# Effect of Logic Cell Granularity on FPGA Performance and Density

Jack L. Kouloheris[1]

IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abbas El Gamal

Dept. of Electrical Engineering
Stanford University
Stanford, CA 94305

[1]This work was performed while the author was at Stanford University and is partially supported by DARPA grant J-FBI-89-101

**ABSTRACT:**

FPGAs today are 10-20 times less dense than mask programmable gate arrays and 3-5 times slower due to the large size and high parasitics of the programmable switches used and the additional circuitry needed for programming. The choice of the logic cell and the routing architecture can, however, significantly impact the density and performance of an FPGA. The paper explores the effect of the choice of the logic cell granularity on the density and performance of an FPGA. A logic cell architecture evaluation workbench is described. Using the workbench and a set of representative benchmark designs the performance and density of an FPGA are evaluated. Two types of logic cells are considered, look up tables and PLAs. It is found that the best cell granularity is an increasing function of the programmable switch time constant, $\tau_{sw}$ When $\tau_{sw}$ is small, a small cell granularity, 4 to 5 inputs and 1 output, yields the smallest total delay. On the other hand, when $\tau_{sw}$ is large, a larger cell with 8 to 10 inputs and 3 to 4 outputs gives the best performance. It is found that multi-output cells do not significantly improve FPGA performance as they do not reduce the number of logic levels required. A probabilistic model for delay in combinational networks is described and compared to the empirical findings.

The effects on FPGA density of varying cell granularity are then investigated using the workbench. It is found that among all multi-output lookup-tables a 4-input, 1-output cell yields the highest density for a wide range of programmable switch sizes. But since such small granularity yields poor performance when $\tau_{sw}$ is large and large granularity lookup tables are too area consumptive, PLAs are investigated. It is found that PLA-based FPGAs achieve their maximum density using an 8 to 10 input, 3 to 4 output PLA, a granularity that coincides with the one yielding the best performance when $\tau_{sw}$ is large.

## 1. Introduction

A Field Programmable Gate Array (FPGA) as depicted in Figure 1 consists of an array of *logic cells* which may be programmably interconnected using a *programmable interconnection network*. The interconnection network as well as the logic cells are programmed via electrically programmable switches.

Commercial FPGAs in use today differ in the type of logic cell used, in the architecture of the interconnection network and in the type of programming technology used in implementing the programmable switches (refer to [1] for a survey of existing FPGAs). The logic cells used can be as simple as a transistor pair[2] with the most popular types being lookup-table based blocks[3, 4, 5, 6], multiplexer based blocks[7, 8] and PLA based blocks[9, 10]. The architectures of the interconnection networks used fall into three basic categories: nearest neighbor mesh, segmented channels, and crossbar switches. The programming technologies used are SRAM, anti-fuse, and EPROM. As shown in Table 1, the programmable switches implemented in these technologies differ widely in their size and electrical characteristics. In all cases, however, a programmable switch occupies larger area and exhibits much larger parasitic resistance and capacitance than a contact or via. Additionally programming the switches requires extra circuitry on an FPGA. As a result, existing FPGAs implemented in the same generation of technology are 10-20 times less dense than mask programmable gate arrays and 2-5 times slower.

The adverse effects of the large size and high parasitics of programmable switches may be reduced, however, by careful architectural choices. By carefully choosing the type and granularity of the logic cell and by designing the routing architecture to achieve high degree of routability while minimizing the number of switches used, FPGA density and performance can be optimized.

In this paper we focus on the problem of selecting the logic cell granularity to optimize the density and/or the performance of an FPGA. If the logic cell is too small, implementing a design would require too many levels with too many cells and too many wires to connect them, thus resulting in poor density and performance. If the logic cell is large, fewer levels with fewer cells would be needed to implement a design, fewer wires would be needed to connect them, and there would be less total routing delay. On the other hand, if the cell is too large it would have a larger internal delay and possibly be less fully utilized than a smaller one, thus again resulting in poor density and performance. The granularity of the cell must therefore be carefully selected to optimize density and/or performance.

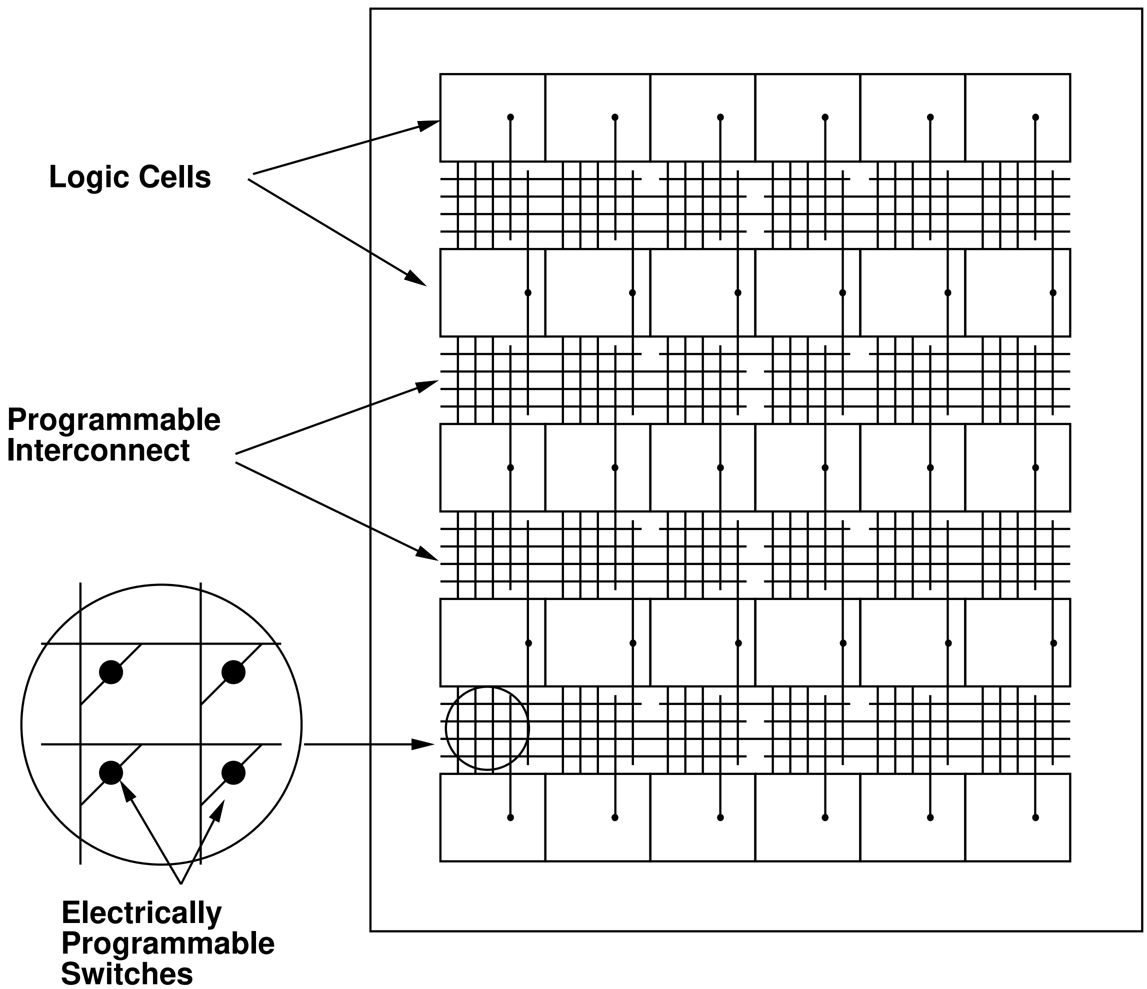


Figure 1: Generic FPGA Structure

| **Programming Technology** | **Area** ($\mu m^2$) | $\boldsymbol{R_{sw}}$ (Ohms) | $\boldsymbol{C_{sw}}$ (ff) | $\boldsymbol{\tau_{sw}}$ (ps.) |
|---|---|---|---|---|
| Static RAM Cell and Pass Transistor | 175 | 1-2 K | 10-20 | 10–40 |
| ONO Antifuse | 2.6 | 300–500 | 3–8 | .9–4 |
| Amorphous Si Antifuse | 2.6 | 50–80 | 1–2 | .05–.16 |
| Floating Gate MOSFET (EPROM or EEPROM) | 32 | 2–4 K | 10–20 | 20–80 |
| Metal Via | 2.6 | <.5 | <.7 | <.35 |

Table 1: Characteristics of Programmable Switches (as compared with metal vias) in a 0.8$\mu m$ process. $R_{sw}$ is the switch equivalent "on" resistance, $C_{sw}$ is the switch loading capacitance, and $\tau_{sw}$ is the switch time constant $R_{sw}C_{sw}$.

We use an experimental approach to explore the tradeoffs involved in the selection of the logic cell granularity. We investigate two logic cell types: the lookup table and the PLA. For these two cell types the cell granularity may be easily defined in terms of the number of the cell's inputs, outputs and, in the case of a PLA, product terms. We empirically determine what cell granularity optimizes FPGA performance and FPGA density and how this choice of granularity is related to various technology parameters such as the programmable switch area and time constant.

Rose *et al.* [11, 12] performed the first experimental work examining the effect of cell granularity on FPGA density. This was followed by our earlier work examining the effect of cell granularity on FPGA performance[13]. Singh[14] has also looked at optimizing FPGA performance, but his experiments use a simple, lumped model for interconnect delay. We extended Rose's earlier work to multi-output lookup table and PLA cells in [15, 16].

This paper is an expanded and extended version of our earlier work.

In sections 3 and 4 we investigate the effect of varying the cell granularity on FPGA performance. We find that the best cell granularity is an increasing function of the programmable switch time constant, $\tau_{sw}$ (see Figure 12). When $\tau_{sw}$ is small, as is the case for antifuse-based FPGAs, a small cell granularity, 4 to 5 inputs and 1 output, yields the lowest total delay. On the other hand, when $\tau_{sw}$ is large, as for an SRAM based FPGA, a larger cell with 8 to 10 inputs and 3 to 4 outputs gives the best performance. We find that multi-output cells help to reduce the total number of cells used to implement a design, but do not significantly improve FPGA performance as they do not reduce the number of logic levels required. Later sections detail these findings and others.

In sections 6–8 we investigate the effect of varying cell granularity on FPGA density. We find that among all multi-output lookup-tables a 4-input, 1-output cell yields the highest density for a wide range of programmable switch sizes (this is consistent with Rose's earlier results[11] on single output lookup tables). However, such small granularity yields poor performance when $\tau_{sw}$ is large as demonstrated in section 3. Since large granularity lookup tables are too area consumptive, we therefore investigate PLA cells. We find that PLA-based FPGAs achieve their maximum density using an 8 to 10 input, 3 to 4 output PLA, a granularity that coincides with the one for maximum performance when $\tau_{sw}$ is large.

The paper is organized as follows. Section 2 describes the FPGA architecture workbench developed for performing this study along with an associated experimental methodology for its use. Section 3 describes the use of the workbench to explore the tradeoff between the cell granularity of a lookup-table based FPGA and its performance. We show how the optimum cell granularity is related to the delay of the programmable switch technology. In section 4 we extend these experiments to FPGAs based on PLA cells. In section 5 we present a probabilistic model for delay in combinational networks comprised of K-input logic cells. Proofs of the propositions in section 5 are relegated to an Appendix. Sections 6–8 describe the results on the effect of cell granularity on FPGA density for the same two cell types. Key results are summarized at the end of each subsection of sections 3–8. In section 9 we discuss the implications of our findings for commercial FPGAs, the limitations of our study, and directions for future work.

## 2. Experimental Approach

Comparison of commercial FPGA architectures is problematic as they each use different cell and routing architectures, programming technologies, and CAD systems. The complex interdependence of these components and their interaction with the logic designs being implemented make it difficult to determine the relative utility of the various architectural features they use. A fair comparison of different architectural features would compare them using the same fabrication technology, the same programming device, the same CAD tools, and the same benchmarks.

An analogous problem occurs when comparing computer architectures: the rich interplay between architectural features, compilers, operating systems, and implementation technology make it difficult to determine the value of various architectural features and how they interact with one another by comparing commercial designs. To help address this problem, computer architects have developed software tools to let them vary one architectural parameter (e.g. number of registers, cache line size, associativity, etc.) at a time while holding other parameters (compiler, instruction set, etc.) constant. This allows the architect to determine the effect of a single parameter and its influence and interaction with the others and with the benchmarks. One such set of tools is the Computer Architect's Workbench, developed by Mitchell and Flynn[17] and later extended by Bray *et. al*[18]. We have developed an analogous workbench for exploring FPGA architecture.

In section 2.1 we describe the FPGA Architecture Workbench. In Section 2.2 we describe the FPGA model assumed by the workbench and in Section 2.3 we outline the experimental design and data reduction methods used.

### 2.1. FPGA Architecture Workbench

The FPGA Cell Architecture workbench is depicted in Figure 2. It accepts designs in a variety of formats including high-level languages (such as Hardware-C), logic or Boolean level descriptions (such as Boolean equations or truth tables), or gate-level netlists from real FPGA designs. All of these formats are converted to a common format, the MIS `eqn` format. High-level designs are first synthesized to logic-level descriptions via the Hercules[19] program. Technology-specific, gate-level netlists are converted to their boolean equivalents by technology "unmapping" using the Ceres[20] program. The Janus[21] and MIS-II[22, 23] programs are used to convert various boolean descriptions such as PLA files, BLIF[1], and SLIF[2] files to the common `eqn` format.

Once the designs are converted to a common format, they may be mapped onto the candidate FPGA architectures via a variety of technology mapping tools. These tools include Chortle[24] and Chortle-CRF[25], MIS-PGA[26, 27], Ceres[20], Hydra[28], and DDmap[29]. The tools take as input a description of the target cell architecture and the boolean equation file and produce as output a network of logic cells. Different technology mapping tools are provided so that their results can be compared and the effect of the CAD tools on the experiments can be determined

[1] Berkeley Logic Interchange Format
[2] Stanford Logic Interchange Format

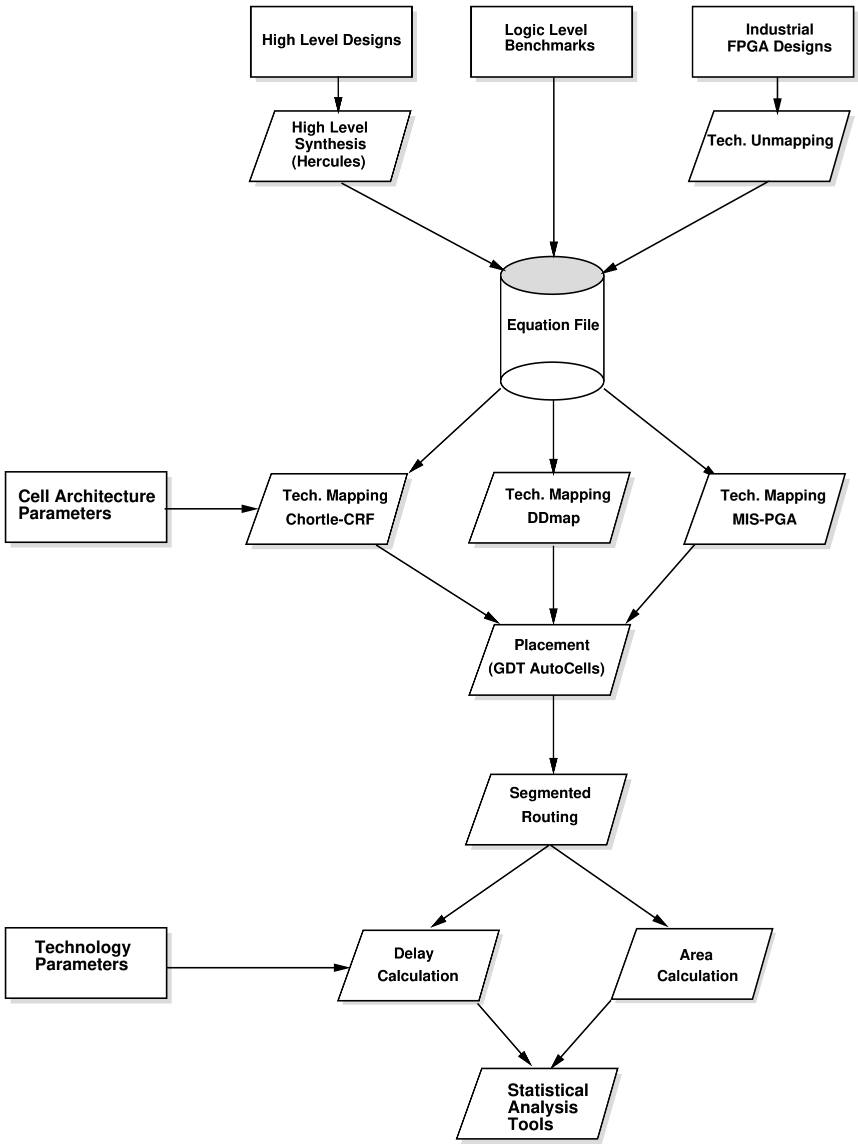


Figure 2: FPGA Cell Architecture Workbench

(see section 9.2).

The netlist produced by the technology mapping tools is fed to a placement and routing program. The placement program used for the experiments described in this paper was GDT AutoCells. GDT AutoCells creates a cell placement in two phases. In the first phase it repeatedly partitions the design in both the horizontal and vertical directions, placing cells on either side of the partition so as to minimize the number of wires crossing the partition. The second phase is an iterative improvement phase using the well-known simulated annealing algorithm[30]. After placement, a simple global routing algorithm is used to assign the nets to channels. The channels are then routed in a straightforward manner using the left-edge[31] algorithm.

The placed and routed design, wiring statistics, and a set of technology parameters (such as wiring pitch, programming element size, programming element resistance and capacitance, etc.) are used as inputs to area and delay calculation programs to determine the total area and delay of the design in the chosen FPGA architecture. Area is calculated by summing the routing and cell area of the placed and routed design. Delays are calculated by tracing the actual cell and interconnect delays for every path in the design. Details on how these delays are calculated are presented in section 3.1.

Finally, the data are fed to a variety of statistical analysis programs including SAS[32] and Minitab[33] so that the relative utility of the architectural features being studied may be compared.

The FPGA Architecture Workbench has been validated by comparing its output with the Actel Action Logic System (ALS) for selected benchmarks. This was done by mapping the benchmarks to the Actel library, then performing placement, routing, and delay calculation using both the Workbench and ALS. When the two systems were given identical architectural and technology parameters, they produced results which were in close agreement.

