The Effect of Logic Block Architecture on FPGA Performance

Satwant Singh, *Member, IEEE*, Jonathan Rose, *Member, IEEE*, Paul Chow, *Member, IEEE*, and David Lewis, *Member, IEEE*

Abstract—**This paper explores the effect of logic block architecture on the speed of a field-programmable gate array (FPGA). Four classes of logic block architecture are investigated: NAND gates, multiplexer configurations, lookup tables, and wide-input AND-OR gates. An experimental approach is taken, in which each of a set of benchmark logic circuits is synthesized into FPGA's that use different logic blocks. The speed of the resulting FPGA implementations using each logic block is measured. While the results depend on the delay of the programmable routing, experiments indicate that five- and six-input lookup tables and certain multiplexer configurations produce the lowest total delay over realistic values of routing delay. The primary reason is that these blocks can implement typical logic using the fewest levels of logic blocks, and thus incur a small number of stages of the slow programmable routing present in all FPGA's. The secondary reason is that their inherent combinational delay is not excessive. The fine grain blocks, such as the two-input NAND gate, exhibit poor performance because these gates require many levels of logic block to implement the circuits and hence require a large routing delay.**

I. Introduction

THE field-programmable gate array (FPGA) is a new ASIC medium that provides instant manufacturing turnaround and extremely low prototype manufacturing costs. An FPGA can be designed like a mask-programmed gate array (MPGA) but is user-programmable like a programmable logic device (PLD). The user-programmability, however, causes an FPGA to have both lower logic density and lower performance than an MPGA that is made in the same process technology. These deficiencies can be addressed by improving the architecture of the FPGA, which consists of its logic block function, interconnection structure, and I/O block design. In previous work, we have investigated the effect of logic block functionality on the area-efficiency of FPGA's [22], and the effect of switching block flexibility on the routability of an FPGA [23]. In this paper we look at the effect of logic block architecture on FPGA performance.

The FPGA was introduced in [8]. Since then newer versions have been introduced [14], [15] and several other types have been implemented [1]–[5], [9], [11], [19]–[21], [26]. An FPGA consists of an array of logic blocks surrounded by a programmable interconnection structure. There are many different kinds of interconnection structures, such as those articulated in these commercial architectures and in [23]. It is universally true, however, that the delay of the routing is significantly greater than that of a simple metal wire in the same process technology because programmable interconnects contain significant resistance and capacitance. For example, in [15] connections are made with pass transistors with 1- to 2-kΩ resistance, and in [1] connections are made with 300- to 500-Ω antifuses. As a result, connection delays often exceed the delay of the logic block, and this is one of the fundamental limitations on FPGA speed.

The performance of an FPGA can be increased by reducing the number of stages of programmable routing used in the critical paths. One way to do this is to use logic blocks with high functionality so that the number of logic block levels in the critical path is minimized, as illustrated in Fig. 1. Fig. 1(a) gives the implementation of the logic function $f = ab\bar{d} + abc + ac\bar{d}$ using a two-input NAND gate as the logic block. It requires four levels of the logic block in the critical path. Fig. 1(b) shows an implementation of the same function using three-input lookup tables, which requires only two levels. Since the latter avoids two levels of slow programmable interconnect, this will likely lead to a significant decrease in delay. Increasing the functionality of the logic block, however, is likely to increase its combinational delay. This increase is only profitable if the reduction in routing delay more than offsets the increase in total delay due to the logic block.

In this paper, an empirical approach is taken to study the effect of the logic block functionality on the total delay of an FPGA. We seek a logic block that minimizes the total delay in an FPGA. The experiments presented in this paper indicate that five- and six-input lookup tables exhibit the lowest total delay over a set of logic circuits for the important values of routing delay, and the multiplexer-based block used in [11] is close behind. The NAND gates, on the other hand, give the largest total delay, while the delay of FPGA's based on wide-input AND-OR gates is between these two.

Manuscript received August 8, 1991; revised November 4, 1991. This work was supported by NSERC under Operating Grants URF0043298, A4029, and OGP0036648, a MICRONET research grant, a research grant from Bell-Northern Research, and ITRC.

S. Singh was with the Department of Electrical Engineering, University of Toronto, Toronto, Ont., Canada M5S 1A4. He is now with AT&T Bell Laboratories, Allentown, PA 18103.

J. Rose, P. Chow, and D. Lewis are with the Department of Electrical Engineering, University of Toronto, Toronto, Ont., Canada M5S 1A4.

IEEE Log Number 9105612.

