DP-FPGA: An FPGA Architecture Optimized for Datapaths

DON CHEREPACHA and DAVID LEWIS*

Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 1A4

This paper presents a new Field-Programmable Gate Array (FPGA) architecture which reduces the density gap between FPGAs and Mask-Programmed Gate Arrays (MPGAs) for datapath oriented circuits. This is primarily achieved by operating on data as a number of identically programmed four-bit slices. The interconnection network incorporates distinct sets of resources for routing control and data signals. These features reduce circuit area by sharing programming bits among four-bit slices, reducing the total number of storage cells required.

This paper discusses the requirements of logic blocks and routing structures that can be used to implement typical circuits containing a number of regularly structured datapaths of various sizes, as well as a small number of irregularities. It proposes a specific set of logic block architectures and analyzes it empirically. Experimental results show that the block with the smallest estimated area contains the following features: a lookup table with four read ports, a dedicated carry chain using a bidirectional four-bit carry skip circuit, a four-bit register with enable and direct input capabilities, and four three-state buffers. Further estimates of implementation area predict that the area of a design's datapath can be reduced by a factor of approximately two compared to a conventional FPGA through the use of programming bit sharing.

Keywords: FPGA, VLSI, Datapath, Architecture

1. INTRODUCTION

FPGAs offer several advantages over MPGAs including lower cost on small volumes, shorter time-to-market, increased flexibility and reduced risks. However, a design implemented in an FPGA requires approximately ten times the area and is roughly three times slower than one implemented in an MPGA [1]. In this paper, we attempt to reduce these differences for a specific class of circuits by developing an architec-

ture optimized for datapaths called the Datapath-FPGA (DP-FPGA). Our primary goal is to improve density, with performance being a secondary objective. Improving the density of designs with large datapaths would make the FPGA a feasible implementation method for a larger range of digital circuits.

Commercially available FPGAs, such as those described in [2], [3] and [4], are general-purpose and were developed for applications containing varying

*Corresponding author.

amounts of datapath and control logic. As such, they are flexible and enjoy widespread usage. However, their bit-level granularity in both the logic and routing structures can be inefficient for implementing wide datapaths. In particular, they are unable to take advantage of the regular bit-slice nature which allows common resources to be shared among slices. As well, no distinction is made between data and control routing resources, even though their characteristics are quite different. In full custom designs, the density of regular structures is roughly three times that of random logic. We would like to similarly exploit the regularity of datapaths to gain area advantages over a general-purpose FPGA.

As a response to the weaknesses of early FPGAs in building datapaths, many recent FPGAs contain special structures to improve datapath density and performance. These structures include dedicated carry logic to support arithmetic functions, on-chip distributed RAM facilities and abundant flip-flops. AT&T's ORCA [5] architecture further attempts to improve its suitability for datapath applications by using logic blocks capable of processing four bits of data.

Application specific field-programmable devices focusing on digital signal processing (DSP) applications have been proposed. In [6], Agarwala introduced a DSP logic module which is particularly efficient at implementing multiplexers and adders which are the building blocks of DSP circuits. In contrast to the bit-level granularity of this architecture, Chen [7] has developed a field-programmable multiprocessor for high performance DSP applications. It operates at a word-level and uses a cluster of execution units interconnected by a configurable crossbar switch.

Unlike the preceding devices, the DP-FPGA is intended for a wide range of circuits containing a variety of datapaths, such as those found in digital signal processing, communications, circuit emulation and special-purpose processor applications. The targeted systems may contain several datapaths of various widths, and have irregularities in some bit-slices. Thus, our architecture must contain some bit-level programmability, yet take advantage of the high degree of regularity that exists in datapaths. This is accomplished using a granularity which falls between

the two DSP architectures mentioned above. Both its logic and routing resources operate on four bits of data as a unit. This medium level of granularity provides area advantages over bit-level architectures by operating on buses rather than individual wires, but is more flexible than an ALU level device.

The DP-FPGA is organized as shown in Figure 1, with separate logic and routing resources to construct the three basic components of a system; control logic, datapath and memory. Area and performance advantages can be achieved by optimizing each section for its specific function. However, the flexibility of each is significantly reduced which could lead to poor utilization of resources, increasing area and delay. In this paper, we concentrate on the datapath section which, due to its regularity, provides the potential for significant density improvements that overcome the loss of flexibility. The control section will likely resemble general-purpose architectures which are good at implementing random logic. High density blocks of RAM may be useful in large datapath designs. As we are targeting datapath intensive designs, the datapath section will occupy a large portion of the total area. Thus, the ability to optimize this section will yield an indication of the feasibility of using specialized resources.