## 2.2. FPGA Model

The FPGA model we use is the generic FPGA depicted in Figure 1 and is comprised of rows of identical K-input, M-output logic cells separated by segmented routing channels. The cells we consider are the K-input, M-output lookup table and the K-input, M-output, N-product term PLA. The logic cell also includes a D flip-flop for each output, which may be optionally configured for implementing sequential logic (see Figure 3). These two cell types are used because they form systematic,

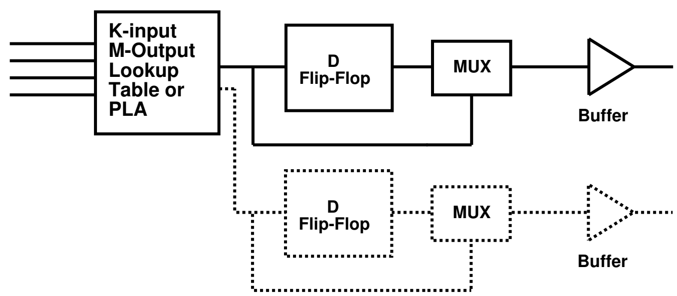


Figure 3: Logic Cell Model

parameterized families of cells which are easy to map to. Furthermore, they are closely related to several of the real FPGA cells. Segmented channel routing is used because it is a highly efficient technique for routing general purpose logic[34] and is used in several commercial FPGAs. Models for delay and area calculation for each type of block considered will be given in the appropriate sections.

### 2.3. Experimental Design

Sections 3–8 describe the results of experiments performed using the FPGA Cell Architecture Workbench to determine the effect of cell granularity on FPGA performance and area. The experimental design for these studies is a full factorial design; i.e., we run an experiment for every possible combination of all levels of each experimental factor.

The primary factors, or independent variables, used in our experiments are:

- $\boldsymbol{K}$ – The number of basic cell inputs.
- $\boldsymbol{M}$ – The number of basic cell outputs.
- $\boldsymbol{\tau_s}$ – The RC time constant for the programmable switch element.
- $\boldsymbol{A_{bit}}$ – The bit area for a lookup table cell.
- $\boldsymbol{A_{fixed}}$ – The fixed overhead area for a basic cell.
- $\boldsymbol{A_{pe}}$ – The programming element area for a PLA cell.
- $\boldsymbol{R_p}$ – The routing pitch for the programmable interconnect.

These experimental variables are defined more precisely in the subsequent sections on area and delay tradeoffs.

The response (or dependent) variables for our experiments are:

- $\boldsymbol{D_{crit}}$ – The FPGA critical path delay.
- $\boldsymbol{A_{total}}$ – The total FPGA area.

In addition to the primary factors described above we also examine a number of secondary factors. These include the benchmark type (dataflow or control), the benchmark size, and the technology mapping program.

The advantage of a full factorial experimental design is that every combination of benchmark and experimental factor is examined. We can find the effect, if any, of every factor, including secondary factors and their interactions. The primary disadvantage is the large number of experiments that must be run. For example, in section 3.3 we examine the effect of $K$, $M$, and $\tau_s$ on critical path delay for 10 values of K, 4 values of M and 14 values of $\tau_s$ for each of 30 benchmarks. This requires running 16,800 separate experiments in a full factorial design. Since this is a new experimental area, we believed that the advantages provided by a full factorial design outweighed the disadvantages of running a large number of experiments.

**2.3.1. Benchmarks.** Since the goal of our investigation is to improve the performance and density of FPGAs for general applications, the 30 benchmarks used in the experiments (Table 2) were selected to cover a wide variety of design types (data flow, control and mixed), design styles (schematic entry, high level language design and synthesis), and sources (academic and industrial benchmark sets and real FPGA designs).

**2.3.2. Data Reduction.** The large volume of data created by our experiments necessitates some method of reduction to render it amenable to analysis. Since the benchmarks we use are all of different sizes, we need some method of normalizing them. The baseline cell architecture we use for normalization and comparison is the 2-input, 1-output lookup-table cell. The critical path delay and total area for each alternative implementation (i.e., value pair for K and M) of a benchmark is divided by the corresponding values for the 2-input, 1-output implementation. This allows us to combine the data from the various benchmarks and show the relative change in area and/or delay over the baseline implementation.

| Benchmark Name | 2-input Cells | Source | Type | Description |
|---|---|---|---|---|
| C499 | 392 | ISCAS | Dataflow | Error Checking and Correcting |
| apex6 | 671 | MCNC | ? | misc logic |
| decbig | 3247 | MIS | Dataflow | misc logic |
| rot | 591 | MCNC | ? | misc logic |
| des | 3218 | MCNC | Dataflow | Data Encryption Standard |
| seq | 3338 | MCNC | Control | Cache Control Logic |
| wmm | 586 | FPGA | Mixed | Morse code receiver/transmitter |
| alu | 384 | MCNC | Dataflow | ALU |
| bmk3 | 82 | FPGA | Dataflow | code translation |
| hsim | 440 | FPGA | Mixed | logic simulation accelerator |
| C880 | 348 | ISCAS | Mixed | ALU and Control |
| C1908 | 429 | ISCAS | Dataflow | Error Checking and Correcting |
| C6288 | 2400 | ISCAS | Dataflow | 16-bit Multiplier |
| C7552 | 3283 | ISCAS | Mixed | ALU and Control |
| afmt | 551 | MIS | Control | control PLA from CPU |
| bam | 612 | FPGA | Mixed | Vector Adder |
| graphic | 8940 | HLL | Mixed | graphics engine |
| k2 | 2596 | MCNC | ? | misc logic |
| sampler | 605 | FPGA | Mixed | audio digitizer |
| soar | 510 | MIS | Control | control pla from SOAR CPU |
| gcd | 426 | HLL | Mixed | Finds greatest common divisor |
| phase_decoder | 428 | HLL | control | CD data decoder |
| daio_receiver | 495 | HLL | Mixed | CD interface to workstation |
| parker1986 | 1158 | HLL | Mixed | 8 bit alu and sequencer |
| dalu | 2075 | MCNC | Dataflow | Dedicated ALU |
| cordic | 70 | MCNC | Dataflow | coordinate transformation |
| i8 | 904 | MCNC | ? | misc logic |
| i9 | 1100 | MCNC | ? | misc logic |
| i10 | 2424 | MCNC | ? | misc logic |
| frg | 2331 | MCNC | ? | misc logic |
| **Total** | 44,634 | | | |

Table 2: Benchmarks used in the experiments

The primary measure of "goodness" used to compare the various architectural alternatives is the mean normalized area and mean normalized delay. In sections 3–8 we plot this mean area and delay versus each of the experimental factors, along with the 95% confidence interval for the mean. We can compare various architectural alternatives by determining whether or not the differences in their respective mean area or delay are statistically significant. By "statistically significant" we mean that the means are different at the 95% confidence level as determined by a two-sided $t$-test[35, 36]. We also use a number of other statistical tests, such as an Analysis of Variance (ANOVA), to check the influence of each of the experimental factors on the outcome and on each other.

## 3. Effect of Cell Granularity on FPGA Performance – Lookup Tables

In this section we describe our experimental results for lookup table cells. In subsection 3.1 we describe the details of the delay model we use. In subsections 3.2 and 3.3 we describe our experimental results for an FPGA based on a single-output lookup table cell and a multi-output lookup table cell, respectively.

### 3.1. Delay Model

The delay of a net is determined by the delay of the logic cell and the interconnect delay. The delay of a lookup table is a function of both its structure and the process technology. The structure used here is a multiplexer tree with the SRAM bits located at the multiplexer inputs. This is the structure used by a commercial FPGA[3] and results in a relatively compact layout for the small lookup tables typically used in an FPGA[37]. The worst case delay of this structure is linear in the number of cell inputs, $K$, i.e.

$$D_{cell}(K) = C_1 + C_2 K. \tag{3.1}$$

For an implementation in a 0.8 micron CMOS technology[38], typical values for $C_1$ and $C_2$ are 0.065 and 0.1 nanoseconds, respectively, as determined by Spice simulations. We model the output buffer of the cell in terms of its equivalent resistance, $R_{out}$.

The routing model assumed (pictured in Figure 4) is similar to Actel's segmented routing [7, 39]. The cell inputs are connected to vertical wire segments that extend over a horizontal routing channel. A switch (such as an antifuse [7] or RAM-controlled pass transistor [3]) is located at each intersection of the input segments with the horizontal wire segments located in the routing channel. The cell outputs are handled in a similar fashion, but the vertical segments extend over multiple routing channels. We additionally assume single segment routing [39] between cell outputs and cell inputs. This implies that each cell output passes through exactly two switches to reach any input that is connected to it, one to connect from the vertical output segment to a horizontal segment and one from the horizontal segment to the vertical input segment (Figure 4).

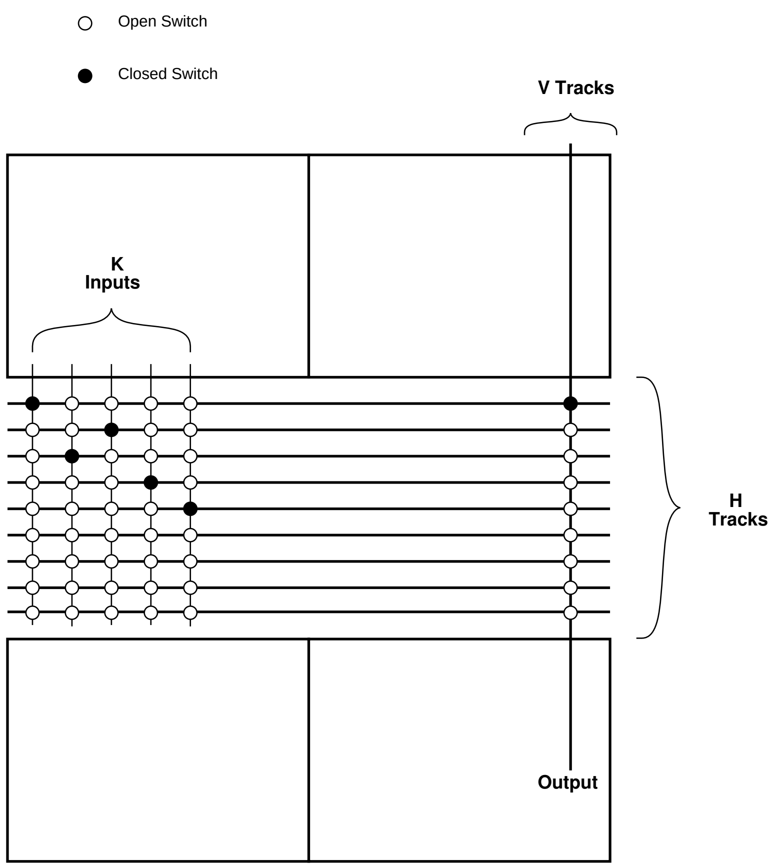


Figure 4: FPGA Routing Model

The switch used to program the routing is modeled very simply in terms of its equivalent on resistance, $R_{sw}$, and parasitic loading capacitance, $C_{sw}$. Note that a switch presents a loading capacitance to the wire segments to which it is attached even if it is not programmed. The switch time constant, $\tau_{sw}$, is $\tau_{sw} = R_{sw}C_{sw}$. By

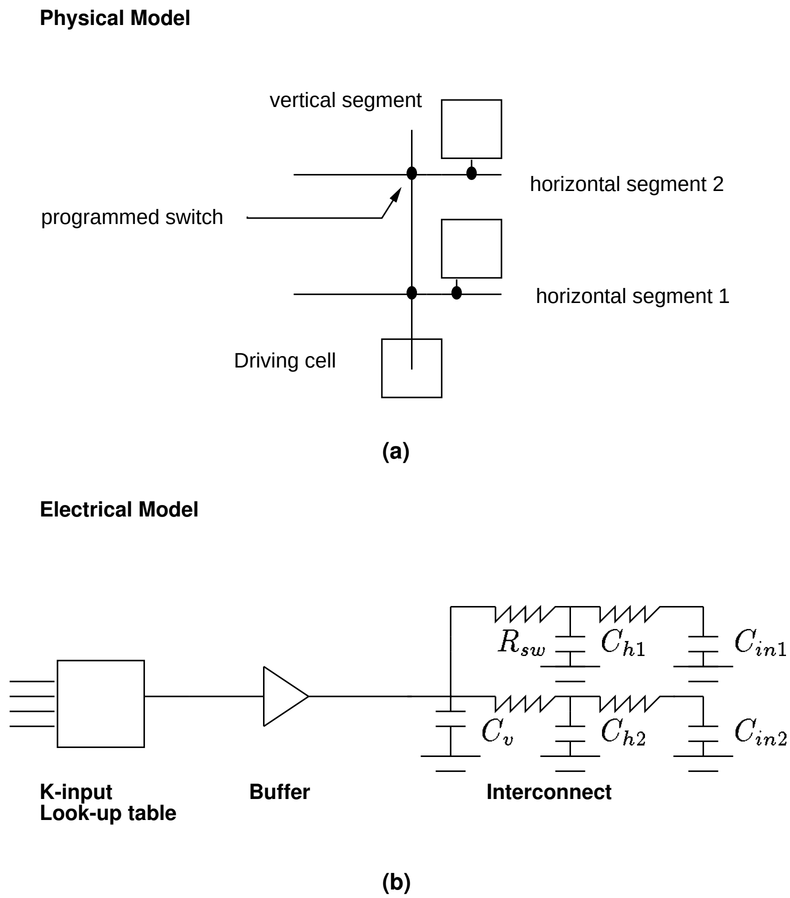


Figure 5: Interconnect Delay Model

appropriately choosing $\tau_{sw}$, this simple model can be used to represent a wide range of programming technologies, including various types of fuses and antifuses, pass transistors, EPROM cells, or some yet to be discovered technology. Table 1 gives typical values of $R_{sw}$, $C_{sw}$, and $\tau_{sw}$ for some representative programming technologies.

With this cell, switch, and routing model, it is now possible to compute the interconnect delay. For example, given the physical placement of cells shown in Figure 5(a), we can construct an RC tree connecting the cell output to each of its fanouts (Figure 5(b)). Capacitor $C_v$ corresponds to the loading capacitance on the vertical wire segment, and capacitors $C_{h_1}$ and $C_{h_2}$ correspond to the loading capacitances on horizontal segments 1 and 2, respectively. $C_{in_1}$ and $C_{in_2}$ are the input loading capacitances on the destination cells. These capacitances are determined as follows.

$$C_v = L_v \left[ HC_{sw} + (HP_h + \sqrt{A_{cell}})C_{m2} \right] \tag{3.2}$$

where $L_v$ is the vertical span of the net (in channels), $H$ is the number of horizontal routing tracks per channel, $P_h$ is the horizontal routing pitch, $A_{cell}$ is the area of a cell, and $C_{m2}$ is the capacitance per unit length of second level metal. The first term accounts for the loading capacitance of the switches, while the second term accounts for the capacitance of the vertical wire itself. Similarly,

$$C_h = L_h \left[ (2K + V)C_{sw} + \sqrt{A_{cell}}C_{m1} \right] \tag{3.3}$$

where $L_h$ is the span of the horizontal segment (in cells) and $C_{m1}$ is the capacitance per unit length of first level metal. The capacitance of the vertical input segment is calculated as

$$C_{in} = HC_{sw} + C_{cell\_input} + HP_hC_{m2} \tag{3.4}$$

where $C_{cell\_input}$ is the input capacitance of the logic cell. The delay of such an RC tree can be computed using a circuit simulator, such as Spice[40] or quickly estimated using the method of Penfield *et al.* [41]. For most of the experiments, we set $R_{out}$, the output buffer resistance, to be $0.1R_{sw}$, as the output buffer can be scaled (within area constraints) to minimize its effect on delay.

### 3.2. Single Output Lookup Tables

Our experimental results are shown in Figures 7 through 12. The error bars in all figures indicate the 95% confidence interval of the mean over all benchmarks.

The experimental procedure used is given in Figure 6. For each design and for each K we find the number of cells needed to implement the design, the horizontal and vertical track requirements H and V, calculate all the net delays, and determine all the critical path delays. We use the average of all the critical path delays (input to output or latch to latch) in the benchmark as a speed metric. As discussed in section 2.3.2, the baseline implementation for comparing the designs is the implementation for K=2. The delay for each design is normalized by dividing the values for $K = 3, \ldots, 10$ by the values for $K = 2$, which enables us to combine the data from the different benchmarks.

Figure 7 shows how the total number of logic cells in a design decreases as the granularity of the logic cell is increased. As K increases from 2 to 4, the number of

For each design in benchmark set

  For $K = 2$ to 10

    1. Remap logic design onto K-input cells(Chortle-CRF[24])
    2. Run placement (GDT AutoCells)
    3. Run global routing to determine horizontal and vertical segment lengths and channel densities
    4. Calculate net delays
    5. Back annotate netlist with delays and find critical path delays (Ceres[20])

  End

End

Figure 6: Experimental Procedure

cells required to implement a design decreases substantially, requiring less than half as many cells as for K=2. Similarly, the number of levels of logic in the critical path is also a decreasing function of K, decreasing approximately as $\log_K n$.[3]

The average net delay (Figure 8) is an increasing function of K – for three reasons. First, the number of switches capacitively loading each net is increasing (equations 3.2 and 3.3). Second, the average fanout is increasing (as it must, since the input to output ratio of the cell is changing). Finally, the wires themselves are becoming longer as the logic cell increases exponentially in area with K.

The combination of two factors, the decrease in number of levels of logic with K and the increase in cell and net delay, results in the curves of average critical path delay versus K shown in Figure 9. The top curve, $\tau_s = 0$, shows the critical path delay with zero interconnect delay. This is an increasing function of K as the cell delay is increasing faster than the number of blocks in the critical path is reduced. For non-zero values of $\tau_s$, the critical path delay first decreases, as the savings from having fewer cells in the critical path exceeds the increase in cell and net delay, then increases as the critical path ceases to be significantly improved with increasing K.

For each benchmark and each value of $\tau_{sw}$ the critical path delay is minimized for a specific value of K, denoted $K^*$ (Table 3). Figure 10 plots the mean value of

[3] In fact, the number of levels of logic of any nondegenerate $n$-input function implemented in $K$-input lookup tables is lower bounded by $\log_K n$ (see Spira[42]).

