modules. In order to make FPGA compiling fast there would have to be many high speed connections among the sections of a 1Mgate chip, perhaps on the order of 5K to 20K lines. Architecting these will be a major effort: they could easily dominate the area and power of the die. Issues here include the number of "local" interblock wires (e.g. wires that connect only adjacent blocks, or blocks within a smaller part of the die), and the number of global interblock wires. This is essentially the same problem discussed in Section 2, just viewed at a die level instead of at a CLB level. If this is done well, the user can ignore the intermodule connections problem. If done poorly or without sufficient bandwidth, partitioning will continue to be the nightmare it is today.

On such a partitioned chip the CAD sequence might be:

- Map logic blocks onto physical regions. Often several smaller blocks will go into one physical region. This process will probably be interactive.
- If there is a large logical block that will not fit onto a single region, it must be partitioned.
- Place the cells within each region, and route each region. This process could be performed on multiple workstations, as is done today with emulator machines.
- Route among regions using the cleverly designed global routing resources mentioned above.
- If the designer changes a source file, then the CAD tool

will have to re-synthesize it, place and re-route the regions affected.

The synthesis process itself would be facilitated by keeping the technology rules simple, with few special cases. Mapping to LUTs, for example, is a relatively well understood problem. Mapping to complex micro-grained architectures, where the speed of each type of gate is a function of its detailed layout is more difficult. But as synthesis technology evolves, the dividing line shifts, and the problems that appear difficult today may prove less difficult tomorrow.

Placement within a section would be facilitated by keeping the relationship between distance and speed easy to model (a quadratic relationship, for example). As discussed in Section 2, this in itself may be a significant challenge.

Routing within a section would be simplified by keeping the system regular, supporting bus routing by enabling parallel signals to turn corners and retain their parallel organization (as was done in ORCA [14]) and in general, as discussed in Section 3, providing more than sufficient routing resources.

5. Re-Configuration Time

A convincing case can be made for using an FPGA programmed in several different ways for during one computation. A key bottleneck for making that a practical reality for one-million gate FPGAs is to make the configuration time fast enough. As the number of gates increases, the number of programming bits will increase quickly, potentially increasing the configuration time. Effective partial reconfiguration strategies, and/or methods of increasing the configuration speed are needed.

6. Yield Enhancement Through Redundancy

To make very large FPGAs, it seems clear that enhancing silicon yield through redundancy should provide significant cost savings. Altera has demonstrated a workable method with their long segment architectures based on column redundancy [11]. An open challenge for island-style architectures[1][5], with short-length segments is to determine if there is a practical way to do a similar kind of redundancy. Although there has been some work on this subject [12][13], none of has yet to be proven practical because of routing issues. Specifically, short routing segments required significant overhead in area to be made redundant.

7. Conclusions

In this paper we have presented several challenges that need to be met for the one-million gate FPGA to become a usable and practical reality. The greatest challenge, however, comes from the fact that all of these issues must be dealt with and traded off within one device. The FPGA design space is not well understood; very few people have even been exposed to all of the issues and engineering trade-offs that are possible across the spectrum of IC processes, circuit design, architecture and CAD. But in order to succeed, all of these factors must be balanced with each other effectively.

References

[1] S. Brown, R. Francis, J. Rose and Z. Vranesic, **Field-Programmable Gate Arrays**, Kluwer, May 1992.

[2] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented Channel Routing," Proc. 27th Design Automation Conference, pp. 567-572, June 1990.

[3] Altera 1995 Data Book, Altera Corporation.

[4] Altera Corporation, Data sheet, *Flex 10K Embedded Programmable Logic Family*, July 1995.

[5] H. Hsieh, W. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey and R. Kanazawa, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays" Proc. 1990 CICC, May 1990, pp. 31.2.1 - 31.2.7.

[6] S. Trimberger, K. Duong, B. Conn, "Architecture Issues and Solutions for a High-Capacity FPGA," FPGA '97, ACM Int'l Symposium on FPGAs, February 1997.

[7] K. Zhu, D.F. Wong, "On Channel Segmentation Design for Row-Based FPGAs," Proc. ICCAD 1992, November 1992, pp. 26-29.

[8] K. Roy and M. Mehendale, "Optimization of Channel Segmentation for Channelled Architecture FPGAs," CICC '92, May 1992, pp. 4.4.1-4.4.4.

[9] M. Khellah, S. Brown, and Z. Vranesic, "Minimizing Interconnection Delays in Array-based FPGAs," Proc. CICC '94, May 1994, pp. 181-184.

[10] M. Pedram, B. Nobandegani, B. Preas, "Design and Analysis of Segmented Routing Channels for Row-Based FPGAs," IEEE Trans. CAD, Vol. 13, No. 12, December 1994, pp. 1470-1479.

[11] Altera press release, "Altera Unveils patented redundancy technology in high-density programmable logic," August 16, 1996.

[12] N.J. Howard, A.M.Tyrell, N.M. Allinson, "The Yield Enhancement of FPGAs," IEEE Trans. VLSI Systems, Vol. 2, No. 1, March 1994, pp. 115-123.

[13] G.H. Chapman, B. Bufort, "Laser Correcting Defects to Create Transparent Routing for Large Area FPGAs", FPGA '97, ACM Int'l Symposium on FPGAs, February 1997.

[14] D. Hill, B. Britton, W. Oswald, N Woo, S. Singh, CT Chen, B. Krambeck, "ORCA: A New Architecture for High-Performance FPGAs", in Field Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping, Lecture Notes in Computer Science #705, H Grunbacer and Hartenstein editors, Springer-Verlag, 1992.

[15] S.J.E. Wilton, J. Rose, Z.G. Vranesic "Memory/Logic Interconnect Flexibility in FPGAs with Large Embedded Memory Arrays," IEEE Custom Integrated Circuits Conference , May 1996, pp. 144-147.