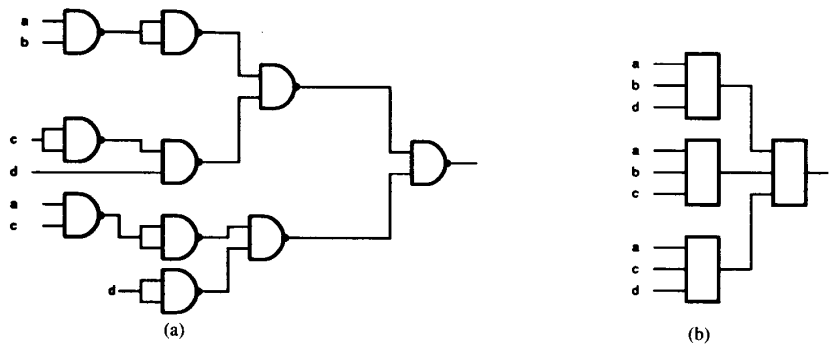


Fig. 1. Two implementations of $f = ab\bar{d} + abc + ac\bar{d}$. (a) Using two-input NAND block; number of blocks in critical path = 4. (b) Using three-input lookup table block; number of blocks in critical path = 2.

There is one major caveat in these experiments; the answers depend heavily on the quality of the logic synthesis tools used to generate them. In all cases the best tools available were used; this does not preclude the possibility that a better tool for a given logic block would improve that block's result.

This paper is concerned only with the speed of the FPGA and so we ignore issues that affect the logic density. While area has an indirect effect on speed, the assumption is that CAD tools can be optimized to reduce that effect. Previous work [22] has addressed the issue of density. An earlier version of this research appeared in [24], and an extended version appears in [25]. A similar study, which focuses on lookup tables and performs in-depth implementations to the full place-and-route level, appears in [16].

This paper is organized as follows. Section II describes the selection of logic blocks investigated, the experimental procedure, and the delay model. Section III presents the experimental results, while Section IV outlines the conclusions and the relevant future work.

II. Experimental Choices, Process, and Model

To compare the different logic blocks for their effect on the speed of an FPGA, our approach is to *synthesize* a set of circuits into many FPGA's. Each circuit is synthesized into a number of different FPGA's, where each FPGA uses a different basic logic block. The delay of the resulting implementation is then measured. The results, summarized by logic block, give an indication of the effect of logic block choice on the speed of an FPGA.

By "synthesize" we mean that a circuit passes through the logic synthesis necessary to transform a logic description of the circuit into an optimized network consisting of connections between one kind of logic block. By "measuring" the delay of the synthesized network, we mean the determination of the critical path length in each FPGA implementation and then estimating the total delay using a model.

The following section discusses the selection of logic blocks used in these experiments. Subsequent sections relate the synthesis procedure and the delay modeling.

A. Logic Block Selection

To represent a wide cross section of the possible blocks, four classes of the logic block were selected for comparison: NAND gates, multiplexers, lookup tables, and wide-input AND-OR gates. Table I gives the name and description of the logic blocks chosen from each class. The four classes are described below.

1) NAND Gates: The nand2, and nand3, and nand4 gates are two-, three- and four-input NAND gates, respectively. The nand2pi, nand3pi, and nand4pi are the corresponding NAND gates that have a *programmable inversion* capability, which allows the inputs to be passed in true or complement form. These were chosen because several FPGA's have been proposed which use NAND or AND gates [9], [19], [20], and this is a similar level of granularity to that used in MPGA's.

The NAND gates were implemented using standard CMOS techniques [25]. The programmable inversion was performed with an inverter and a pass gate [25].

2) Multiplexers: The mux21 and mux41 logic blocks can implement all possible logic functions of a multiplexer by connecting the primary inputs and the selector inputs to either constants (0 or 1) or signals. The muxA logic block is the one used in [11], which can implement over 700 logic functions. Multiplexers are of interest because of their use in two FPGA's [5], [11] and because previous work in Universal Logic Elements [29] has indicated their ability to provide many logic functions.

The multiplexers were implemented as trees of transmission gates [25].

3) Lookup Tables: The $K2$ to $K9$ logic blocks are lookup tables with two to nine inputs. A K-input lookup table is a digital memory with 2^K bits, K address lines, and one output. If the truth table of a logic function is stored in the correct addresses of the memory, and the K address lines are used as logic block inputs, then the memory will implement the truth table. These blocks were chosen because lookup tables, which were also studied in [16] and [22], were used in one of the first FPGA's [8], [14], [15].

The lookup tables are implemented as one large multiplexer with the appropriate number of inputs connected to static memory cells [25].