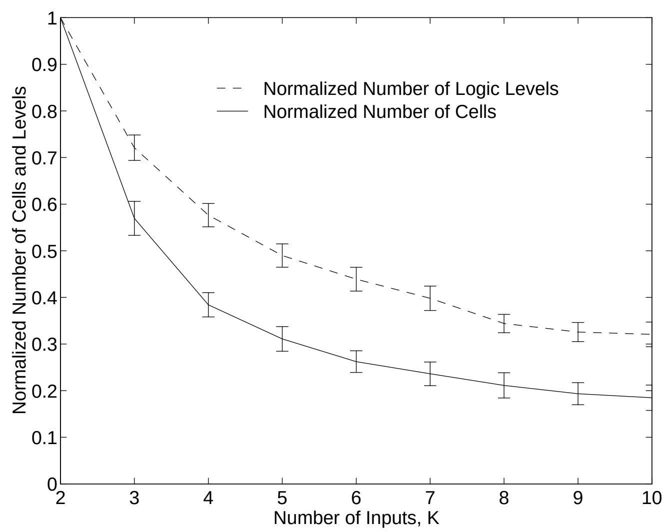


Figure 7: Number of Logic Cells vs. K

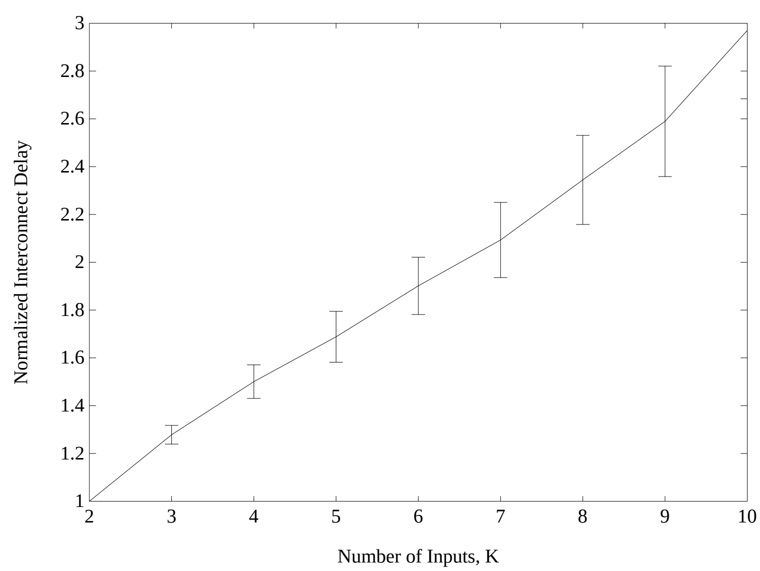


Figure 8: Average Net Delay vs. K

$K^*$ vs $\tau_s$. As hypothesized, technologies with small $\tau_s$ favor small granularity while technologies with larger $\tau_s$ favor larger granularity (see Figure 10).

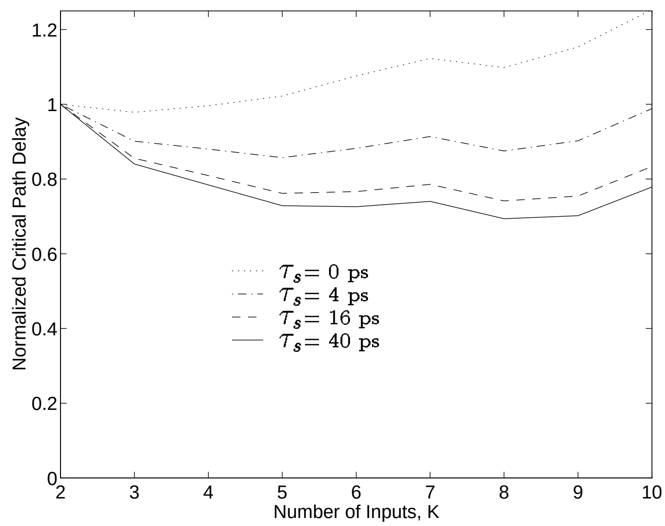


Figure 9: Critical Path Delay vs. K and $\tau_s$. The top curve, $\tau_s = 0$, shows the critical path delay with zero interconnect delay. The curves for $\tau_s = 4$ps and $\tau_s = 16$ps correspond to programming technologies such as the antifuse and pass transistor, respectively.

**3.2.1. Choosing the Best K.** Different benchmarks may achieve their minimum delay for different values of K, as seen in Table 3; however when an FPGA is designed, a specific value of K must be chosen. How should this value of K be chosen? It seems reasonable to choose K so that all benchmarks are implemented as close as possible to their best case implementation, $K^*$. That is to say, K should be chosen such that the average error

$$E(K) = \frac{1}{N} \sum_{B=1}^{N} \frac{\mathrm{Delay}_B(K)}{\mathrm{Delay}_B(K^*)} \tag{3.5}$$

over all benchmarks, B, is minimized. Note that this does not, in general, result in the simple average of the $K^*$ plotted in Figure 10 as the distribution of the $K^*$ for the benchmarks is not necessarily normal, nor does each benchmark's critical path delay

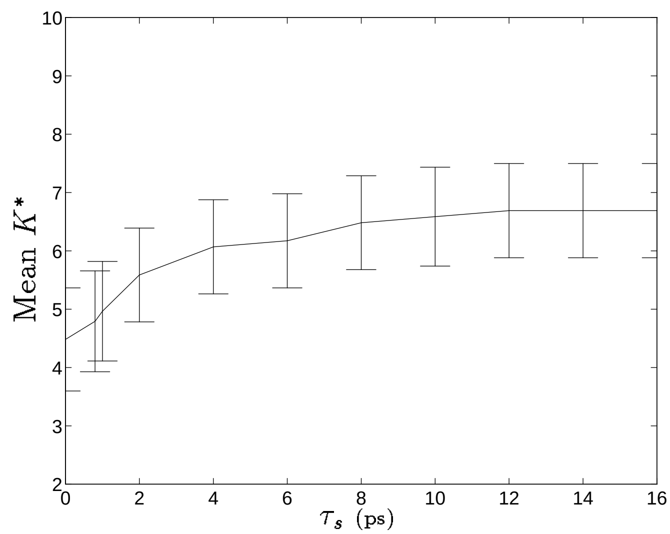


Figure 10: Mean $K^*$ vs. $\tau_s$. Low $\tau_s$ architectures (close to a mask-programmed gate array) favor small granularity while for larger $\tau_s$ larger granularity yields higher performance.

| Benchmark name | 2-input cells | Best K for $\tau_s$ 0 | 0.8 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|---|---|
| C499 | 392 | 5 | 5 | 5 | 5 | 5 | 5 |
| apex6 | 671 | 8 | 8 | 8 | 8 | 8 | 10 |
| decbig | 3247 | 2 | 4 | 4 | 4 | 5 | 5 |
| rot | 591 | 2 | 2 | 7 | 7 | 7 | 7 |
| des | 3218 | 9 | 9 | 10 | 10 | 10 | 10 |
| seq | 3338 | 3 | 3 | 8 | 8 | 8 | 8 |
| wmm | 586 | 7 | 7 | 7 | 7 | 7 | 7 |
| alu | 384 | 3 | 3 | 3 | 8 | 8 | 8 |
| bmk3 | 82 | 3 | 3 | 3 | 6 | 6 | 6 |
| hsim | 440 | 4 | 4 | 4 | 9 | 9 | 9 |
| C880 | 348 | 4 | 4 | 4 | 5 | 5 | 5 |
| C1908 | 429 | 2 | 2 | 6 | 6 | 6 | 6 |
| C7552 | 3283 | 4 | 4 | 4 | 4 | 5 | 5 |
| afmt | 551 | 3 | 3 | 5 | 5 | 8 | 8 |
| bam | 612 | 4 | 4 | 4 | 4 | 9 | 9 |
| graphic | 8940 | 8 | 8 | 8 | 8 | 8 | 8 |
| k2 | 2596 | 2 | 2 | 4 | 4 | 4 | 4 |
| sampler | 605 | 2 | 3 | 8 | 8 | 8 | 9 |
| soar | 510 | 3 | 8 | 8 | 9 | 9 | 9 |
| gcd | 426 | 10 | 10 | 10 | 10 | 10 | 10 |
| phase_decoder | 428 | 5 | 5 | 5 | 5 | 5 | 5 |
| daio_receiver | 495 | 4 | 4 | 5 | 5 | 5 | 5 |
| parker1986 | 1158 | 7 | 7 | 5 | 4 | 4 | 4 |
| dalu | 2075 | 5 | 5 | 5 | 5 | 5 | 5 |
| cordic | 70 | 3 | 3 | 3 | 3 | 3 | 6 |
| i8 | 904 | 3 | 3 | 3 | 3 | 3 | 3 |
| i9 | 1100 | 7 | 7 | 7 | 7 | 7 | 7 |
| i10 | 2424 | 2 | 3 | 3 | 3 | 3 | 3 |
| frg | 2331 | 6 | 6 | 6 | 6 | 8 | 8 |
| **Median** | | 4 | 4 | 5 | 6 | 7 | 7 |
| **Mean** | | 4.5 | 4.8 | 5.6 | 6.1 | 6.5 | 6.7 |

Table 3: Best K vs. $\tau_s$ for each benchmark

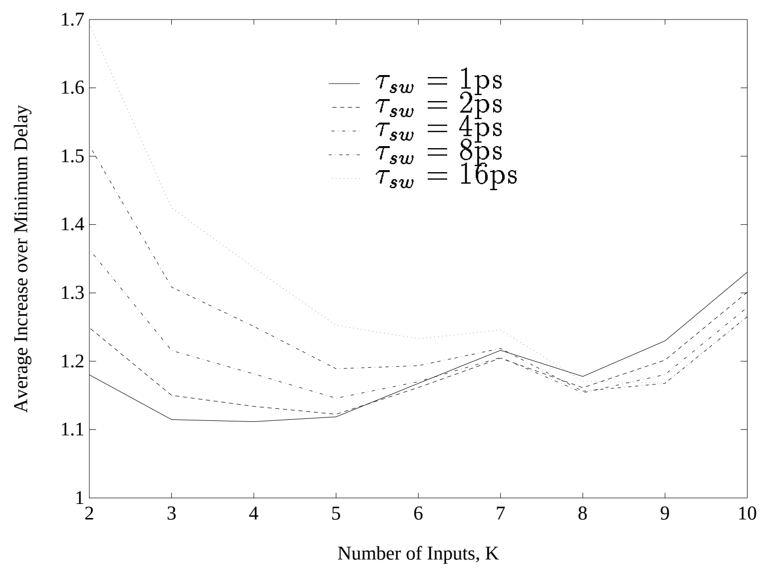


Figure 11: Delay Increase over Best Case versus K and $\tau_{sw}$

necessarily degrade equally in either direction as K is made larger or smaller than $K^*$. Figure 11 plots $E(K)$ for several values of $\tau_{sw}$. The value of K which minimizes $E(K)$ for various $\tau_{sw}$ is shown in Figure 12. These two figures clearly show that as the interconnect delay increases, a larger granularity cell yields lower critical path delay. Figure 11 also shows that for large $\tau_{sw}$, the penalty for choosing too small a granularity is very high, while the penalty for choosing too large a granularity is not as great. It is interesting to note in this figure that choosing $K = 8$ yields the least variation in performance with $\tau_{sw}$, a property that might be useful if the switch technology has a wide variation (with either manufacturing or programming) in resistance or capacitance.

**3.2.2. Statistical Significance of Effects.** An analysis of variance (ANOVA) was performed on the data for the curves in Figures 7–9, to see if the effect due to varying the granularity was significant compared to the variability between benchmarks. For all three cases, the effects were significant with $p < 0.0001$. The non-parametric Kruskal-Wallis test[35] was used to test the variation of the best K vs. $\tau_s$ shown in Figure 10, as the distributions of the $K^*$ were highly non-normal. In

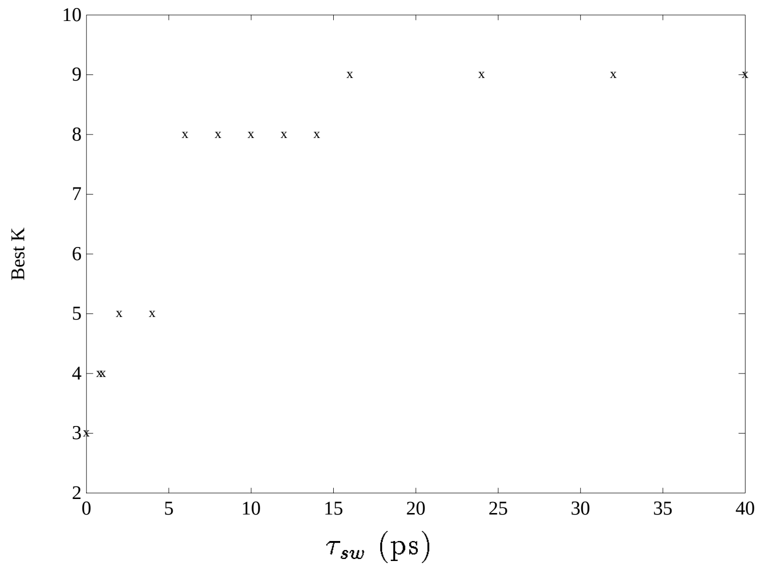


Figure 12: Best K versus $\tau_{sw}$

this case also, the effect was significant with $p < 0.0001$. No significant differences in behavior were found for large designs ($>$ 1000 cells) vs. small designs or for dataflow-oriented vs. control-oriented benchmarks (see Table 2 for benchmark sizes and types).

**3.2.3. Key Results.** The key results of our experiments on the effect of cell granularity on performance for FPGAs based on single-output lookup table cells are as follows:

- The best cell granularity for performance is a function of the RC delay, $\tau_s$, of the programmable switch.
- A cell with 4 to 5 inputs yields the lowest total delay for FPGAs using a programming technology with small $\tau_s$, such as an antifuse.
- A cell with a larger granularity, 8 to 9 inputs, yields the lowest total delay for FPGAs using a programming technology with a large $\tau_s$, such as the pass transistor.

- These results seem to be independent of design type (dataflow vs. control) or size.

### 3.3. Multi-Output Lookup Table Cells

In this section the experiments of Section 3.2 are extended to multiple output lookup tables. The delay of this cell is assumed to be the same as that for the single-output lookup table (equation 3.1), as it is possible to implement the cell as M independent K-input lookup tables. The area of the cell is M times the area of the single-output lookup table cell. Routing and delay calculation are performed in the same manner as for the single output cells.

The experimental procedure is similar to the one described above, except now we vary the number of cell outputs as well (see Figure 13).

```
For each benchmark
    For K = 2 to 10
        For M = 1 to 4
            1. Partition logic design into K-input
               equations (Chortle-crf)[25]
            2. Merge equations into (K,M) cells (DDmap)4
            3. Place and route
            4. Record number of cells and tracks
        End
    End
End
```

Figure 13: Experimental Procedure

For each K and M, we determined the number of cells and levels of logic required to implement each benchmark and then normalized these numbers to those for an implementation with (2,1) cells. The normalization allows us to combine the results for the various benchmarks, and to show the relative utility of adding more inputs

[4] DDmap is based on a multidimensional bin-packing algorithm. For a complete description see [29].

and outputs (see Figure 14). As seen in this figure, adding a second output can result in a significant reduction in a design's cell count while adding a third and fourth output yields diminishing returns, especially for small values of K.

The number of levels of logic in a design, however, is not significantly changed by using a multi-output cell (see Figure 14). It is clear that Spira's bound [42] of $\log_K n$ continues to apply to multi-output cells. There may be some small reduction in depth when both the true and complement of the same function are required; an output inverter is no longer needed and so the number of levels of logic is reduced by one.

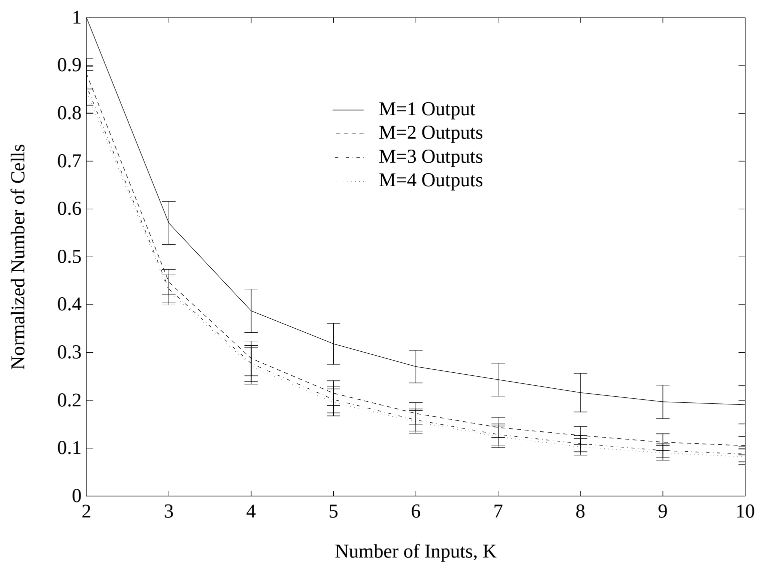


Figure 14: Normalized Number of Cells vs. K and M

Figure 16 shows the effect of using multi-output cells on the cell interconnect delay. For small values of K (less than 5), the interconnect delay for M=3 or 4 outputs is greater than that for M=1 or 2 outputs due to the increased wiring capacitance caused by the larger cell area. For larger values of K the increase in wiring capacitance is offset by a reduction in total cell count (resulting in lower wire lengths) and by a reduction in the cell fanout (caused by the lower input to output ratio per cell). Together these two factors cause the interconnect delay for the multi-output cells to be reduced over that for single output cells.