The remainder of this paper is organized as follows. Section 2 is devoted to the development of logic block and routing architectures optimized for datapaths. In section 3, an experimental analysis of the logic block is presented. A quantitative analysis of the benefits of sharing programming bits is carried out in section 4. Finally, conclusions drawn from this work are summarized in section 5.

FIGURE 1 DP-FPGA organization.

2. DATAPATH ARCHITECTURE

The datapath section of the DP-FPGA provides an opportunity for significant density optimization due to its regularity in both logic and routing. The primary method used to exploit this regularity reduces the number of programming bits required. After introducing this idea, we discuss some issues involved in the design of logic block and routing architectures for datapaths.

2.1 Programming Bit Sharing

FPGAs have lower density than MPGAs because of the large number of configuration bits required. The goal of the DP-FPGA is to reduce this advantage, which can be partially accomplished by sharing each programming bit across a number of identically programmed datapath slices. While many programming technologies exist, the need to use a single programming bit to control several circuit elements constrains the DP-FPGA's choice to SRAM based technologies. This organization is possible in the DP-FPGA due to the regularity of datapath logic, as illustrated in the following example.

A section of datapath is illustrated in Figure 2. An implementation of this section using an FPGA whose logic blocks contain a 3-input lookup table and a flip-flop is shown in Figure 3. A total of $17N$ SRAM cells are required for the selected logic and routing connections shown. The important features to note in this diagram are that all of the lookup tables implement the same logic function and that the routing connections are identical across bit-slices. This motivates our idea of bit sharing as shown in Figure 4. A single set of 17 SRAM cells is shared by all N bit-slices, reducing the total number of SRAM cells by a factor of N . The N

FIGURE 2 Section of datapath.

FIGURE 3 Mapping to an FPGA without programming bit sharing.

distinct lookup tables have been replaced by a single table with N read ports. Each read port is constructed as a multiplexer tree, selecting one table value.

The application of bit sharing to the DP-FPGA raises the question of how many bit-slices should be controlled by each programming bit. We define N to represent this value and W to be the total width of the

FIGURE 4 Mapping to an FPGA with programming bit sharing.

datapath. We use a simple model to illustrate some of the trade-offs involved in the selection of N . A fixed cost (MC) is assigned to the memory cell and an incremental cost (CE) for each connection element. The area per connection (A) is expressed by the following equation:

$$A(N) = \frac{MC + N \times CE}{N} = \frac{MC}{N} + CE$$

This is a decreasing function of N . However, the incremental benefit decreases rapidly as MC/N becomes smaller than CE . The relative sizes of MC and CE depend on both the layout and the type of connection being made. For example, CE will differ depending on whether the connection element is a pass-transistor or multiplexer. A special case is the connection of control signals. As shown by the *Enable* signal in Figure 4, it is the control signal itself which is shared, so only one connection element is required regardless of N . This illustrates the significant savings that can be achieved by sharing control signals. It should be noted that as N increases, the area required to route connections to programming bits exceeds the area of the logic, causing this simple model to be inaccurate.

For a datapath of width W , a total of $\lceil W/N \rceil$ shared N -bit units will be needed across the width of a connection. $\lceil W/N \rceil \times N - W$ bits are *wasted* as they were not intended in the original design. There is a trade-off with wastage favouring small N and the area per connection favouring large N . Based on our estimates of datapath size and that on average $MC \approx 2\text{--}3$ CE, we have selected a value of $N = 4$. We believe that this value will take good advantage of bit sharing, while keeping the amount of wastage low and maintaining ample flexibility for logic partitioning, placement and routing. For this initial study, using $N = 4$ is sufficient to show that bit sharing has the potential to yield significant density improvements.

2.2. Logic Block Architecture

The number of possible choices of logic block is unlimited. We simplify the problem by breaking the

process into two steps. First, by examining a number of target circuits, we informally characterize the way data is manipulated and propose a parameterized set of logic blocks that can meet these requirements. In the second step, we experimentally examine all possible values of these parameters and determine optimal values for each. In this section, the basic logic block architecture is presented as part of the first phase. A description of the architectural parameters and a discussion of the experimental results is left for section 3.

The basic logic block architecture developed through our informal investigation is shown in Figure 5. It contains four bit-slices, all of which must be programmed identically since bit sharing is used. Control signals are common to all four slices. The majority of the combinational logic is implemented using a lookup table with one read port per bit-slice. The use of a lookup table allows the granularity of the logic block to be varied using a single parameter K , the number of inputs to the table per bit-slice. The value of K will be discussed in section 3.