TABLE I
LOGIC BLOCK SELECTION AND DELAY

Block Name	Logic Function	Delay (ns) 1.2-μm CMOS
	NAND Gates	
nand2	2-input NAND gate	0.70
nand3	3-input NAND gate	0.88
nand4	4-input NAND gate	1.08
nand2pi	2-input NAND gate with prog inv	1.26
nand3pi	3-input NAND gate with prog inv	1.42
nand4pi	4-input NAND gate with prog inv	1.62
	Multiplexers	
mux21	2-to-1 mux	1.08
mux41	4-to-1 mux	1.31
muxA	logic block from [11]	1.31
	Lookup Tables	
*K*2	2-input 1-output lookup table	1.39
*K*3	3-input 1-output lookup table	1.44
*K*4	4-input 1-output lookup table	1.71
*K*5	5-input 1-output lookup table	2.03
*K*6	6-input 1-output lookup table	2.38
*K*7	7-input 1-output lookup table	2.85
*K*8	8-input 1-output lookup table	3.26
*K*9	9-input 1-output lookup table	3.78
	AND-OR Gates	
A2O3pi	OR of 3, 2-input product terms	1.88
A4O3pi	OR of 3, 4-input product terms	2.17
A8O3pi	OR of 3, 8-input product terms	2.69
A16O3pi	OR of 3, 16-input product terms	3.77
A32O3pi	OR of 3, 32-input product terms	5.98
A2O5pi	OR of 5, 2-input product terms	1.98
A4O5pi	OR of 5, 4-input product terms	2.27
A8O5pi	OR of 5, 8-input product terms	2.80
A16O5pi	OR of 5, 16-input product terms	3.95
A32O5pi	OR of 5, 32-input product terms	6.05

4) AND-OR Gates: These gates perform a two-level AND-OR logic function. We use the notation AxOy pi to describe each gate, where x is the total number of inputs that can be selected to form y separate product terms. Each of the y product terms is ORed together in the logic block to generate the output. For example, A8O3pi has a total of eight inputs, each of which can be selected to form three separate product terms that are ORed together. These gates have the programmable inversion capability. The AND-OR gates consisting of the following values of x and y are investigated: $x = \{2, 4, 8, 16, 32\}$ and $y = \{3, 5\}$. These gates were chosen because of their wide use in the original PLD's [17], and use in the large-scale PLD's in [3], [4], [21], and [26].

The AND-OR gates are implemented as cascaded pseudo-NMOS NORs [25].

Table I also gives the worst-case delay of each logic block determined using the SPICE 2G6 circuit simulator [27], in a 1.2-μm CMOS process. The simulation includes a small buffer following the logic function, but no loading or delay due to routing.

B. Logic Synthesis Procedure

Logic synthesis is required to convert each test circuit into a network of logic blocks, while minimizing the number of logic stages between the primary inputs and the output of the circuit. The procedure employed is described below. Note that this procedure deals only with combinational circuits, as we assume that the sequential and combinational portions of the circuits have been separated.

1) Collapse the logic circuit into a two-level representation, using the MIS II [7] *collapse* command so that each output is only a function of its primary inputs. Optimize the two-level expression using Espresso [6].

2) Factor each output separately into a multilevel logic expression using the misII *decompose* command [7]. This may result in logic that is used in more than one expression, and so we say that it creates *fan-out*. We then remove this fan-out by replicating the logic expression that is the source of the fan-out. This step is necessary because most technology mapping approaches, including some of those used in step 3, do a poor job across fan-out. By removing fan-out the delay is reduced but the area is increased, and as such the results presented below are optimistic. It is possible that better CAD tools would be able to operate on networks with fan-out and achieve similar results. Note also that without fan-out some circuits, such as a parity tree, have a size that is exponential in the number of inputs. It was thus not possible to run this class of circuits in these experiments.

3) Perform the technology mapping of the Boolean network. This converts the Boolean logic expressions into a network of logic blocks. The best available technology mapping tool was used for each class of logic block, as described below.

a) NAND gates and multiplexers: For these logic blocks the technology mapping is done using the MIS 2.2 technology mapping program, which is the most recent version of the one presented in [10]. The mapper is set to optimize the critical path delay. It requires a library to be generated for each logic block that describes all possible logic functions that the block can perform. In all cases, a complete library was constructed (or obtained from other sources), including that for the muxA [11] block, which has over 700 logic functions [18]. Note that the MIS 2.2 technology mapper has the ability to handle blocks with internal reconvergent fan-out, and so is able to make good use of the multiplexers, as distinct from MIS II [10].

b) Lookup tables: The technology mapping for lookup tables is done by the Chortle-d [13] technology mapping program. Chortle-d is tuned to optimize lookup table networks for delay.

c) AND-OR gates: For these blocks, the logic synthesis in step 2 was omitted. The two-level expressions from step 1 are mapped into a balanced tree of gates and the depth of the critical path is computed as follows. Consider the general case where the logic block has p inputs to the AND section and can OR together s product terms. This gate would be referred to as Ap Os pi in our notation. If

the logic expression to be mapped has a product term with v variables in it, then the number of levels in the logic tree to implement the product term is $\lceil \log_p v \rceil$. Similarly, if the number of product terms in the expression is t, then the depth of the OR tree would be $\lceil \log_s t \rceil$. When the two trees are combined, the total depth in logic blocks is $\lceil \log_p v \rceil + \lceil \log_s t \rceil$. Note that in some cases where some variables are common in many product terms, it is possible to reduce the number of levels by one less than given by this expression. Thus, the critical path calculations are pessimistic by at most one logic level.