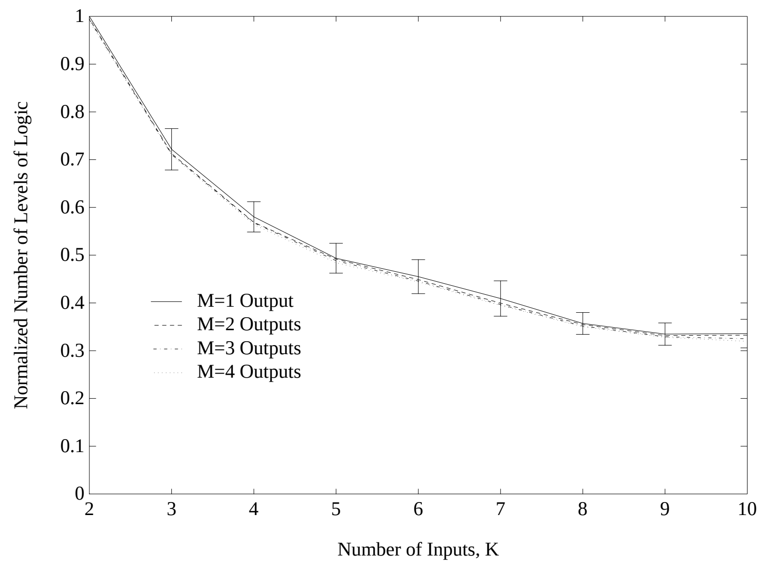


Figure 15: Normalized Levels of Logic vs. K and M

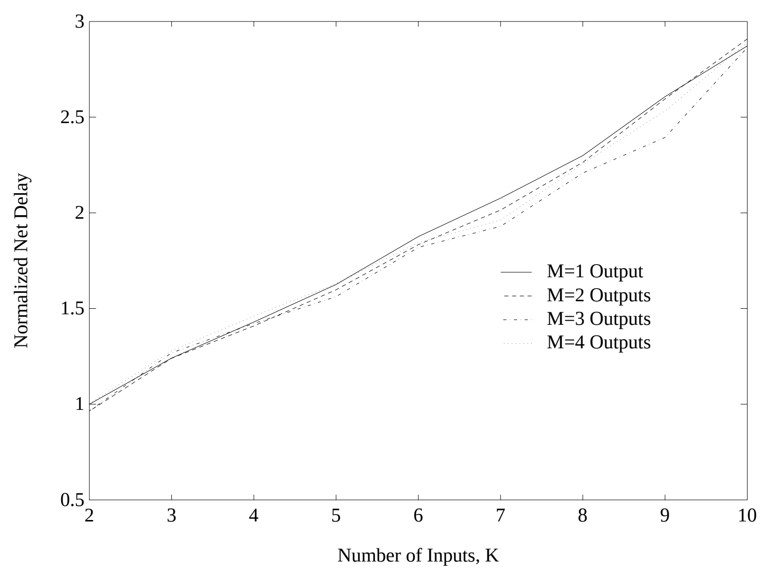


Figure 16: Average Interconnect Delay vs. K and M ($\tau_{sw} = 32ps$)

**3.3.1. Critical Path Delay vs. K and M.** Figures 17–20 show the effect that varying M, the number of cell outputs, has on the average critical path delay.

When the switch time constant is small, as in Figure 17, using a multi-output cell is of little benefit in decreasing critical path delay since, as noted above, the average interconnect delay may increase slightly while the number of cells and levels of logic does not change significantly.

For larger values of $\tau_{sw}$, multi-output cells offer limited benefits in reducing the critical path delay, with 2-output cells yielding the smallest delay for $\tau_{sw} = 4ps$ (Figure 18) and 3-output cells for larger $\tau_{sw}$ (Figures 19 and 20). The improvement over single-output cells is only a few percent, however.

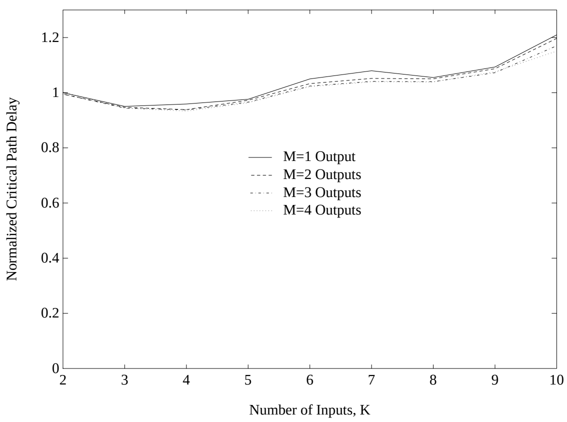


Figure 17: Critical Path Delay vs. K and M ($\tau_{sw} = 0.8ps$)

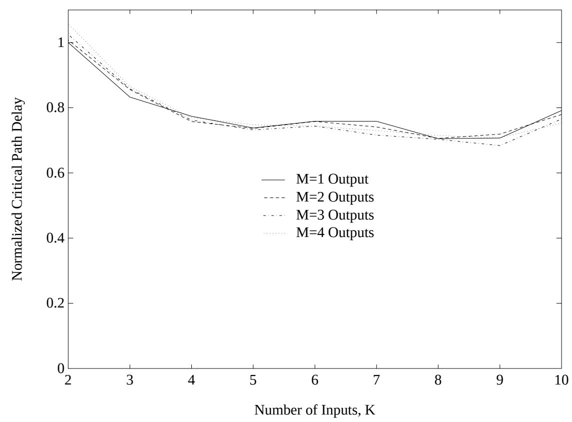


Figure 19: Critical Path Delay vs. K and M ($\tau_{sw} = 16ps$)

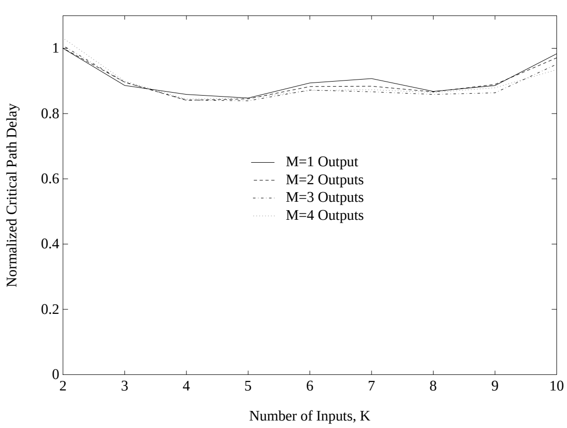


Figure 18: Critical Path Delay vs. K and M ($\tau_{sw} = 4ps$)

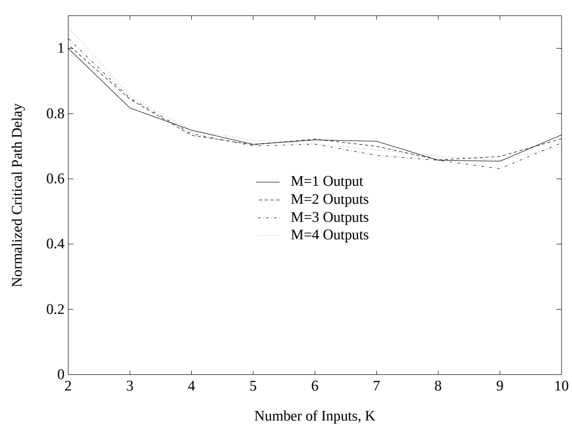


Figure 20: Critical Path Delay vs. K and M ($\tau_{sw} = 32ps$)

**3.3.2. Key Results.** The key results for our experiments on the effect of cell granularity on performance for FPGAs based on multi-output lookup table cells are as follows:

- Multi-output cells can significantly reduce the total number of cells needed to implement a logic design (Figure 14) but do not significantly reduce the number of levels of logic (Figure 15).
- Multi-output lookup table cells have little or no performance advantage over single output cells for FPGAs using a small $\tau_s$ programming technology and only very small advantages for FPGAs using a large $\tau_s$ programming technology.

## 4. Effect of Cell Granularity on FPGA Performance – PLA Cells

In section 6 we will show that multi-output lookup table cells, especially those of large granularity, are very poorly utilized and are wasteful of area. We also show that a multi-output PLA cell can be area efficient. Accordingly, in this section we repeat the experiments of section 3.3 for an FPGA based on a PLA logic cell. We want to determine if multi-output PLA cells yield good performance in addition to being area efficient.

### 4.1. PLA Cell Delay Model

The PLA cell we consider is a static NOR/NOR PLA as shown in Figure 21. The delay of such a PLA may be approximated by an RC model parameterized with number of inputs, K, outputs, M, and product terms N, as well as the equivalent resistances and capacitances of the transistors and programming elements. The critical path for such a PLA, with optimized device sizes, consists of the delay through the input buffer, the pull-down delay of the product term line, the pull up delay of the output (OR) line, followed by the output buffer delay (see [43], Chapter 5). Accordingly, we may estimate the PLA delay as a function of K, M, and N as follows.

The delay on the vertical input segments may be approximated by

$$D_{in} = R_{buf} N C_g \tag{4.1}$$

where $R_{buf}$ is the input buffer equivalent resistance and $C_g$ is the gate input capacitance of the programming transistor. Similarly, the delay on the horizontal product line is

$$D_{pt} = R_{pe}(2KC_d + MC_g) \tag{4.2}$$

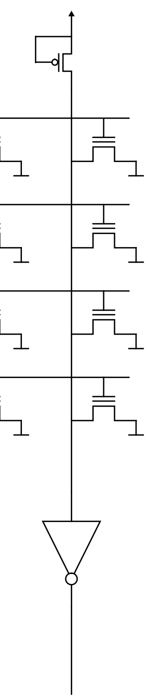

Figure 21: Pseudo-NMOS NOR/NOR PLA Circuit

where $C_d$ is the drain capacitance of the programming device. The delay on the vertical output segments may be approximated by

$$D_{out} = NC_dR_{pu}. \tag{4.3}$$

The total compute delay of the PLA is simply the sum of these delays plus the delay of the output buffer (which is computed in the same manner as for the lookup table cells),

$$D_{cell} = D_{in} + D_{pt} + D_{out} + D_{buffer} \tag{4.4}$$

### 4.2. Critical Path Delay vs. K and M

Using the above delay model, we repeated the experiments of section 3.3 for a PLA-based FPGA. We assume that the PLA cell is programmed with floating gate MOSFETs, as in the Altera[9] FPGA. The number of product terms per logic cell, N, was chosen for each value of K and M as described in section 8.2.

Figures 22 through 25 plot the mean normalized critical path delay for various values of $\tau_s$, the programmable switch time constant for the interconnect.

For small values of K and $\tau_s$, a single output cell yields the lowest critical path delay. This is because for small numbers of inputs, the extra outputs cannot be effectively utilized and serve only to load the product term lines, increasing the total delay. Since the cell delay is the dominant factor when $\tau_s$ is small, the critical path delay increases as the number of outputs, M, is increased (see Figures 22 and 23).

On the other hand, for large values of $\tau_s$ and large values of K, the additional outputs are utilized, and since interconnect delay now dominates, the additional cell delay is not as important. The reduction in the total number of cells and in the average cell fanout causes a reduction in the average net delay and hence multiple output cells perform marginally better than single output cells for large $\tau_s$ (see Figures 24 and 25. A cell with 8–10 inputs and 3–4 outputs yields the best performance for programming technologies based on pass transistors (SRAM or EEPROM based).

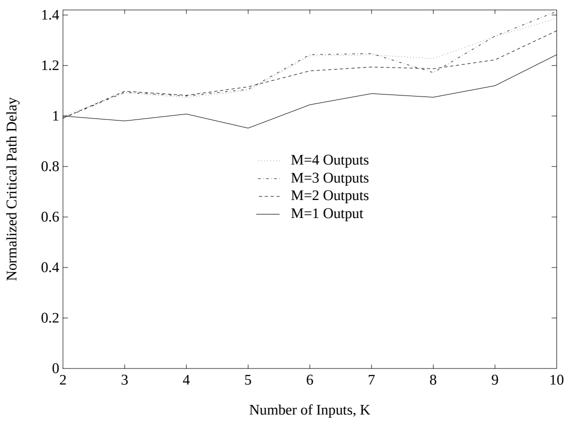


Figure 22: Critical Path Delay vs. K and M ($\tau_{sw} = 0.8ps$)

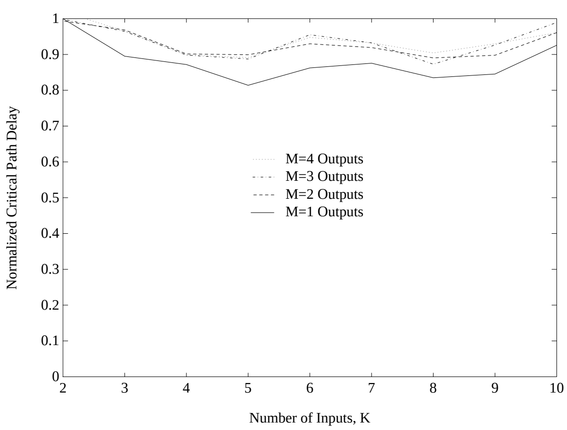


Figure 23: Critical Path Delay vs. K and M ($\tau_{sw} = 4ps$)

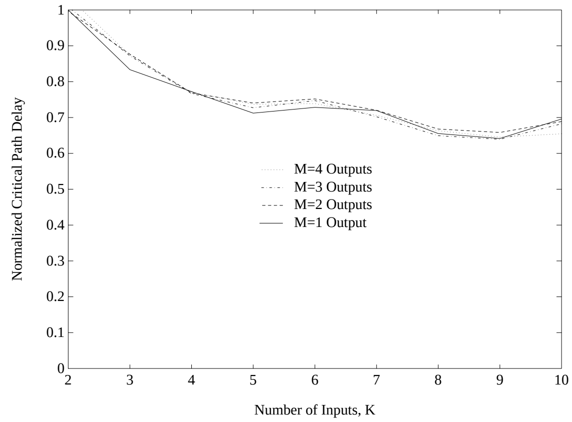


Figure 24: Critical Path Delay vs. K and M ($\tau_{sw} = 16ps$)

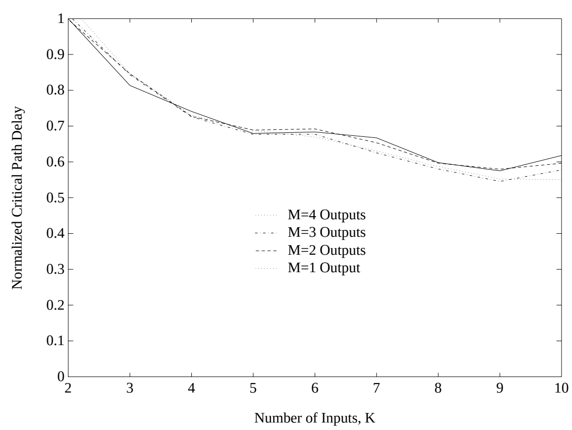


Figure 25: Critical Path Delay vs. K and M ($\tau_{sw} = 32ps$)

### 4.3. Key Results

The key results for our experiments on the effect of cell granularity on performance for FPGAs based on multi-output PLA cells are as follows:

- For small values of $\tau_s$ ($< 16ps$) a single output PLA cell yields the best performance.
- For large values of $\tau_s$ ($> 16ps$), a PLA cell with 8–10 inputs and 3–4 outputs yields the best performance.

## 5. Probabilistic Model for Delay

The experiments we have described are fairly time consuming, especially the process of performing a detailed placement and routing for each benchmark. Consequently, it is useful to develop an analytical model for the critical path delay through a combinational logic circuit based on minimal information. In particular, we would like a model which does not require that we compute the specific delays on each net in the design, but rather requires only the statistical distribution of the net delays. Such a model would be useful in making quick estimates of delay without going through the actual process of placement and routing. It could also be used as part of a high-level synthesis program – estimating the effect of alternative implementations on delay before a detailed mapping has been completed.

### 5.1. Bounds on Critical Path Delay

We wish to find upper and lower bounds for the average critical path delay of a combinational logic network implemented using K-input logic gates, e.g. a network implemented using K-input lookup tables (see Figure 26).

We assume that each logic gate has the same delay, $D_g$, and that the delays, $d_w$, on the wires connecting the gates are independent random variables all with mean $\mu$ and variance $\sigma^2$. For the moment, we also assume that all gates have fanout of one (we will relax this assumption later). We also assume that the logic network has $L$ levels. The critical path delay of such a network is defined as follows:

Let $P_1, \ldots, P_m$ be all possible paths from the network inputs to its

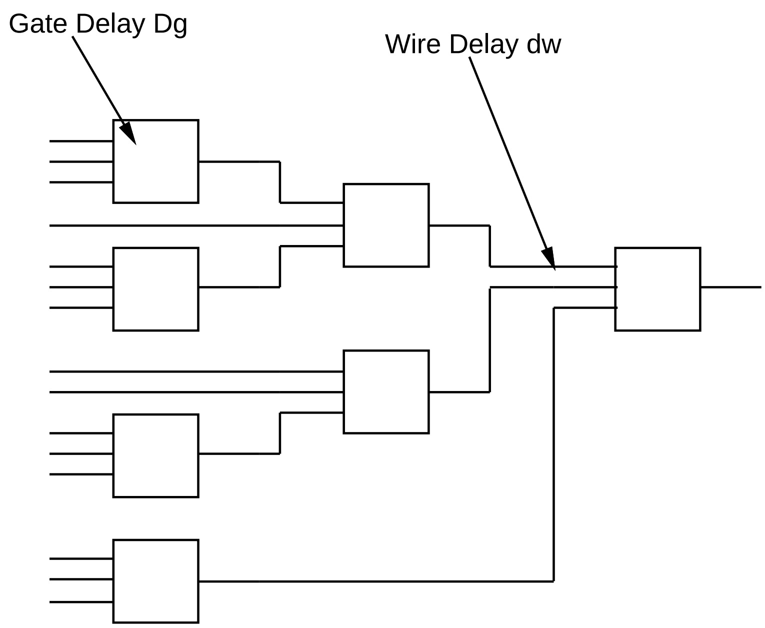


Figure 26: Logic network implemented in K-input lookup tables

output. The critical path delay is

$$D_{crit} = \max_{1 \leq i \leq m} \left( \sum_{j \in P_i} \left( D_g + d_{w_j} \right) \right) \tag{5.1}$$

This problem is closely related to bounding the average job execution time on an asynchronous multiprocessor system[44, 45]. The gates in Figure 26 are replaced by tasks (one per processor) and the combinational logic network is replaced by the task dependency graph for the job. The tasks may have an arbitrary number of inputs (dependencies) on their predecessors. Robinson[44] describes a lower bound, which we will review below, and a relatively loose upper bound that depends on the structure of the specific task graph being analyzed. Kung[45] describes an upper bound which does not depend on a specific task graph, but assumes that the tasks may have an unlimited number of inputs. We present an improved upper bound which makes use of the fact that the number of inputs to each gate in the logic circuit is bounded. In addition, our proof is constructive, whereas Kung's is by contradiction.

## 5.2. Lower Bounds

Taking the expectation of $D_{crit}$ we have:

$$E[D_{crit}] = E\left[ \max_{1 \leq i \leq m} \left( \sum_{j \in P_i} \left( D_g + d_{w_j} \right) \right) \right]$$

and since $E[\max\{x_i\}] \geq \max\{E[x_i]\}$ for any set of random variables $\{x_i\}$, we have

$$E[D_{crit}] \geq \max_{1 \leq i \leq m} \left( \sum_{j \in P_i} E[D_g + d_{w_j}] \right).$$

Since the number of levels in the longest path in the network is $L$, we may rewrite this as

$$\begin{aligned} E[D_{crit}] &\geq L\left[ D_g + E[d_{w_j}] \right] \\ &\geq L\left[ D_g + \mu \right], \end{aligned} \tag{5.2}$$

which provides a lower bound to the expected longest path delay.