As part of our investigation, we mapped eight target circuits into the Xilinx XC4000 series architecture. We found that the dedicated carry circuitry was utilized in an average of 36% of the CLBs. This high utilization implies that the resources available to support arithmetic logic will have a significant impact on both density and performance. For this reason, dedicated carry logic is included in each block. A carry skip scheme is used to reduce the four bit carry-in to

FIGURE 5 Basic logic block architecture.

carry-out delay to that of a single pass transistor plus buffers, as shown in Figure 6. The carry chain is bi-directional for added flexibility.

The propagate (P) and generate (G) signals are efficiently computed by decomposing a K input lookup table into two K-1 input lookup tables with shared inputs, as illustrated in Figure 7. This method is particularly area efficient as the full capabilities of the lookup table can be used. For example, an array multiplier cell can be implemented in a single datapath logic block if $K \geq 4$. This is not possible in a Xilinx CLB since the set of carry functions is restricted. Thus, two CLBs are needed (one for the partial product and one for the sum), doubling the size of the multiplier compared to the DP-FPGA implementation. When the carry chain is not used, all internal carries are set to zero to allow the lookup table output to appear as SUM_{0-3} .

We also found that the flip-flops in the Xilinx CLB were highly utilized when our benchmark circuits were implemented. Therefore, a four bit register is included in each logic block. This idea is supported

FIGURE 6 Carry chain.

FIGURE 7 Computation of P and G.

by an experimental study conducted by Rose et al. [8] who concluded that it is beneficial to include a D flip-flop in the logic block.

2.3. Routing Architecture

The routing architecture of the DP-FPGA is shown in Figure 8. In contrast to conventional FPGAs, the DP-FPGA provides separate routing resources for *data* and *control* signals. A third resource, unique to the DP-FPGA, is the *shift block*, which is included to eliminate some of the restrictions associated with the use of bit sharing. In this subsection we discuss several issues involving the basic structure of each of the

FIGURE 8 General routing architecture.

three routing resources. We have not yet undertaken a detailed experimental analysis of the routing requirements.

The application of programming bit sharing over four bits of data leads to a higher level of data routing granularity in the DP-FPGA in which all data signals are routed as four-bit buses. A single SRAM cell controls each set of connection elements for the entire bus, providing higher density data routing resources. A key feature of the data interconnection network is the presence of a strong horizontal bias due to the fact that nearly all data bus connections are made between blocks in the same row. Vertical routing is required for shifts of multiples of four bits, connections to the I/O pads, and connections required in situations where the entire bit-slice cannot be placed in the same row (a result of the array of logic blocks being fixed). The fact that there are inherently fewer bends in the pin-to-pin data connections may allow the number of tracks per channel to be reduced for a given flexibility of the channel-to-channel switch blocks, with respect to a general-purpose FPGA with a symmetric routing architecture [9].

In contrast to data signals, control signals must be routed individually. Rather than explicitly sharing control routing programming bits across four bit-slices, each control signal itself is shared. This leads to significant area savings since only one pin is required to connect a control signal to the four-bit slice. Control inputs to lookup tables are treated differently. They have the same pin-to-channel connections as other control signals, but are replicated four times within the logic block to form the equivalent of a data bus.

As in custom VLSI, control signals are run perpendicular to the direction of data flow, giving the control routing structure a strong vertical bias. Ideally, control routing segments should span the entire datapath for improved density and performance. As well, since all control signals are generated above the datapath, the use of long lines would be beneficial. In order to improve routability when more than one datapath is stacked vertically, shorter tracks may also be needed.

The use of programming bit sharing causes two major difficulties. First, since data signals are routed as four-bit buses, it is not possible to implement shifts of non-multiples of four. Second, irregularities in bit-slices cannot be handled. To solve these problems, an additional resource, referred to as the shift block, is included in the DP-FPGA.

The shift block is illustrated in Figure 9. A bidirectional CMOS shifter forms the core of this unit, allowing data buses to be shifted up or down by zero to three bits. Each block can be connected to those directly above and below to perform shifts on wide buses. Each input or output is selected from one of the following four sources on an individual bit basis: a data signal, a common control signal, power or ground. While many programming bits are required, the high degree of flexibility in the choice of signal sources has several purposes. It allows the bits being shifted in to be programmed as zero, one or a control signal and sign extension to be implemented. It provides the ability to place constants on data and control tracks which is useful in handling irregularities, as explained below. Finally, since the connection is bidirectional, control signals can be inserted into data buses and conversely, individual data signals can be tapped off using control resources. The need for a block of such generality was found when implementing circuits in an architecture which uses bit sharing.

FIGURE 9 Shift block.