C. Model for Measuring Delay

The speed of a circuit implemented in an FPGA with a given logic block is a function of the combinational delay of the logic block (D_{LB}), the number of logic blocks on the critical path (N_L), and the delay incurred in the routing between each logic block (D_R). Assuming that each stage of logic block incurs one routing delay and one logic block delay, then the total delay (D_{TOT}) can be calculated as

$$D_{TOT} = N_L \times (D_{LB} + D_R). \quad (1)$$

The value of N_L can be measured for each circuit after it is mapped into a logic block using the procedure described above. The value of D_{LB} was determined as described in Section II-A.

The value of D_R is much more difficult to determine. It is a function of the routing architecture, the fan-out of a connection (which would be determined by the physical placement), the length of the connection, the process technology, and the programming technology. Since our purpose is to understand general architectural principles, it is important not to fix any of these parameters. As such, most of the results below will be given as a function of D_R, rather than choosing a specific D_R.

This assumption, however, makes the approximation that D_R is constant for each connection. This is a simplifying abstraction that makes this broad set of experiments possible, but it is inaccurate, and must be considered when any conclusions are drawn. It is comforting to note that in [16], where complete implementations down to the place-and-route level were performed, results where the experiments overlap are similar to those presented here.

III. Experimental Results

The experimental circuits that were used are a selection of 15 logic synthesis benchmarks provided by the Microelectronics Center of North Carolina (MCNC) and one standard cell-based circuit from Bell-Northern Research. They range in size from 28 to over 700 two-input NAND gate equivalents. Each circuit was passed through the implementation procedure described in Section II-B once for every logic block listed in Table I. Sections III-A through -D discuss the relative performance of the logic blocks in each of the four classes (NAND gates, multiplexers, lookup tables, and AND-OR gates). Section III-E compares the best logic blocks from the four classes.

The comparison of different logic blocks is done by averaging the critical path delays over all the test circuits.

TABLE II
AVERAGE CRITICAL PATH LENGTH AND TOTAL DELAY—NAND GATES

Logic Block	D_{LB} (ns)	$\overline{N_L}$	Std. Dev.	$D_{TOT} = N_L \times (D_{LB} + D_R)$			
				$D_R = 0$ (ns)	$D_R = 2$ (ns)	$D_R = 4$ (ns)	$D_R = 10$ (ns)
nand2	0.70	15.2	5.8	11	41	71	163
nand3	0.88	11.8	4.0	10	34	57	128
nand4	1.1	10.8	3.8	12	33	55	120
nand2pi	1.3	10.8	4.5	14	35	57	121
nand3pi	1.4	9.3	3.8	13	32	50	106
nand4pi	1.6	8.9	3.8	15	32	50	103

A. NAND Gates

Table II gives the summarized delay data for NAND gates. The first column names the gate, the second column lists the combinational delay from Table I, the third column gives the average number of logic blocks in the critical path ($\overline{N_L}$) over all 16 circuits, and the fourth column contains the standard deviation of this average. Columns five through eight give the total delay (D_{TOT}) for different values of the routing delay (D_R). The D_{TOT} calculations use the values of 0, 2, 4, and 10 ns for D_R. These values of D_R are chosen because they range from the minimum possible to the point where the ranking of the logic blocks based on D_{TOT} does not change. Typical FPGA delays, in 1.2-μm CMOS, range from 2.5 to 10 ns [28].

Table II shows that the total delay for the nand3 block, for all ranges of D_R, is less than nand2. This occurs because the increase in functionality from nand2 to nand3 results in a lowering of the number of logic blocks in the critical path ($\overline{N_L}$) from 15.2 to 11.8. While the delay of nand3 (0.88 ns) is slightly more than for the nand2 (0.70 ns), the saving in the number of levels more than compensates. Interestingly, this is true for $D_R = 0$, which would correspond to mask-programmed routing. As the routing becomes slower ($D_R > 0$), the relative performance of nand3 improves over nand2 because each routing stage costs more in delay.

The reduction in total delay from nand3 to nand4 is not as significant as it is from nand2 to nand3. This is because the increase in logic block delay is not offset by a reduction in the total number of logic block levels. Note that no further improvement was achieved with the nand5 gate or any larger NAND gates.