## 5.3. Upper Bounds

In this section we find an upper bound to the expected critical path delay in a logic network with N K-input gates in L levels. We assume that the gates have a fanout of one and assume that all inputs are available at time $t = 0$.

We may upper bound $D_{crit}$ (equation 5.1) by summing the worst delay at each level of the network, e.g.

$$\begin{aligned} D_{crit} &= \max_{1 \leq i \leq m} \left( \sum_{j \in P_i} \left( D_g + d_{w_{i,j}} \right) \right) && (5.3) \\ &\leq \sum_{j=1}^{L} \left( \max_{i \in \text{level } L} \left( D_g + d_{w_{i,j}} \right) \right). && (5.4) \end{aligned}$$

Now assuming that there are $n_i$ gates at level $i$ of the network ($1 \leq i \leq L$), and that the wire delays $d_{w_i}$ are independent and identically distributed with cdf $P(x)$, mean $\mu$ and variance $\sigma^2$, we wish to find $n_i$, ($1 \leq i \leq L$) to upper bound the right hand side of

$$\begin{aligned} E[D_{crit}] &\leq E\left[ \sum_{i=1}^{L} \max(d_{w_{i,1}}, d_{w_{i,2}}, \ldots, d_{w_{i,n_i}}) \right] && (5.5) \\ &= \sum_{i=1}^{L} E\left[ \max(d_{w_{i,1}}, d_{w_{i,2}}, \ldots, d_{w_{i,n_i}}) \right], && (5.6) \end{aligned}$$

subject to the conditions that (1) the total number of gates is $N$, (2) there is at least one gate in each level, and (3) the maximum fanin of each gate is $K$. The cdf of $\max(d_1, d_2, \ldots, d_n)$ is

$$\begin{aligned} \Pr\{\max(d_1, d_2, \ldots, d_n) \leq x\} &= \\ \Pr\{d_i \leq x, 1 \leq i \leq n\} &= \Pr\{d_1 \leq x\} \cdot \Pr\{d_2 \leq x\} \cdots \Pr\{d_n \leq x\} \\ &= P^n(x) \end{aligned}$$

and $E[\max(d_1, d_2, \ldots, d_n)]$ is

$$E[P^n(x)] = \int_0^\infty [1 - P^n(x)]\, dx.$$

Therefore, combining all levels, we wish to maximize

$$\begin{aligned} &\int_0^\infty [1 - P^{n_1}(x)]\, dx + \int_0^\infty [1 - P^{n_2}(x)]\, dx + \cdots \int_0^\infty [1 - P^{n_L}(x)]\, dx \\ = &\int_0^\infty L - [P^{n_1}(x) + P^{n_2}(x) + \cdots + P^{n_L}(x)]\, dx, \end{aligned}$$

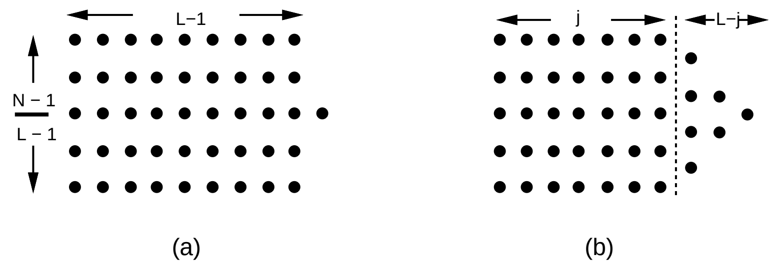


Figure 27: Arrangements of N cells into L levels with largest $E[D_{crit}]$

which is equivalent to minimizing

$$f(n_1, n_2, \ldots, n_L) \quad = \quad \alpha^{n_1} + \alpha^{n_2} + \cdots + \alpha^{n_L} \tag{5.7}$$

subject to

$$\sum_{i=1}^{L} n_i \quad = \quad N \tag{5.8}$$

$$n_L \quad = \quad 1 \tag{5.9}$$

$$n_i \quad \geq \quad 1 \tag{5.10}$$

$$n_{i-1} \quad \leq \quad K n_i. \tag{5.11}$$

Proposition 1 provides the solution to this constrained non-linear programming problem when the fanin is unbounded ($K = \infty$) and Proposition 2 extends the solution to the case of bounded $K$.

**Proposition 1.** *Of all possible arrangements of N gates into L levels, the arrangement with one gate at level L and $\frac{N-1}{L-1}$ gates at each of the remaining levels maximizes $E[D_{crit}]$ (see Figure 27(a)).*

The proof is given in Appendix A.

**Proposition 2.** *Of all possible arrangements of N gates (each with a maximum of K inputs) into L levels, the arrangement of an equal number of gates, m, to levels $1 \ldots j$, and $K^{L-i}$ gates to each level $i$, $i = j+1 \ldots L$, where*

$$m = \frac{N - \frac{1-K^{(L-j)}}{1-K}}{j}$$

*and $j$ is chosen to be the largest integer that satisfies*

$$m \geq K^{L-j}$$

*maximizes $E[D_{crit}]$ (see Figure 27(b)).*

The proof is given in Appendix A.

If $P(x)$, the cdf of the wire delays, is known, the upper bound on $E[D_{crit}]$ can be calculated directly. If P(x) is unknown, we may make use of the fact that[46]

$$E[max(d_1, d_2, \ldots, d_n)] \leq \mu + \frac{n-1}{\sqrt{2n-1}}\sigma.$$

For unlimited fanin the upper bounds become

$$E[D_{crit}] \quad \leq \quad L\left[D_g + \mu + \frac{\frac{N-1}{L-1} - 1}{\sqrt{2\frac{N-1}{L-1} - 1}}\sigma\right] \tag{5.12}$$

and with K bounded

$$E[D_{crit}] \quad \leq \quad L(D_g + \mu) + j\left[\frac{m-1}{\sqrt{2m-1}}\sigma\right] + \sum_{i=j+1}^{L} \frac{K^{L-i} - 1}{\sqrt{2K^{L-i} - 1}}\sigma \tag{5.13}$$

Finally, we note that these bounds continue to hold when the wire delays are not independent, but are *associated*[47]. Two random variables are associated if their covariance is greater than or equal to zero. In our problem this can occur if wires have fanout greater than one. The delays to each of the wire's fanouts will now be positively correlated since they all have in common the delay of the wire connecting them. Positively correlated random variables have positive covariance and are hence associated. Barlow and Proschan show ([47] Theorem 3.2) that if a set of random variables $X_1, X_2, \ldots, X_n$ are associated, then

$$Pr[X_1 \leq x, X_2 \leq x, \ldots, X_n \leq x] \leq Pr[X_i \leq x]^n.$$

Since this is the case, we can write

$$E[max(X_1, X_2, \ldots, X_n)] \quad \leq \quad \int_0^\infty [1 - P^n(x)]\, dx. \tag{5.14}$$

Using this fact together with the property that sums of associated random variables are also associated random variables, we can immediately see that the assumption of independence of wire delays results in the worst possible $E[D_{crit}]$ and the upper bounds we have found continue to hold when the wire delays are not independent.

Figure 28 shows an example of the upper and lower bounds on average critical path delay versus measured results.

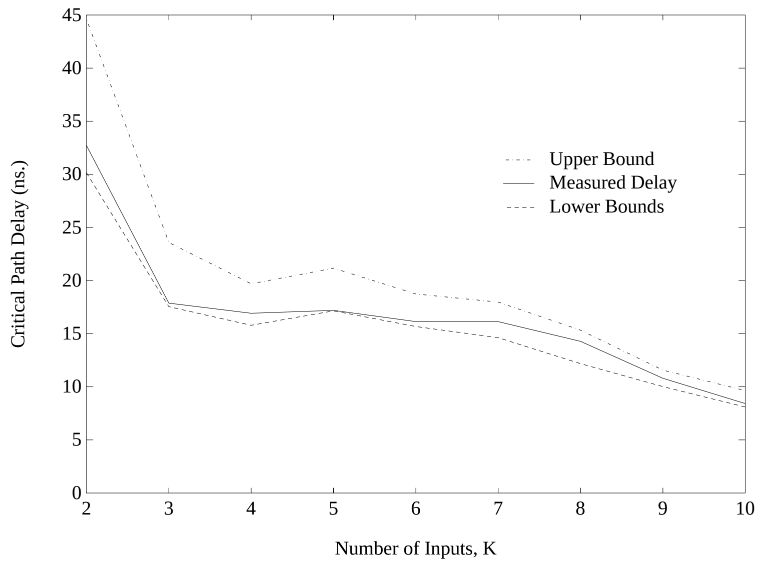


Figure 28: Upper and Lower Bounds versus Experiment

## 6. Effect of Cell Granularity on Density – Lookup Table Cells

In [12] Rose examined the tradeoff between FPGA density and cell granularity for single output lookup table cells. Here we extend his study to multi-output cells. We empirically determine the granularity, i.e. number of inputs and outputs, which yields the highest overall FPGA density.

### 6.1. Area Model

The cell we consider can implement any M Boolean functions of K variables (which we will denote as a (K,M) cell), implying a storage capacity of $M \cdot 2^K$ bits. The cell area model is similar to the one used by Rose [12], with two modifications. First, we extend his model to multiple output cells. Second, we decouple the horizontal routing pitch, $R_p$, from the cell bit area, $A_{bit}$. This allows us to consider their effects on the best granularity independently. The cell area, $A_{cell}$, is given by

$$A_{cell} = (2^K \cdot A_{bit} + A_{fixed}) \cdot M,$$

where $A_{bit}$ includes the necessary overhead for decoding. The fixed area, $A_{fixed}$, accounts for the D flip-flop, output driver, and associated fixed internal connections. We assume the cells are roughly square so that the cell width, $C_w$, is given by

$$C_w = \sqrt{A_{cell}}.$$

The routing area per cell, $A_{route}$, is then

$$A_{route} = C_w \cdot T \cdot R_p,$$

where $T$ is the number of horizontal wiring tracks in a wiring channel. Thus the total area for a design of size $N_{cells}$ is simply

$$A_{total} = (A_{cell} + A_{route}) \cdot N_{cells}.$$

### 6.2. Experimental Procedure

We use a set of 20 benchmarks chosen to represent a variety of design types (e.g. random logic, ALU, ECC, etc.) and sources (MCNC[48] and ISCAS benchmark sets, and real FPGA designs). Our experimental procedure is as follows:

```
For each benchmark
    For K = 2 to 10
        For M = 1 to 4
            1. Partition logic design into K-input
               equations (Chortle-crf)[25]
            2. Merge equations into (K,M) cells (DDmap)[5]
            3. Place and route
            4. Record number of cells and tracks
        End
    End
End
```

### 6.3. Number of Cells vs. K and M

For each (K,M), we determined the number of cells required to implement each benchmark and then normalized these numbers to those for an implementation with

[5]DDmap uses a multidimensional bin packing algorithm. For more information see [29].

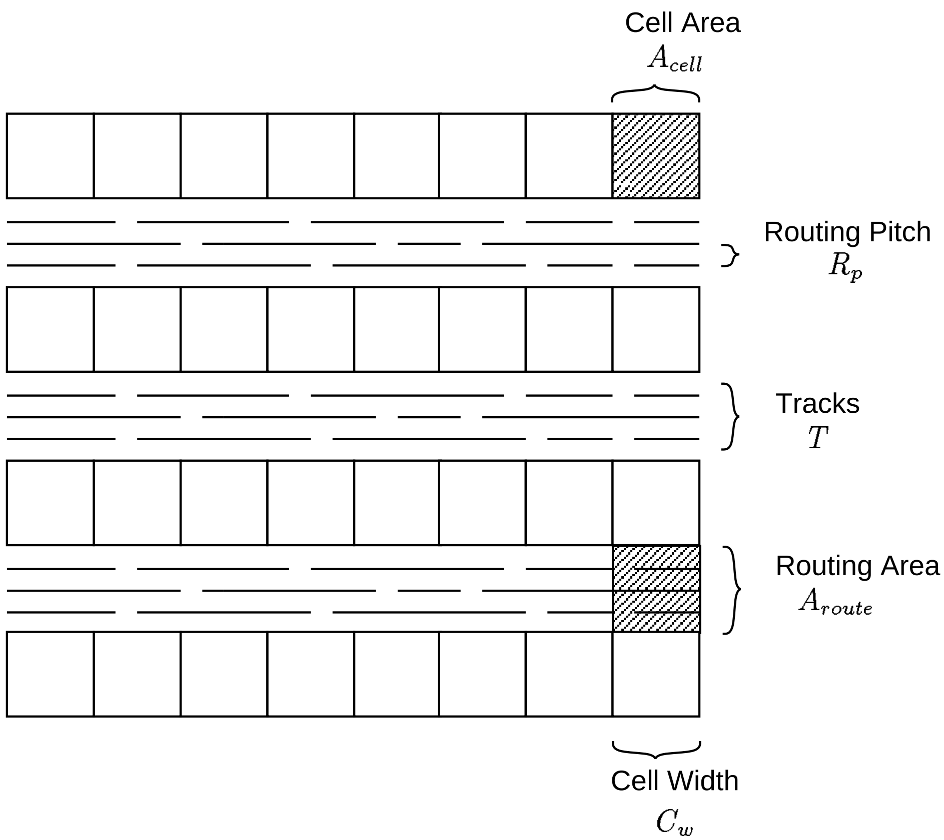


Figure 29: FPGA Area Model

(2,1) cells. The normalization allows us to combine the results for the various benchmarks, and to show the relative utility of adding more inputs and outputs (see Figure 30). Adding a second output can result in a significant reduction in a design's cell count while adding a third and fourth output yields diminishing returns, especially for small values of K.

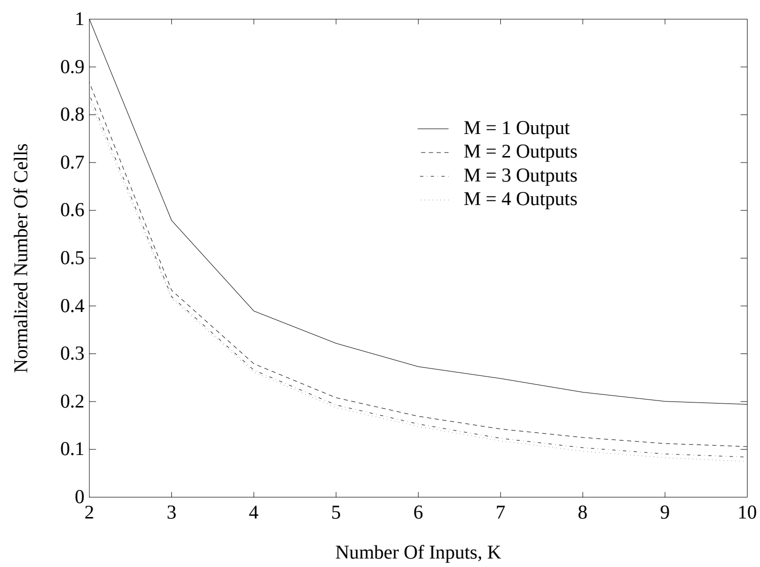


Figure 30: Normalized Number of Cells vs. K and M

## 6.4. Area vs. K and M

Figure 31 shows the average normalized $A_{total}$ versus K and M. (The error bars on all figures represent a 95% confidence interval of the mean over all benchmarks.) For this figure we have used a value of $1100\lambda^2$ for $A_{bit}$ [12, 49], a value of $13100\lambda^2$ for $A_{fixed}$, and a value of $34\lambda$ for $R_p$, which roughly corresponds to an SRAM-based FPGA. Note that for any K a single-output cell results in the smallest total area. This is because the reduction in number of cells for 2, 3, or 4 outputs is less than the doubling in size that occurs with each added output. For K $>$ 6, the differences between the 1 and 2 output cells are not statistically significant. Figures 32 and 33 show the effects of varying $R_p$ and $A_{bit}$ on the optimum K for single output cells, but the curves for 2, 3, and 4 outputs look much the same. As Figure 32 shows, varying

$R_p$, even over a wide range, does not have much effect on the optimum K. However, varying $A_{bit}$, as shown in Figure 33, has a greater effect on the best K.

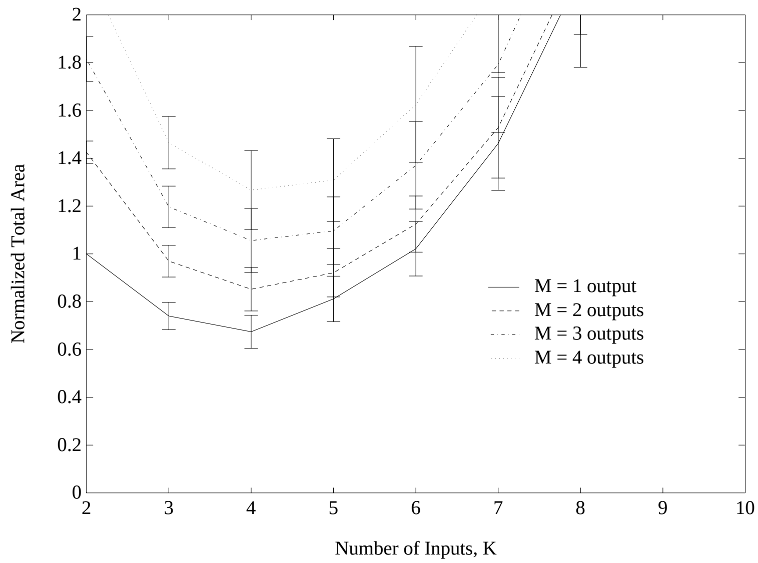


Figure 31: Total Area vs. K and M

The optimum K is largely determined by the ratio of $A_{bit}$ to $A_{fixed}$. By making a few simplifying assumptions, we can write an expression for the optimum K in terms of $A_{bit}$ and $A_{fixed}$. First, we ignore $A_{route}$, because as Figure 32 shows, it has a negligible effect on the optimum K. Second, we approximate the normalized number of cells versus K (Figure 30), by the curve

$$1.17 \log_K 2 - 0.17.$$

The form of this equation was suggested by the work of Spira[50], and provides a very good fit to the empirical data. Using this model and the cell area model described in section 6.1, the normalized cell area can be expressed as

$$(1.17 \log_K 2 - 0.17) \cdot (A_{bit} \cdot 2^K + A_{fixed}).$$

So given $A_{bit}$ and $A_{fixed}$, we can easily find the optimum K.

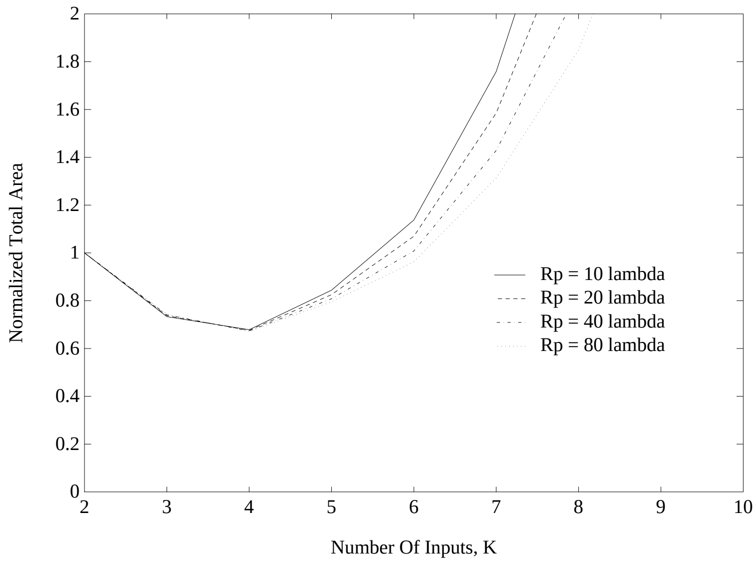


Figure 32: Total Area vs. K and $R_p$

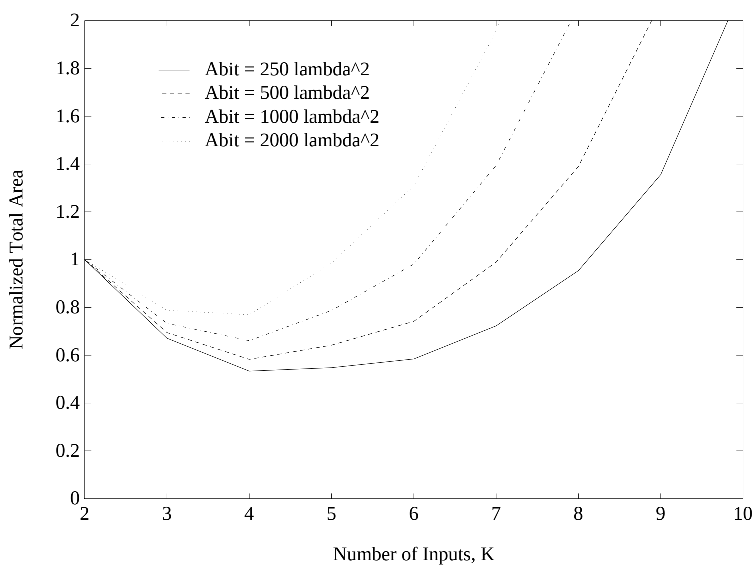


Figure 33: Total Area vs. K and $A_{bit}$

### 6.5. Key Results

The key results of our experiments on the effect of cell granularity on FPGA density for lookup table cells are as follows:

- The (4,1) lookup table yields the minimum $A_{total}$ of any (K,M) lookup table cell for a wide range of programming technologies and routing pitches.
- The best K for area is determined largely by the ratio of $A_{bit}$ to $A_{fixed}$ and exhibits little dependence on $R_p$.
- The number of cells required to implement a logic design by K-input lookup tables appears to be a function of $\log_K 2$. Fitting this function to the empirical curves of Figure 30 enables us to estimate the best K for area in terms of $A_{bit}$ and $A_{fixed}$.

## 7. Lookup Table Utilization

For an FPGA with large interconnect delays, a cell with more than 4 inputs yields better performance (see part I). In fact, for a pass-transistor based programming technology a cell with 8 or 9 inputs yields the best performance. A lookup table with much more than 4 inputs will be prohibitively large because of its exponential growth in size with the number of inputs. In this section we examine the complexity (in terms of the number of product terms) of the logic functions being mapped into the lookup tables to determine the average cell utilization. We find that the lookup tables are not fully utilized. This fact has motivated us to investigate structures such as PLAs which do not exhibit the exponential growth of the lookup table.

We first examine the cell utilization of an FPGA based on a single-output lookup table cell[12, 13, 3]. We define the cell utilization $U$ as the ratio of the number of product terms used per cell to the cell capacity, $2^{K-1}$.

For each of the benchmarks and for each value of K and M, we performed an exact minimization of the functions mapped into each lookup table cell using Espresso and counted the number of product terms in the minimized function. We were then able to determine the average number of product terms used per cell, $N_{avg}$, and the average cell utilization, $U_{avg}$ (see Figure 34). Also plotted in Figure 34 is the lookup table capacity, $2^{K-1}$, and the minimized number of product terms used for randomly generated K-input boolean functions.

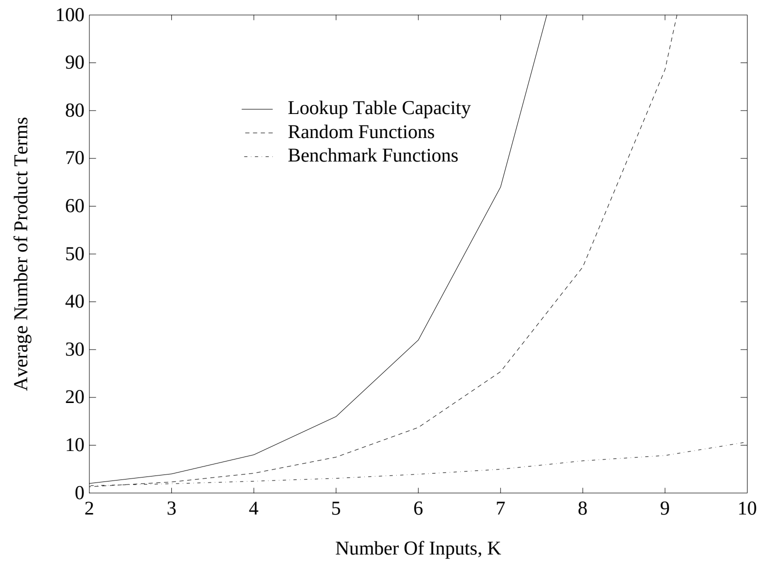


Figure 34: $N_{avg}$ vs. K

Figure 34 shows that $N_{avg}$ for the benchmark functions grows very slowly, almost linearly, with K. Its rate of growth is considerably less than the lookup table capacity, which grows as $O(2^K)$, or the rate of growth of randomly generated truth tables, $O\left(\frac{2^K}{\log K \cdot \log\log K}\right)$ [51, 52]. Most of the benchmarks have similar $N_{avg}$. The exceptions are C499 and C1908, which are error checking and correcting circuits dominated by XOR functions. Even for these benchmarks, however, $N_{avg}$ is considerably less than the lookup table capacity.

The average cell utilization, $U_{avg}$, as defined above is plotted in Figure 35 for 1, 2, 3, and 4 output cells. We see that lookup table based cells become increasingly underutilized with increasing K and M.

The relatively low number of product terms used per cell immediately suggests the use of a PLA-based cell. Such a cell allows for flexible function implementation, but grows in area as $K^2$ (since the PLA has K inputs and $N_{avg}$ is linear in K). In the next section, we repeat the experiments of section 6 for an FPGA using a PLA logic cell.

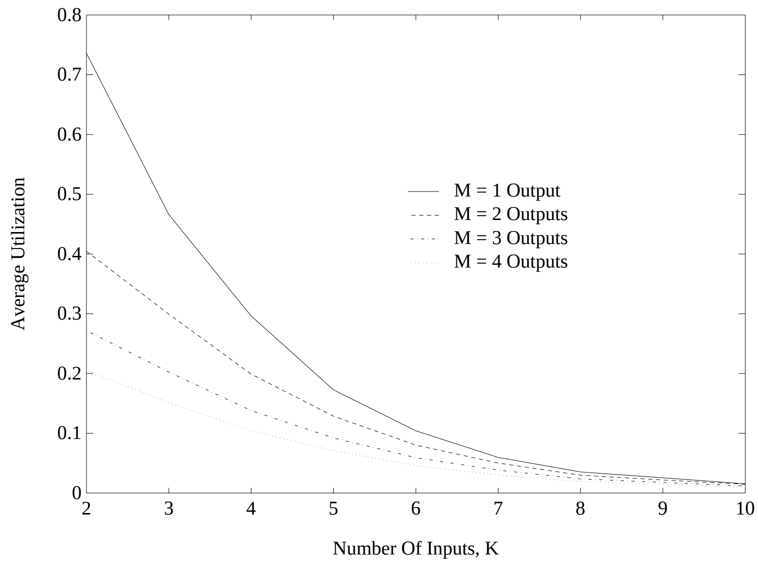


Figure 35: $U_{avg}$ vs. K and M

### 7.1. Key Results

The key results of our experiments on lookup table cell utilization are as follows:

- Excluding the ECC benchmarks, $N_{avg}$ grows linearly with K, rather than exponentially.

- The relatively low $N_{avg}$ suggests that PLA based cells may be a good choice for larger cell granularities.

## 8. Effect of Cell Granularity on Density – PLA Cells

We use the same FPGA model described in section 2, but replace the lookup table cell with a K-input, M-output, N-product term PLA, denoted (K,M,N). Each output can be optionally followed by a flip-flop, as before.

## 8.1. Area Model

The PLA cell area model is based on the layout of a static pseudo-NMOS PLA generated by MPLA[53]. We assume that the elements used to program the PLA (e.g. fuses, EPROM cells, etc.) are roughly square and have an area $A_{pe}$. The cell width (in lambda) of this layout is then given by

$$C_w = \max(18, \sqrt{A_{pe}}) \cdot K + \max(10, \sqrt{A_{pe}}) \cdot M + 98,$$

and the cell height is given by

$$C_h = \max(10, \sqrt{A_{pe}}) \cdot N + 136.$$

Thus the PLA cell area is given by

$$A_{cell} = C_w \cdot C_h + A_{fixed} \cdot M,$$

where $A_{fixed}$ is defined as for the lookup table cell. $A_{total}$ is computed the same way as for the lookup table.

## 8.2. Optimizing $N$

In addition to choosing K and M, we also choose N to minimize the total cell area as follows. Let $p_B$ be the probability mass function of the number of minimized product terms per cell in each benchmark (Figure 36) and $N_{max}$ be the maximum number of product terms in any cell in the benchmark. We assume that each lookup table with less than N product terms fits into one PLA and those with more than N product terms must be broken up into several PLAs. For each benchmark and for each value of K and M we want to choose N such that the total cell area, $A_{cells}(N)$, is minimized, where

$$A_{cells}(N) \quad = \sum_{j=1}^{N_{max}} p_B(j) A_{cell} \cdot \left\lceil \frac{j}{N} \right\rceil . \tag{8.1}$$

A plot of $A_{cells}(N)$ for a typical benchmark is shown in Figure 37. Figures 36 and 37 show that if $N$ is chosen too small, many lookup tables will have to be broken up into multiple PLA cells and $A_{cells}$ will be very large. $A_{cells}$ has a minimum (around

10-12 in Figure 37) that corresponds to capturing the majority of the lookup tables into one PLA cell. Increasing $N$ beyond this point wastes area as very few of the lookup tables requires more product terms. For each benchmark and each value of K and M, an area-minimum value of $N$ can be determined. This optimal value of N is an increasing function of K and M (Figure 38) and is fairly close in value across all the benchmarks. The exceptions are the ECC benchmarks, as noted above.

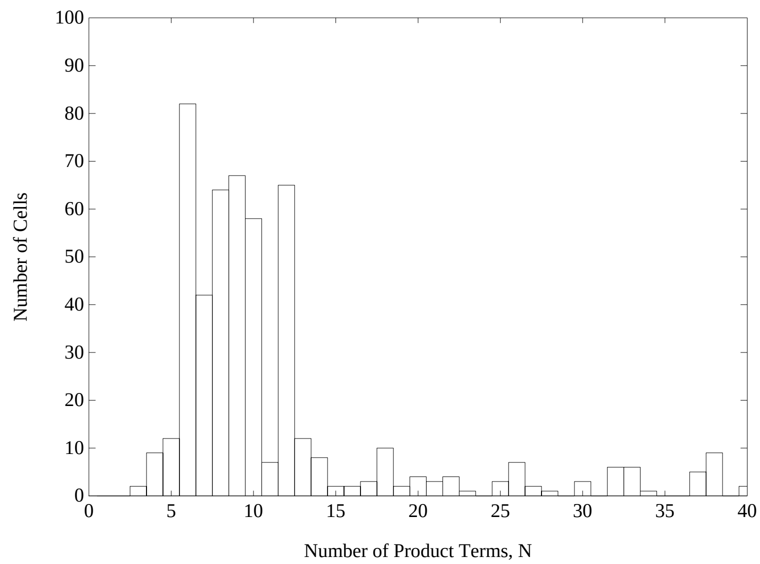


Figure 36: Histogram of the Number of Product Terms per Cell (Benchmark graphic, $K = 10$, $M = 3$)

## 8.3. Experimental Procedure

The experimental procedure is the same as in section 6.2, except that after mapping the equations onto (K,M) lookup tables, we minimize the functions mapped into the lookup tables using Espresso.

## 8.4. Area vs. K and M

Figure 39 plots $A_{total}$ versus (K,M) as computed using the area model described

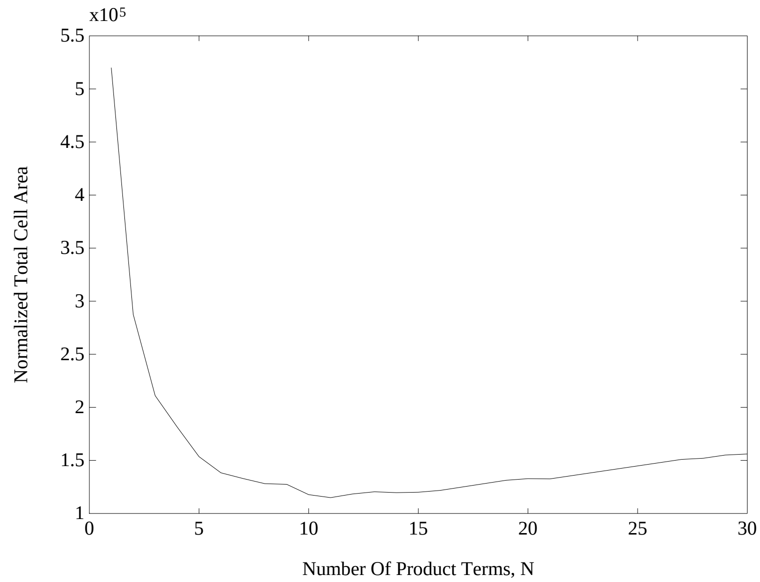


Figure 37: $A_{cells}$ vs. N for a specific benchmark and (K,M)

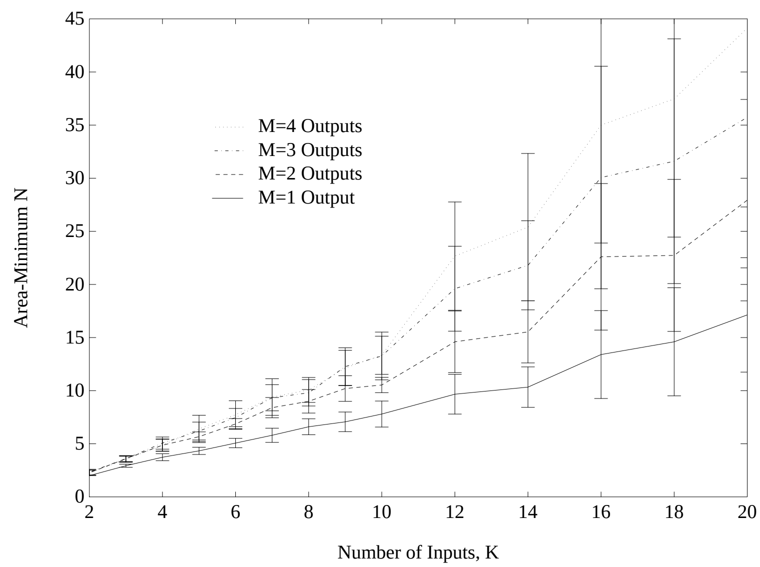


Figure 38: Area-Minimum N vs. K and M

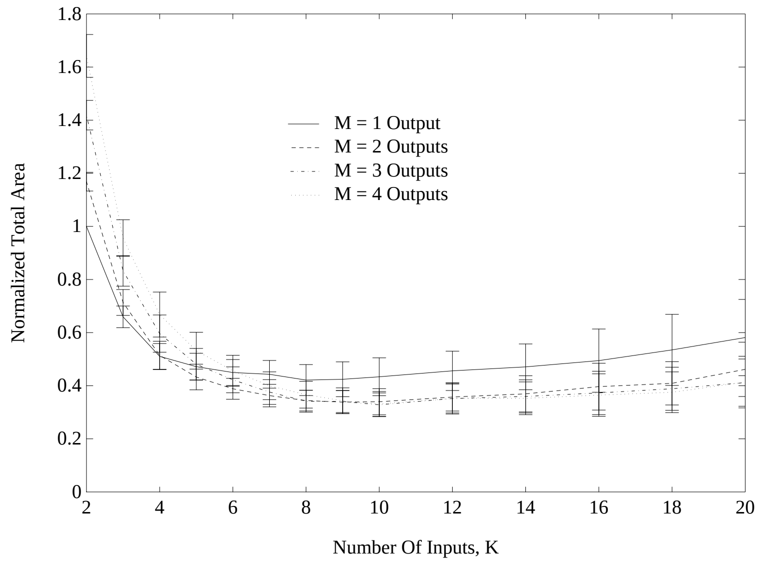


Figure 39: Total Area vs. K and M

in section 8.1[6]. For each (K,M) and each benchmark, the best N was chosen using the method described in 8.2. In this figure we used a value of $200\lambda^2$ for $A_{pe}$[54], a value of $13100\lambda^2$ for $A_{fixed}$ and an $R_p$ of $15\lambda$, which roughly corresponds to an EPROM based programming technology. $A_{total}$ is smallest for a PLA with 8–10 inputs, 3–4 outputs, and 12–13 product terms. Note that unlike the lookup table cell, additional outputs are of increasing utility as K increases. One output cells are the smallest from K=2 to K=4, two outputs cells from K=4 to K=7, and three output cells beyond K=7. For K>4, the differences between 2, 3, and 4 output cells were not statistically significant.