Table II also indicates that the addition of programmable inversion (the gates suffixed with "pi") to the NAND gate inputs causes a significant reduction in the number of logic block levels. The programmable inversion, however, requires roughly 0.5-ns extra combinational delay which makes such gates slower at $D_R = 0$, as compared to the pure NAND gates. As D_R increases, nand2pi, nand3pi, and nand4pi give increasingly better delay than nand2, nand3, and nand4, respectively. This is because the increased combinational delay is being more than offset by the reduction in the routing delay due to a lower $\overline{N_L}$. For $D_R > 0$, nand3pi and nand4pi give the best per-

formance among NAND gates. While nand3pi and nand4pi exhibit almost the same performance, nand3pi has fewer inputs and so it would be the best choice among the NAND gates. This is because the number of inputs has an indirect effect on delay, which is not considered in the delay model: as the number of inputs to a block increases, there are more connections to the block, and hence more parasitic capacitance to be driven.

B. Multiplexers

The total delay results for three multiplexer configurations are given in Table III, which has the same columns as Table II. The muxA logic block exhibits the lowest $\overline{N_L}$. This is due to the high number of logic functions that this logic block can perform, several of which have appreciable fan-in. These wider gates are capable of reducing logic depth because depth is roughly logarithmic in the number of inputs, with the base of the logarithm equal to the fan-in of the gate. The combinational delay of the muxA logic block is same as that of a four-to-one multiplexer (mux41), and because it has lower $\overline{N_L}$, it gives better performance for all values of D_R. Thus, the muxA block would be the best choice among the multiplexer configurations investigated.

TABLE III
AVERAGE CRITICAL PATH LENGTH AND TOTAL DELAY—MULTIPLEXERS

Logic Block	D_{LB} (ns)	$\overline{N_L}$	Std. Dev.	$D_{TOT} = N_L \times (D_{LB} + D_R)$			
				$D_R = 0$ (ns)	$D_R = 2$ (ns)	$D_R = 4$ (ns)	$D_R = 10$ (ns)
mux21	1.1	9.9	4.7	11	30	50	110
mux41	1.3	6.1	2.3	8	20	33	69
muxA	1.3	4.4	2.0	6	15	23	50

TABLE IV
AVERAGE CRITICAL PATH LENGTH AND TOTAL DELAY—LOOKUP TABLES

Logic Block	D_{LB} (ns)	$\overline{N_L}$	Std. Dev. (ns)	$D_{TOT} = N_L \times (D_{LB} + D_R)$			
				$D_R = 0$ (ns)	$D_R = 2$ (ns)	$D_R = 4$ (ns)	$D_R = 10$ (ns)
*K*2	1.4	10.0	3.9	14	34	54	114
*K*3	1.4	5.6	2.1	8	19	31	64
*K*4	1.7	4.1	1.5	7	15	24	48
*K*5	2.0	3.4	1.2	7	14	21	41
*K*6	2.4	2.8	1.0	7	12	18	35
*K*7	2.9	2.4	1.0	7	12	16	31
*K*8	3.3	2.1	0.8	8	11	16	28
*K*9	3.8	2.0	0.8	8	12	16	28

C. Lookup Tables

Table IV summarizes the total delay results for K-input lookup tables, with K ranging from 2 to 9. As the number of inputs to the lookup table increases, we observe that the number of logic block levels in the critical path continues to decrease up to $K = 9$. Significant decreases are obtained as far as $K = 8$. This occurs because lookup tables can implement *any* function of K inputs, and so they can contain many levels of logic. Notice that the lookup table inherently has the programmable inversion capability.

As the number of inputs to the lookup table increases, the logic block delay (D_{LB}) increases roughly 0.4 ns for every added input, after $K = 3$. This is because each added input causes one more transistor to be added in series in the multiplexer tree that implements the lookup table.

For very fast routing delay ($D_R = 0$), the fastest logic block is strictly a function of the number of logic block levels ($\overline{N_L}$) and the delay of the logic block (D_{LB}). As shown in Table IV, for values of K greater than 2, the total delay (D_{TOT}) is almost constant. This says that reduction in delay due to a lower $\overline{N_L}$ as K increases above 3 is exactly offset by the increase in delay due to the increase in D_{LB}.

As D_R increases, the cost in delay of each logic block level increases, and so the blocks with lower values of $\overline{N_L}$ achieve superior performance. For $D_R = 2$ the six-input lookup table achieves the best performance. For $D_R = 4$ the seven-input lookup table achieves the best performance. Notice, however, that the five-input lookup table achieves similar performance in both cases, and the accuracy of these experiments makes this small spread insignificant. In addition, as noted in Section III-A, the delay model does not account for the increase in delay due to extra capacitive loading from the higher number of pins, and so the marginal improvement shown in Table IV from $K = 5$ to $K = 7$ may be lost. The actual choice of logic block might be more strongly influenced by the fact that each added input doubles the number of bits in the lookup table, and hence the area. Thus, the five- and six-input lookup tables are good choices for $D_R = 2$ and $D_R = 4$ ns, which are realistic values for routing delay.

As D_R increases to 10 ns, the best value of K continues to increase, to $K = 8$. It is clear that as long as increasing K results in a decrease in $\overline{N_L}$, then higher values of K will be faster for higher values of D_R.

D. AND-OR Gates

Table V gives the total delay for the wide AND-OR gates. It is clear that for all ranges of the routing delay, the AND-OR blocks with five product terms (A*x*O5pi) exhibit lower total delay than the corresponding blocks with three product terms (A*x*O3pi). There is an average of 10 to 15% improvement in delay from three to five product terms. This occurs because the blocks with five product terms have smaller $\overline{N_L}$ than those with three product terms, while the increase in D_{LB} from three to five product terms is minor.

For fast routing, with D_R up to 2 ns, the A4O5pi block gives the lowest delay as this logic block presents a good balance between combinational delay and functionality—it has 26% fewer average logic block levels than A2O5pi, and only 15% more combinational delay. Blocks with more than four inputs incur a combinational delay that more than offsets the gain due to a reduction in $\overline{N_L}$, at the lower values of routing delay.

TABLE V
AVERAGE CRITICAL PATH LENGTH AND TOTAL DELAY—AND-OR GATES

Logic Block	D_{LB} (ns)	$\overline{N_L}$	Std. Dev.	$D_{TOT} = N_L \times (D_{LB} + D_R)$ $D_R = 0$ (ns)	$D_R = 2$ (ns)	$D_R = 4$ (ns)	$D_R = 10$ (ns)
A2O3pi	1.9	7.4	1.9	14	29	43	88
A4O3pi	2.2	5.6	1.5	12	24	35	69
A8O3pi	2.7	5.0	1.6	13	23	33	63
A16O3pi	3.8	4.6	1.5	18	27	36	64
A32O3pi	6.0	4.3	1.3	25	34	42	68
A2O5pi	2.0	6.5	1.6	13	26	39	78
A4O5pi	2.3	4.8	1.2	11	20	30	58
A8O5pi	2.8	4.1	1.2	12	20	28	53
A16O5pi	4.0	3.8	1.2	15	22	30	52
A32O5pi	6.1	3.4	1.0	20	27	34	54

For $D_R = 4$ ns, the A8O5pi block exhibits the lowest delay, while for $D_R = 10$ ns, the A16O5pi block is the fastest. As the routing delay increases, the effect of $\overline{N_L}$ dominates the total delay since the routing delay per stage is much greater than the combinational delay per stage. However, as discussed in Section III-A, the delay model in this study gives an advantage to logic blocks with a large number of inputs, and so marginal decreases in delay resulting from a larger number of inputs should be ignored. Thus, the A4O5pi and A8O5pi blocks provide the best performance over the realistic values of routing delay. These two blocks will be compared against the best from other categories in the next subsection.

E. Overall Comparison

Fig. 2 is a plot for the total delay of the best logic blocks from each class versus the routing delay D_R. Table VI tabulates the same data. The first clear conclusion from these data is that the fine-grain logic blocks, such as the two-input and three-input NAND gates (even with programmable inversion), exhibit markedly lower performance than any other class of logic block. This is a significant conclusion, given that several commercial FPGA's use the two-input NAND gate as the basic logic block. Notice that the result is true even for a routing delay of zero, which provides an interesting perspective on mask-programmed architectures—they should perhaps use a more coarse-grain basic block, as suggested in [12].

At zero routing delay, the muxA logic block is the fastest because it has a very small combinational delay combined with a low number of logic block levels.

For the midrange routing delays (2 ns $\leq D_R \leq$ 4 ns) the five- and six-input lookup tables and the muxA logic block exhibit similar delays, with the lookup tables slightly faster. At this point the routing delay is mostly greater than the logic block delay, so the number of logic block levels begins to dominate in the comparison. These blocks have quite low values of $\overline{N_L}$. The wide AND-OR gates, which have $\overline{N_L}$ close to the muxA block, exhibit worse performance because of a significantly higher combinational delay.

For large delays ($D_R = 10$ ns) the five- and six-input lookup tables are significantly faster. This is because in this delay range the only important factor is the number of logic levels, and as Table VI shows, the lookup tables have significantly lower values of $\overline{N_L}$. Notice that the wide AND-OR gates do not approach this level. It is possible, however, that the conservative approximation described in Section II-B unfairly disadvantages these blocks.

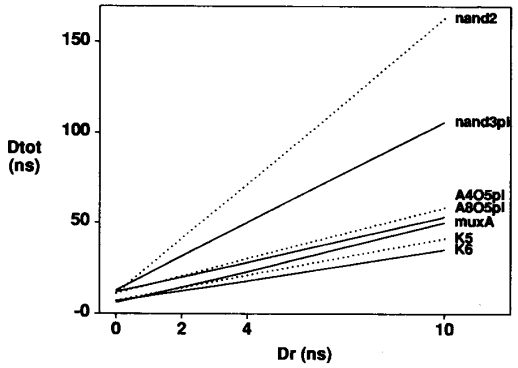


Fig. 2. D_{TOT} versus D_R for the best blocks in each class.