### 8.5. Comparison with Lookup Tables

To compare the relative area efficiency of the lookup table cell and the PLA cell, we compared the smallest lookup table implementation, (4,1), with the smallest PLA implementation, (10,3,12). For a fair comparison, we considered both imple-

[6]In this figure we have plotted a subset of the benchmarks that does not include the ECC benchmarks (C499 and C1908) as they distort the trends followed by most of the benchmarks. To handle these sorts of designs well, it would be useful to add XOR gates external to the PLA as done in [9, 10].

mentations in the same programming technology, the EPROM cell. We found that $A_{total}$ of the PLA cell implementations ranged from 71% to 154% of the lookup table based implementations. The worst results were obtained with the ECC benchmarks. Exclusive of these two benchmarks, the PLA implementations ranged from 71% to 136% of $A_{total}$ for the lookup table implementations, and on the average were about 4% smaller. A disadvantage of these PLA-based implementations is that they will dissipate static power. On the other hand, the large grained PLA implementations will require 25% fewer wiring tracks on average, which would lead to better performance[13].

The PLAs are still relatively sparse, however, and could be improved over the simple-minded model considered here. We have collected a number of statistics on the PLAs and found that on average only about half of the inputs are involved in any one product term (Figure 40). This suggests that a fixed product term folding could be used to reduce the size of the PLA. We also found that in the multi-output PLAs, only about 10% of the product terms were shared between outputs (Figure 41). This suggests that fixing the OR-plane (as in PALs) would further reduce the PLA size. On average, less than half of the input variables occur in both true and complement form, which suggests that some of the input inverters could be deleted (Figure 42). On average with (10,3) PLA cells, only 4 inputs are involved in each product term, each product term is involved in about 1.1 outputs, and about 3 of the inputs require input inverters.

### 8.6. Key Results

The key results of our experiments on the effect of cell granularity on FPGA density for FPGAs using PLA cells are as follows:

- For the technology parameters and layout considered here, a PLA with 8–10 inputs, 3–4 outputs, and 12–13 product terms gave the smallest $A_{total}$.
- A simple PLA implementation, such as the one considered here, results in an $A_{total}$ similar to a (4,1) lookup table implementation when compared in the same technology.
- Benchmark statistics show that several methods may be useful in reducing the PLA size: fixing the OR plane, product term folding, and removing some of the input inverters.

- Combining the results of part I with the results described here, we see that FPGAs based on a PLA cell achieve maximum performance and maximum density at the same cell granularity (8-10 inputs, 3–4 outputs). This is in contrast to the lookup table cells, which reach maximum density with a 4-input, 1 output cell but whose maximum performance is reached with a larger cell with 8 or 9 inputs and one output.

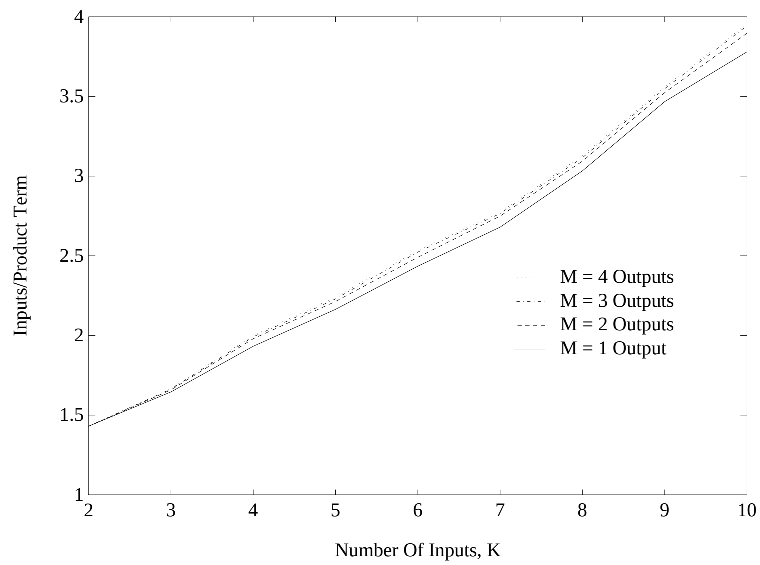


Figure 40: Inputs/Product Term vs. K and M

## 9. Conclusions

In this paper we described an FPGA Architecture Workbench and its use in systematically evaluating the effect of various architectural parameters on the density and performance of an FPGA.

In sections 3 and 4 we investigated the effect of cell granularity on FPGA performance. We have shown that the best granularity for an FPGA cell is an increasing function of the switch delay, $\tau_s$, of the interconnect network. We have generated empirical curves showing the optimum cell granularity (in terms of the number of cell inputs and outputs) as a function of $\tau_s$ for two families of FPGA cell, the lookup table and the PLA. We also presented upper and lower bounds on the average critical

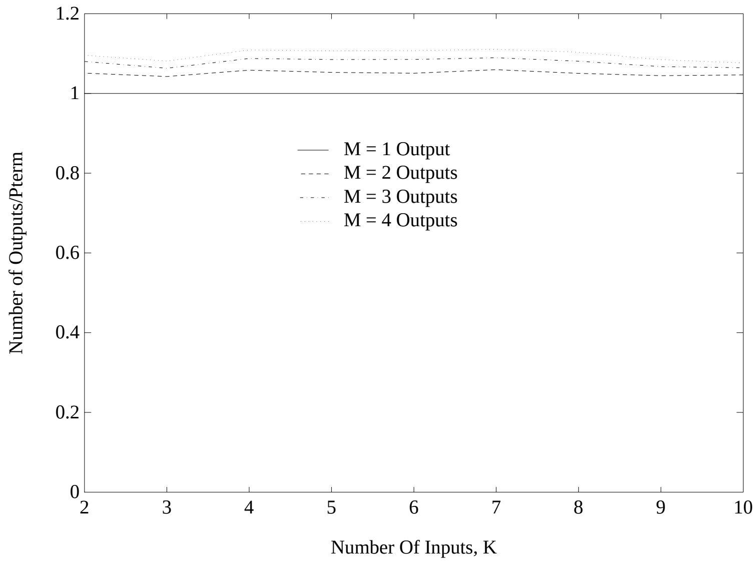


Figure 41: Outputs/Pterm vs. K and M

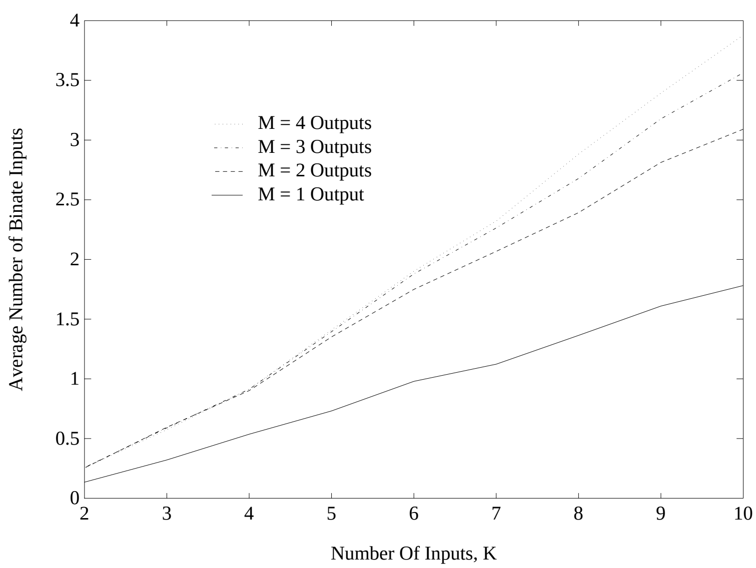


Figure 42: Binate Inputs vs. K and M

path delay of a combinational logic network constructed of K-input cells connected by wires with random, independent, identically distributed delays. The bounds depend only on the number of logic levels, the cell delay and the wire delay distribution and not on the network structure.

In sections 6 and 8 we investigated the effect of cell granularity on FPGA density. For the lookup table, we found that the 4-input, 1-output cell yields the smallest overall area for a wide range of programming technologies. For PLA cells, a larger cell with 8–10 inputs and 3–4 outputs yielded the smallest area. Combining these results with those we obtained for FPGA performance leads us to conclude that FPGAs based on large granularity PLAs hold the potential for yielding better performance in about the same area as the minimum area lookup table cell.

### 9.1. Implications for Current FPGA Architectures

Our results can be used to analyze existing, commercial FPGA architectures to see if they have made an appropriate choice of cell granularity for their chosen programming technology. We will review several of the architectures in turn.

The Plessey[55] FPGAs utilize SRAM as the programming technology and a 2-input NAND gate as the logic cell. Figures 10 and 11 clearly show that, for an FPGA using pass transistors as the programming technology (i.e., $10 \leq \tau_s \leq 40$ ps), a 2-input cell is far from optimal. The small granularity means that the relatively slow interconnect network will have to be used very often, resulting in delays that are more than 150% higher than could be achieved with a larger cell granularity (see Figure 11).

Such a small cell is not optimal from an area point of view either. The 2-input cell has a large overhead area (for the D-latch and associated programming logic) as compared with the area used for performing logic. Very few designs would make use of a D-latch for every NAND gate. We can infer from Figure 33 that a larger granularity cell makes more sense from a density point of view as well.

The Xilinx 2000 series cell[3] utilizes SRAM as the programming technology and a 4-input lookup-table as the logic cell. Figures 11 and 12 show that a larger granularity cell, in the range of eight to nine inputs, would probably yield delays 15-20% smaller.

However, as Figure 31 shows, a cell with eight or nine inputs would be prohibitively large – in fact it would double the total FPGA area. Consequently, in designing FPGAs with lookup tables a difficult tradeoff between performance and density must be made. Competitive pressures have increasingly forced this tradeoff in the direction of higher performance. This is evidenced by the fact that later generations of the Xilinx FPGA, the 3000 and 4000 series, have moved to larger granularity logic cells. The 3000 series uses a 5-input lookup table (reconfigurable to 2 4-input lookup tables sharing a total of 5 inputs). The 4000 series utilizes two 4-input lookup tables with completely independent sets of inputs feeding into a third 3-input lookup table, allowing it to implement some functions of as many as nine variables. The ORCA FPGA[56, 57], another second generation lookup table FPGA, utilizes a six input logic cell.

The Actel Act-1 cell[7] is based on a 4-input, 1-output multiplexer. As such, it can implement any function of 3 input variables. The extra OR gate at the input to the second multiplexer gives it the capability of implementing a subset of functions with 4 to 8 inputs. Mapping experiments show that the Actel cell is approximately equivalent to a lookup-table cell with 3 to 4 inputs. Figure 11 shows that this is very close to the optimal granularity for an FPGA with a programmable switch time constant in the neighborhood of 1 to 4 picoseconds, such as the ONO antifuse used in the Actel FPGA. The Act-1 FPGA used an antifuse with $\tau_s = 4$ps, for which a cell with 5 inputs is optimal. The most recent version of the Actel FPGA, the Act-3[58], utilizes an improved antifuse with $\tau_s = 1.6ps$, for which a cell with 3 inputs is best. The performance of the Actel cell is within 10% of optimal in both cases.

Similarly, Figures 32 and 33 show that a cell with from 3 to 4 inputs is optimal from an area point of view as well[7].

The Altera Logic Array Block[9] is essentially a PLA with from 20-100 inputs and 16 outputs. It is difficult to directly apply the results of sections 4 and 8 to the Altera cell, as the Altera FPGA partially combines the cell routing and programming area. Additionally, the cell to cell routing is performed by a crosspoint switch rather than by channel routing. However, we may make several observations by using the results of sections 7 and 8.

The results from section 7 show that most of the PLAs will be fairly sparsely utilized. On average, only 3 to 5 product terms per output can be utilized. Altera's MAX 5000 series cell has 3 product terms per output, while the MAX 7000 series has 5; these seem to be reasonable choices given our statistics.

On the other hand, Figure 40 shows that, on average, only half the inputs to a cell are used in any one product term. This suggests that not all the inputs to a Logic Array Block need to connect to each and every product term. Instead, it

[7]Note that while the area models in section 6 are for K-input lookup tables, they can also be applied to a K-1 input multiplexer cell.

should be possible to partially depopulate the AND plane of the Altera cell and have each product term attach to only a subset of the inputs (as in a folded PLA). While doing this may not save much area, due to the regular routing structure of the Altera cell, it could improve performance substantially, as the capacitive loading on each of the input lines and product terms could be halved. The tradeoff in doing this is that now the product terms are not completely interchangeable – one must choose the product term line to use based on what inputs it connects to. This introduces a placement phase into the CAD process, a function not needed in their current CAD system.

### 9.2. Limitations

The experiments in this paper have been limited to one routing scheme (segmented channel routing), one set of placement and routing tools, and (primarily) a single technology mapping program in order to isolate the effect we wanted to study. Each of these limitations could have an effect on the results. We address each of these factors in turn below.

The density achieved by the simple single-segment routing scheme used has been previously shown [39] to differ by only a small constant factor from unrestricted channel routing, hence we do not believe it would change the basic tradeoffs we observed. However, other routing schemes, such as hierarchical routing, could change the results.

The technology mapping and placement and routing tools we used optimized for area. It is possible that performance-directed CAD tools might perform differently. However we note that Singh *et al*[14] have performed a similar set of experiments using performance-directed technology mapping tools and observed tradeoffs very similar to the ones we observed. Their experiments, performed concurrently with those described in this paper, use a fixed number for interconnect delay on each net and do not use actual placed and routed delays.

Just as compiler technology can significantly influence a comparison between two computer architectures, the choice of logic mapping tools could also exert an influence on the results presented in section 3.2. In order to examine this effect, a subset[8] of the benchmarks were mapped using both MIS-pga[26] and Chortle-CRF[25], two lookup table mappers which use very different algorithms. The results are shown in

[8]It was only possible to process a subset of the benchmarks through MIS-pga as the program core dumped on the larger benchmarks.

Figures 43 and 44, along with the results of Chortle-CRF on the same subset. Both programs produce similar results in terms of the number of cells versus K, but this is not surprising since ultimately both are limited by Spira's bound. Designs mapped by MIS-pga favored a slightly higher granularity, but the differences are within the confidence interval shown in Figure 10 and are not statistically significant. An older version of Chortle[24] used for some of our earlier experiments[13] did produce significantly different results. The earlier version of Chortle did not make as effective use of large granularity cells as did the later version, and as a result smaller values of $K^*$ appeared optimal. We infer from this that, as in the case of computer architecture, the hardware and the compiler (synthesis tool) must be optimized together. It also appears that the current generation of lookup table mapping tools are beginning to approach the theoretical bounds in terms of logic depth, hence we do not expect that the experimental results presented in the previous section will change significantly as a result of new developments in synthesis technology.

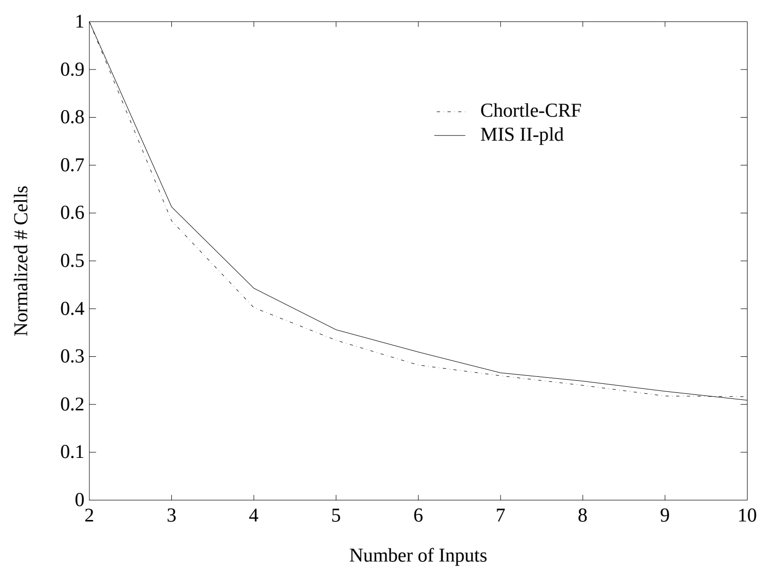


Figure 43: Number of Basic Cells vs. K for two different mapping tools. The difference between the two tools is not statistically significant

Given these observations we believe that the basic tradeoffs we have observed are basically sound, despite the limitations of our experiments. The exact numbers for the "best cell granularity" may, however, shift slightly with various tools and routing

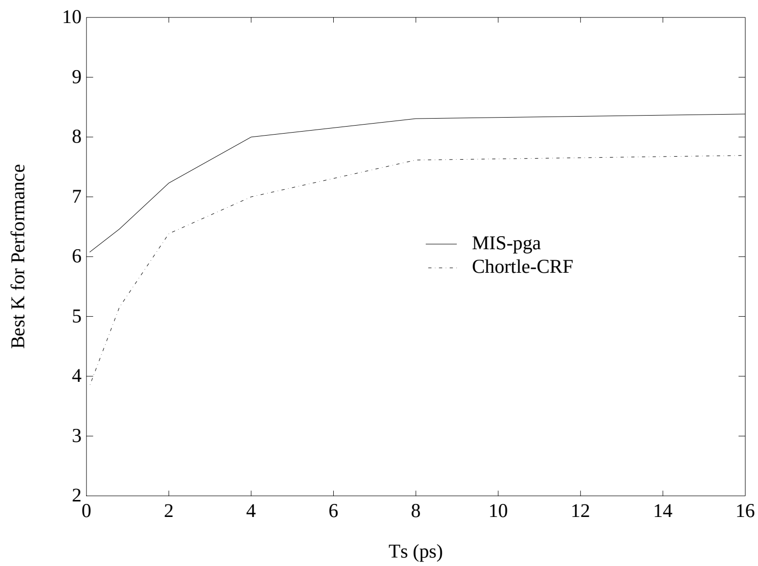


Figure 44: Best K vs. $\tau_s$. The two mapping programs produce very similar curves. MIS-pga seems to favor larger granularity, though the difference is not statistically significant.

schemes.

## 9.3. Directions for Future Work

FPGA architecture, unlike computer architecture, is still a relatively new and immature field with many areas open for exploration.

Our work has focused principally on studying one aspect of FPGA architecture, cell granularity, and its effect on performance and density for two families of logic cells, the lookup table and the PLA. The results of our experiments can also be applied to multiplexer-based cells (which are one family of CMOS functional cell based architectures), since these, like the lookup-table, are a form of Universal Logic Module. A direct extension of this work could explore other families of CMOS functional cell architectures. The principal problem here is identifying a systematic family of cells to explore and developing appropriate mapping tools for the family.

Other facets of FPGA architecture, besides cell architecture, are equally deserving of systematic study. Perhaps the most important of these is FPGA routing architecture. Brown[59] has explored routing algorithms and some of the tradeoffs involved in routing architecture. The use of hierarchy in FPGA routing architecture remains a largely unexplored area. Many FPGAs provide two levels of routing, local and global. The number of levels in this hierarchy, the allocation of routing resources to each level in the hierarchy, and the utility of hierarchy in increasing performance and density are all factors that will become increasingly important as FPGAs grow in size.

Our experiments have looked at optimizing FPGAs for general applications; consequently our benchmarks were selected from a wide range of applications. The FPGA Cell Architecture Workbench and its associated experimental methodology could also be used to optimize cell designs for specific application areas such as digital signal processing, cellular automata, neural networks, and the like. This could be done by repeating the experiments using benchmarks selected only from the target applications. Some initial work has been done in the area of digital signal processing[60]. One particularly exciting application area, likely to have very different requirements, is the use of FPGAs as reconfigurable, special purpose computing accelerators for workstations[61, 62, 63].

In all of these areas of future work the complexity and interaction of the various facets of FPGA architecture and the target applications will necessitate the use of experimental techniques such as the ones described here.

**Acknowledgements.**

We would like to thank Jonathan Rose of the University of Toronto for many helpful discussions and feedback. We would also like to thank Steve Trimberger of Xilinx, Clive McCarthy of Altera, and David Marple of Crosspoint for technical information on their company's products as well as for useful discussions and criticism. Giovanni DeMicheli was an active participant in our FPGA research and provided us with much useful insight into the CAD issues involved. We would also like to thank Dana How for his help with improving the presentation.

The FPGA Cell Architecture Workbench depends heavily on tools developed at Stanford, Berkeley, and the University of Toronto. We would like to thank Bob Francis for supplying us with the Chortle-CRF[25] technology mapping program, Frederic Mailhot for enhancements to his Ceres [20] and Mercury[21] tools, and the team at U.C. Berkeley for providing us with misII-pld[26]. Mentor Graphics (Silicon Design Division) generously provided us with their GDT software.

One of the authors, Jack Kouloheris, gratefully acknowledges the support of the IBM Corporation through the IBM Resident Study Program. This research was partially supported by DARPA grant J-FBI-89-101.

**Appendix A – Proofs of Propositions 1 and 2.**

In order to derive Propositions 1 and 2 we must minimize

$$f(n_1, n_2, \ldots, n_L) = \alpha^{n_1} + \alpha^{n_2} + \ldots + \alpha^{n_L} \tag{.1}$$

subject to

$$\sum_{i=1}^{L} n_i = N \tag{.2}$$

$$n_L = 1 \tag{.3}$$

$$n_i \geq 1 \tag{.4}$$

$$n_{i-1} \leq K n_i. \tag{.5}$$

for two cases. First, we consider the case with unbounded fanin ($K = \infty$), and secondly, we show how this solution is modified when $K$ is finite. This is a constrained non-linear programming problem which we may solve using Lagrange multipliers and Kuhn-Tucker conditions.

**Kuhn-Tucker Conditions.** *Let* $\mathbf{x}^*$ *be a relative minimum point for the problem*

$$\begin{array}{ll} \textit{minimize} & f(\mathbf{x}) \\ \textit{subject to} & \mathbf{h}(x) = \mathbf{0}, \qquad \mathbf{g}(x) \leq \mathbf{0} \end{array}$$

*and suppose* $\mathbf{x}^*$ *is a regular point for the constraints. Then there is a vector* $\boldsymbol{\lambda}$ *and a vector* $\boldsymbol{\mu} \geq \mathbf{0}$ *such that*

$$\begin{aligned} \nabla \mathbf{f}(\mathbf{x}^*) + \boldsymbol{\lambda} \nabla \mathbf{h}(\mathbf{x}^*) + \boldsymbol{\mu} \nabla \mathbf{g}(\mathbf{x}^*) &= \mathbf{0} \\ \boldsymbol{\mu} \mathbf{g}(\mathbf{x}^*) &= \mathbf{0}. \end{aligned}$$

In our case we have

$$(\nabla \left[ \sum_{i=1}^{L} \alpha^{n_i} \right] + \lambda_1 \nabla \left[ \left( \sum_{i=1}^{L} n_i \right) - N \right] + \lambda_2 \nabla [n_L - 1] + \mu_i \nabla [n_{i-1} - K n_i] = 0 \qquad (.6)$$

$$\mu_i (n_{i-1} - K n_i) = 0. \qquad (.7)$$

**Proposition 1.** *Of all possible arrangements of N cells into L levels, the arrangement with one cell on level L and* $\frac{N-1}{L-1}$ *cells on each of the remaining levels results in the maximum* $E[D_{crit}]$ *(see figure 27(a)).*

Proof.

With K unbounded, we may set the inequality constraints inactive (i.e., the $\mu_i$ in (.6) and (.7) are all zero). Differentiating (.6) with respect to the $n_i$, $\lambda_1$, and $\lambda_2$ we have

$$\begin{aligned} n_1 \alpha^{n_1 - 1} + \lambda_1 &= 0 \\ n_2 \alpha^{n_2 - 1} + \lambda_1 &= 0 \\ &\vdots \\ n_{L-1} \alpha^{n_{L-1} - 1} + \lambda_1 &= 0 \\ n_L \alpha^{n_L - 1} + \lambda_1 + \lambda_2 &= 0 \\ \sum_{i=1}^{L} n_i &= N \\ n_L &= 1 \\ \Rightarrow n_1 = n_2 = n_3 = \cdots = n_{L-1} & \\ \Rightarrow n_{1,\ldots,L-1} &= \frac{N-1}{L-1} \square \end{aligned}$$

**Proposition 2.** *Of all possible arrangements of $N$ cells (each with a maximum of $K$ inputs) into $L$ levels, the arrangement of an equal number of cells, $m$, on levels $1, \ldots, j$, and $K^{L-i}$ cells on each level $i$, $i = j+1, \ldots, L$, where*

$$m = \frac{N - \frac{1-K^{(L-j)}}{1-K}}{j}$$

*and $j$ is chosen to be the largest integer that satisfies*

$$m \geq K^{L-j},$$

*results in the largest $E[D_{crit}]$ (see Figure 27(b)).*

Proof.

We use the same method as above, but this time we consider the case where the inequality constraints are active on levels $j+1, \ldots, L$. We have

$$\begin{aligned}
n_1 \alpha^{n_1 - 1} + \lambda_1 + \mu_2 &= 0 \\
n_2 \alpha^{n_2 - 1} + \lambda_1 - K\mu_2 + \mu_1 &= 0 \\
&\vdots \\
n_{L-1} \alpha^{n_{L-1} - 1} + \lambda_1 - K\mu_{L-1} + \mu_{L-2} &= 0 \\
n_L \alpha^{n_L - 1} + \lambda_1 + \lambda_2 - K\mu_L + \mu_{L-1} &= 0 \\
\sum_{i=1}^{L} n_i &= N \\
n_L &= 1 \\
\mu_i [n_{i-1} - K n_i] &= 0 \\
\mu_i &= 0 \qquad (1 \leq i \leq j) \\
\mu_i &> 0 \qquad (j < i \leq L) \\
\Rightarrow n_1 = n_2 = n_3 = \cdots = n_{j-1} & \\
\Rightarrow n_{i-1} &= K n_i \qquad (j < i \leq L) \\
\text{so} & \\
n_{1,\ldots,j-1} &= \frac{N - \sum_{i=j+1}^{L} K^{L-i}}{j} \\
&= \frac{N - \frac{1-K^{L-j}}{1-K}}{j}
\end{aligned}$$

Note that the inequality constraints must be active on all levels $j+1, \ldots, L$ or else the above equations do not have a solution with $\mu_i \geq 0$ for all $0 \leq \alpha \leq 1$. Also note that this solution is a global minimum (and hence a global maximum for $E[D_{crit}]$) as the Lagrangian is convex. □

**References.**


[1] Jonathan Rose, Abbas El Gamal, and Alberto Sangiovanni-Vincentelli, "Architecture of Field Programmable Gate Arrays," *Proceedings of the IEEE*, August 1993.

[2] David P. Marple and Larry Cooke, "An MPGA Compatible FPGA Architecture," in *FPGA '92 Conference Proceedings*, pp. 39–44, February 1992.

[3] William Carter *et al.*, "A User Programmable Reconfigurable Gate Array," in *Proceedings of the 1986 Custom Integrated Circuits Conference*, pp. 233–235, 1986.

[4] H. Hsieh *et al.*, "A Second Generation User Programmable Gate Array," in *Proceedings of the 1987 Custom Integrated Circuits Conference*, pp. 515–521, 1987.

[5] H. Hsieh *et al.*, "A 9000-Gate User Programmable Gate Array," in *Proceedings of the 1988 Custom Integrated Circuits Conference*, pp. 15.3.1–7, 1988.

[6] H. Hsieh *et al.*, "Third Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," in *Proceedings of the 1990 Custom Integrated Circuits Conference*, pp. 31.2.1–7, 1990.

[7] Abbas El Gamal *et al.*, "An Architecture for Electrically Configurable Gate Arrays," in *Proceedings of the 1988 Custom Integrated Circuits Conference*, pp. 15.4.1–4, 1988.

[8] J. Birkner *et al.*, "A Very High Speed Field Programmable Gate Array Using Metal-to-Metal Antifuse Programmable Elements," in *New Hardware Product Introduction, 1991 Custom Integrated Circuits Conference (not in proceedings)*, 1991.

[9] S. C. Wong *et al.*, "A 5000-gate CMOS EPLD with Multiple Logic and Interconnect Arrays," in *Proceedings of the 1989 Custom Integrated Circuits Conference*, pp. 5.8.1–4, 1989.

[10] Cecil H. Kaplinsky, "Programmable Logic Device." U.S. Patent 4,847,612, July 11, 1989.

[11] Jonathan Rose *et al.*, "The Effect of Logic Block Functionality on Area of Programmable Gate Arrays," in *Proceedings of the 1989 Custom Integrated Circuits Conference*, pp. 5.3.1–5, 1989.

[12] Jonathan Rose *et al.*, "Architecture of Field Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, vol. 25, pp. 1217–1225, October 1990.

[13] Jack L. Kouloheris and Abbas El Gamal, "FPGA Performance vs. Cell Granularity," in *Proceedings of the 1991 Custom Integrated Circuits Conference*, pp. 6.2.1–4, 1991.

[14] Satwant Singh *et al.*, "Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed," in *Proceedings of the 1991 Custom Integrated Circuits Conference*, pp. 6.1.1–6, 1991.

[15] Jack L. Kouloheris and Abbas El Gamal, "FPGA Area vs. Cell Granularity – Lookup Tables and PLA Cells," in *FPGA '92 Workshop Notes*, pp. 9–14, 1992.

[16] J. L. Kouloheris and A. El Gamal, "PLA-Based FPGA Area versus Cell Granularity," in *Proceedings of the 1992 Custom Integrated Circuits Conference*, pp. 4.3.1–4, 1992.

[17] Chad L. Mitchell and Michael J. Flynn, "A Workbench for Computer Architects," *IEEE Design and Test of Computers*, vol. 5, pp. 19–29, February 1988.

[18] B. Bray, K. Cuderman, M. Flynn, and A. Zimmerman, "The Computer Architect's Workbench," in *Information Processing 89*, pp. 509–514, 1989.

[19] D. C. Ku and G. De Micheli, *Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[20] F. Mailhot and G. De Micheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," in *Proc. European Design Automation Conference*, pp. 212–216, March 1990.

[21] G. De Micheli, D. Ku, F. Mailhot, and T. Truong, "The Olympus Synthesis System," *IEEE Design & Test of Computers*, vol. 7, pp. 37–53, October 1990.

[22] R. K. Brayton *et al.*, "MIS: A Multi-level Logic Optimization System," *IEEE Transactions on CAD*, vol. 6, pp. 1062–1081, September 1987.

[23] Ewald Detjens *et al.*, "Technology Mapping in MIS," in *Proceedings of the IEEE International Conference on CAD*, pp. 116–199, November 1987.

[24] R. J. Francis, Jonathan Rose, and Zvonko Vranesic, "Chortle: A Technology Mapping Program for Lookup Table-Based FPGAs," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 613–619, June 1990.

[25] R. J. Francis, Jonathan Rose, and Zvonko Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 227–233, June 1991.

[26] R. Murgai *et al.*, "Logic Synthesis for Programmable Gate Arrays," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 620–625, June 1990.

[27] R. Murgai *et al.*, "Improved Logic Synthesis for Table Look Up Architectures," in *1991 IEEE International Conference on Computer-Aided Design*, pp. 564–567, November 1991.

[28] D. Filo, J.C. Yang, F. Mailhot, and G.D. De Micheli, "Technology Mapping for a Two-Output RAM-based Field-Programmable Gate Array," in *European Design Automation Conference*, pp. 534–538, 1991.

[29] Jack L. Kouloheris, *An Empirical Study of the Effect of Cell Granularity on FPGA Density and Performance*. PhD thesis, Stanford University, 1993.

[30] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 671, 1983.

[31] Ronald L. Rivest and Charles M. Fidducia, "A "Greedy" Channel Router," in *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 418–424, March 1982.

[32] SAS Institute, Cary, N.C., *SAS User's Guide: Statistics*, 1985.

[33] Barbara F. Ryan *et al.*, *Minitab Handbook*. PWS-Kent Publishing Co., 2nd ed., 1992.

[34] Abbas El Gamal, Jonathan Greene, and Vwani Roychowdhury, "Segmented Channel Routing is Nearly as Efficient as Channel Routing (and Just as Hard)," in *Proceedings of the 1991 U. C. Santa Cruz Conference on Advanced Research in VLSI*, March 1991.

[35] John A. Rice, *Mathematical Statistics and Data Analysis.* Wadsworth & Brooks Cole, 1988.

[36] Raj Jain, *The Art of Computer Systems Performance Analysis.* Wiley, 1991.

[37] Ken Easton, Sergio Golombek, and Sudhakar Ravi, "FPGA Hardware Project, Final Report." June 1990.

[38] Michael P. Brassington, Monir H El-Diwany, Reda R. Razouk, Michael E. Thomas, and Prateep T. Tuntasood, "An Advanced Single-Level Polysilicon Submicrometer BiCMOS Technology," *IEEE Transactions on Electron Devices*, vol. 36, pp. 712–719, April 1989.

[39] Abbas El Gamal, Jonathan Greene, and Vwani Roychowdhury, "Segmented Channel Routing," in *Proceedings of the 27th Design Automation Conference*, pp. 567–572, June 1990.

[40] L. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Tech. Rep. ERL-520, University of California at Berkeley, May 1975.

[41] Jorge Rubinstein, Paul Penfield, Jr., and Mark Horowitz, "Signal Delay in RC Tree Networks," *IEEE Transactions on Computer Aided Design*, vol. 2, July 1983.

[42] Philip M. Spira, "Computation Times of Arithmetic and Boolean Functions in (d,r) Circuits," *IEEE Transactions on Computers*, vol. C-22, pp. 552–555, June 1973.

[43] David P. Marple, *Performance Optimization of Digital VLSI Circuits.* PhD thesis, Stanford University, 1986.

[44] John T. Robinson, "Some Analysis Techniques for Asynchronous Multiprocessor Algorithms," *IEEE Transactions on Software Engineering*, vol. SE-5, pp. 24–31, January 1979.

[45] Kenneth Ching-Yu Kung, *Concurrency in Parallel Processing Systems.* PhD thesis, UCLA, 1984.

[46] H. A. David, *Order Statistics.* Wiley, 1981.

[47] Richard E. Barlow and Frank Proschan, *Statistical Theory of Reliability and Life Testing.* Holt, Rinehart, and Winston, 1975.

[48] R. Lisanke, *Logic Synthesis and Optimization Benchmarks, User Guide.* Microelectronics Center of North Carolina, 2.0 ed., 1988.

[49] S. Trimberger, "private communication." 1991.

[50] P. Spira, "On time-hardware complexity tradeoffs for Boolean functions," in *Proc. 4th Hawaii International Conference on System Sciences*, pp. 525–527, 1971.

[51] A. D. Korshunov, "O Slozhnosti Kratchaishikh Dizyunktivnykh Normalnykh Form Bulevykh Funktsii (Complexity of the shortest disjunctive normal forms of Boolean functions)," *Metody Diskretnogo Analiza*, no. 37, pp. 9–41, 1981. (in Russian).

[52] N. Pippenger, "private communication." 1991.

[53] Walter S. Scott *et al.*, "1986 VLSI Tools: Still More Works by the Original Artists," Tech. Rep. UCB/CSD 86/272, University of California at Berkeley, December 1985.

[54] R. Hartmann, "private communication." 1991.

[55] Plessey Semiconductor, Swindon, England, *ERA60100 Preliminary Data Sheet*, 1989.

[56] Robert H. Krambeck, C. T. Chen, and R. Y. Tsui, "ORCA: A High Density FPGA Architecture," in *Compcon 93 Digest of Papers*, pp. 367–272, 1993.

[57] D. Hill and N.-S. Woo, "The Benefits of Flexibility in Lookup-Table FPGAs," in *Oxford 1991 International Workshop on Field Programmable Logic and Applications*, pp. 127–136, 1991.

[58] Actel Corporation, Sunnyvale, California, *ACT-3 Data Sheet*, 1993.

[59] Stephen Dean Brown, *Routing Algorithms and Architectures for Field-Programmable Gate Arrays.* PhD thesis, University of Toronto, 1991.

[60] M. Agerwala and P. T. Balsara, "Application Specific Logic Module Architecture for FPGAs," in *Proceedings of the 1992 Custom Integrated Circuits Conference*, pp. 4.1.1–4, 1992.

[61] T. A. Kean and J. P. Gray, "Configurable Hardare: A New Paradigm for Computation," in *Proceedings of the Decennial Caltech Conference on VLSI*, March 1989.

[62] Tom Kean, John Gray, and Bernard Pruniaux, "Implementation of Configurable Hardware Using Wafer Scale Integration," in *IEEE International Conference on Wafer Scale Integration*, pp. 68–73, 1990.

[63] P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to Programmable Active Memories," in *Systolic Array Processors*, pp. 300–309, Prentice Hall, 1989.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598