TABLE VI
AVERAGE CRITICAL PATH LENGTH AND TOTAL DELAY—OVERALL COMPARISON

Logic Block	D_{LB} (ns)	$\overline{N_L}$	Std. Dev.	$D_{TOT} = N_L \times (D_{LB} + D_R)$ $D_R = 0$ (ns)	$D_R = 2$ (ns)	$D_R = 4$ (ns)	$D_R = 10$ (ns)
nand2	0.70	15.2	5.8	11	41	71	163
nand3pi	1.4	9.3	3.8	13	32	50	106
A4O5pi	2.3	4.8	1.2	11	20	30	58
A8O5pi	2.8	4.1	1.2	12	20	28	53
muxA	1.3	4.4	2.0	6	15	23	50
K5	2.0	3.4	1.2	7	14	21	41
K6	2.4	2.8	1.0	7	12	18	35

F. Limitations of Results

It should be noted that these results depend heavily on the quality of the logic synthesis tools. We have observed shifts in these results by moving from technology mappers that optimize for area to those that optimize for delay. In these experiments we have used the best mapping tools available to us.

Another limitation is the approximation of D_R as a constant, as discussed in Section II-C. While this value will certainly vary, even within one connection, one would expect it to change in similar ways for each block. The conclusions presented above would likely not change in a significant way if the variation is taken into account, as shown by the fact that the results in [16] for lookup tables are similar to those presented here.

IV. CONCLUSIONS AND FUTURE WORK

This paper has explored the relationship between logic block architecture and the speed of the resulting FPGA. There are two principal conclusions:

1) five- and six-input lookup tables and the muxA [11] logic block are all good choices for a logic block for midrange values of programmable routing delay;

2) fine-grain logic blocks, such as two-input NAND gates, result in significantly worse delay (by a factor of more than 3) than these blocks.

In addition, wide AND-OR gates do not achieve comparable performance to the best blocks, but it is possible that better logic synthesis for these blocks would improve their performance.

In the future, we will adapt these experiments and tools to account for the area cost of the gain in delay. In addition we will explore the performance gains possible when hard-wired (fast) links between basic logic blocks are used.

REFERENCES

[1] M. Ahrens *et al.* "An FPGA family optimized for high densities and reduced routing delay," in *Proc. CICC*, May 1990, pp. 31.5.1-31.5.4.

[2] CAL 1024 Datasheet, Algotronix Ltd., Edinburgh, Scotland, 1989.

[3] *Mach Devices High Density EE Programmable Logic Data Book*, Advanced Micro Devices, Sunnyvale, CA, 1990.

[4] S. Baker, "Lattice fields FPGA," *Electron. Eng. Times*, no. 645, p. 1, June 10, 1991.

[5] J. Birkner *et al.*, "A very high-speed field programmable gate array using metal-to-metal anti-fuse programmable elements," New Hardware Product Introduction at CICC '91.

[6] R. K. Brayton *et al.*, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA: Kluwer Academic, 1984.

[7] R. K. Brayton *et al.*, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 6, pp. 1062-1081, Nov. 1987.

[8] W. Carter *et al.*, "A user programmable reconfigurable gate array," in *Proc. CICC*, May 1986, pp. 233-235.

[9] Concurrent Logic CFA6006 Field-Programmable Gate Array Data Sheet, Concurrent Logic Inc., Sunnyvale, CA, 1991.

[10] E. Detjens *et al.*, "Technology mapping in MIS," in *Proc. ICCAD*, Nov. 1987, pp. 116-119.

[11] A. El Gamal *et al.*, "An architecture for electrically configurable gate arrays," *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 394-398, Apr. 1989.

[12] A. El Gamal, J. Kouloheris, D. How, and M. Morf, "BiNMOS: A basic cell for BiCMOS sea-of-gates," in *Proc. CICC*, May 1989, pp. 8.3.1-8.3.4.

[13] R. J. Francis, J. Rose, and Z. Vranesic "Technology mapping of lookup table-based FPGAs for performance," in *Proc. ICCAD '91*, Nov. 1991, pp. 568-571.

[14] H. Hsieh *et al.*, "A 9000-gate user-programmable gate array," in *Proc. CICC*, May 1988, pp. 15.3.1-15.3.7.

[15] H. Hsieh *et al.*, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proc. CICC*, May 1990, pp. 31.2.1-31.2.7.

[16] J. Kouloheris and A. El Gamal "FPGA performance vs. cell granularity," in *Proc. CICC*, May 1991, pp. 6.2.1-6.2.4.

[17] P. K. Lala, *Digital System Design Using Programmable Logic Devices*, E. J. McCluskey, Ed. Englewood Cliffs, NJ: Prentice Hall, 1990.

[18] F. Mailhot, private communication, 1991.

[19] H. Muroga *et al.*, "A large scale FPGA with 10K core cells with CMOS 0.8 μm 3-layered metal process," in *Proc. CICC*, May 1991, pp. 6.4.1-6.4.4.

[20] Plessey Semiconductor ERA60100 Data Sheet, Swindon, England, 1989.

[21] Plus Logic FPGA2040 Field-Programmable Gate Array Data Sheet, Plus Logic, San Jose, CA, 1989.

[22] J. S. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1217-1225, Oct. 1990.

[23] J. S. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277-282, Mar. 1991.

[24] S. Singh, J. Rose, D. Lewis, K. Chung, and P. Chow "Optimization of field-programmable gate array logic block architecture for speed," in *Proc. CICC*, May 1991, pp. 6.1.1-6.1.6.

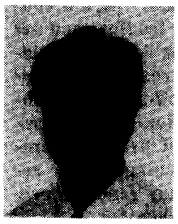
[25] S. Singh, "The effect of logic block architecture on the speed of field-programmable gate arrays," M.A.Sc. thesis, Dept. Elec. Eng., Univ. of Toronto, Toronto, Ont., Canada, Aug. 1991.

[26] S. C. Wong, H. C. So, J. H. Ou, and J. Costello, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proc. CICC*, May 1989, pp. 5.8.1-5.8.4.

[27] A. Vladimirescu, K. Zhang, A. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE Version 2G, User's Guide*, Dept. Elec. Eng., Univ. of California, Berkeley, Aug. 1981.

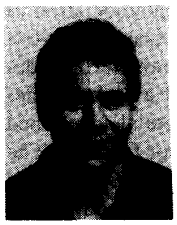
[28] J.-M. Vuillamy, "Performance enhancement in field-programmable gate arrays," M.A.Sc. thesis, Dept. Elec. Eng., Univ. of Toronto, Toronto, Ont., Canada, Apr. 1991.

[29] S. Yau and C. Tang, "Universal logic modules and their application," *IEEE Trans. Comput.*, vol. C-19, pp. 141-149, 1970.



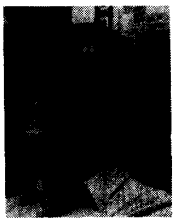
Satwant Singh (S'89-M'91) received the B.E. degree with honors in electronics and communications engineering from Guru Nanak Engineering College, Panjab University, India in 1987 and the M.A.Sc. degree in electrical engineering from the University of Toronto, Toronto, Canada, in 1991.

From 1987 to 1989 he was with Semiconductor Complex Ltd., Chandigarh, India, where he designed custom CMOS VLSI circuits. In the summer of 1991 he was with Bell-Northern Research Ltd., Ottawa, Canada, in the Merchant ASIC Engineering group. He is presently employed with AT&T Bell Laboratories, Allentown, PA, as a Member of the Technical Staff in FPGA Development Group. His research interests include field-programmable gate array architectures and applications, VLSI systems design, circuit design, and CAD tools.



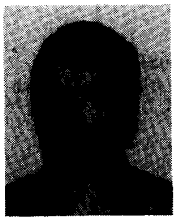
Jonathan Rose (S'79-M'86) received the B.A.Sc. degree in engineering science in 1980, and the M.A.Sc. and Ph.D. degrees in electrical engineering in 1982 and 1986, respectively, from the University of Toronto, Toronto, Canada.

During the summer of 1983 he was with Bell-Northern Research Ltd., Ottawa, Canada, in the Integrated Circuits CAD/CAM group. From 1986 to 1989 he was a Research Associate in the Computer Systems Laboratory at Stanford University, Stanford, CA. In 1989 he joined the faculty of the University of Toronto, where he is currently an Assistant Professor of Electrical Engineering. His research interests include CAD and architecture for field-programmable gate arrays, automatic layout, and parallel CAD algorithms.



Paul Chow (S'79-M'83) received the B.A.Sc. degree with honors in engineering science, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1977, 1979 and 1984, respectively.

In 1984 he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Postdoctoral Fellow and later as a Research Associate. He was a major contributor to the MIPS-X project. Since January 1988 he has been an Assistant Professor in the Department of Electrical Engineering at the University of Toronto. His current research interests include high-performance computer architectures, VLSI systems design, and field-programmable gate array architectures and applications.



David Lewis (M'88) received the B.A.Sc. degree with honors in engineering science and the Ph.D. degree in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1977 and 1985, respectively.

From 1982 to 1985 he was employed as a Research Associate on the Hubnet project, and developed custom integrated circuits for a 50-Mb/s local area network. He has been an Assistant Professor at the University of Toronto since 1985. His research interests include logic and circuit simulation, logarithmic arithmetic, field-programmable hardware, and VLSI architecture.

Dr. Lewis is a member of the ACM.