In many datapaths, the logic in some bit-slices is slightly different from the others. Such irregularities can be handled using constant data buses since an independent value can be placed in each bit position. Consider an example in which the logic for the most significant bit of a four-bit section is $F(a,b,c)$, but for the other three bits is $G(a,b,c)$. A four-input lookup table can be used to implement this logic by calculating F and G for each slice and selecting one result using a multiplexer controlled by the fourth input s . Setting s equal to the constant 1000_2 , will produce the required functionality. The failure to include a mechanism to handle such irregularities would significantly restrict the range of circuits that could be implemented using the DP-FPGA.

3. INVESTIGATION OF DATAPATH LOGIC BLOCKS

This section presents a more detailed study of datapath logic blocks. It parameterizes the set of possible logic blocks based on our informal characterization of datapath circuits and conducts an experimental evaluation of their relative merits. This study is conducted similarly to [8][10][11], but considers only datapaths and is based on the DP-FPGA architecture.

3.1. Logic Block Options

The description of the logic block specifies the components for a single bit-slice. This slice is replicated four times to form the complete logic block, with control and programming signals being common to all four slices. For further ease of description, the shared memory lookup table is described as if there are four separate tables that are programmed identically, although it is really implemented as a single table with four read ports.

The trade-off between area efficiency and logic block granularity is examined by varying the size of the lookup table as was done in [8]. In addition, we

investigate the possibility of using dedicated resources to increase the density of an FPGA. A dedicated resource will improve the overall density if the reduction in the total number of blocks as a result of the added functionality outweighs the increase in tile area. Thus, the resource must be highly utilized.

Our initial analysis of datapaths showed that the most common functional units are adders, multiplexers and flip-flops. We determined in section 2 that fast carry support for arithmetic functions was essential to datapaths. The use of dedicated logic for multiplexers is investigated by optionally including a hardwired multiplexer and/or a three-state output driver. These options are illustrated in Figure 10. To support sequential logic, a flip-flop per bit is included in every block. As well, dedicated enable circuitry and direct inputs from the logic block pins (allowing the flip-flop to be used independently of the lookup table) can be added.

The inclusion of numerous dedicated resources in the logic block raises the question of which signals should be available as block outputs. In this study, we consider sixteen different output configurations. One such configuration (enumerated as #7) is shown as part of the sample logic block in Figure 10. One output is either the multiplexer or flip-flop output, while the possibilities for the other include these two plus the option to place the output at high impedance. The other output configurations are variations, with the number of output pins ranging from one to four and the number of three-state buffers ranging from zero to two.

Each of the features being examined has been assigned a parameter to ease the identification of the various architectures. These parameters are summa-

FIGURE 10 Example of a logic block candidate (identified as d1e1k4m1o7).

rized in Table I. Parameters D, E, and M take on binary values, with a value of 1 indicating the presence of the resource. The output configurations are enumerated as 1 through 16. This parameter (O) also signifies the presence of one or more three-state buffers. A total of 640 combinations of the five parameters are possible.

3.2. Experimental Procedure

An experimental approach is taken in order to compare the relative merits of the different logic block options. As with all empirical studies of this nature, the usefulness of the results is highly dependent on the benchmarks used. The eight benchmark circuits used in this study were selected to encompass a range of circuit characteristics, from perfectly regular designs to systems containing several datapaths with irregularities. For example, the circuit *mult*, is a conventional array multiplier and is perfectly regular. In contrast, the circuit *awsim* is a hardware accelerator for circuit simulation and includes several subsystems with numerous irregularities.

Each circuit was partitioned into every logic block configuration using an optimization goal of minimizing the number of logic blocks required. The implementation procedure is as follows:

For each benchmark circuit

1. Separate the datapath(s) from the rest of the circuit and extend the width(s) to a multiple of four if necessary. Add logic required to handle irregularities.

For each logic block architecture

2. Remove logic that will be implemented using dedicated resources, such as carry chains, flip-flops, flip-flop enable logic and three-state buffers.
3. Partition remaining combinational logic into K-input lookup tables using chortle-crf [12].
4. Search graph of lookup tables for multiplexer functions. Remove multiplexer logic

when beneficial to do so and replace with a dedicated multiplexer or two three-state buffers. If a multiplexer is removed (only one removed per iteration), return to step 3, else continue to step 5.

5. Pack all resources (lookup tables, carry chains, multiplexers, flip-flops, three-state buffers) into logic blocks under the constraints imposed by the output configuration. When packing flip-flops, utilize direct inputs to minimize the number of blocks.
6. Output the total number of logic blocks required.

End for

End for

Since step 1 of the procedure was done only once for each circuit, it was done manually. Once all datapath widths are multiples of four bits and logic has been added to handle irregularities, the technology mapping problem becomes equivalent to that of general purpose FPGAs which have bit-level granularity (since four-bit slices are mapped as a unit). The inner loop of the procedure, which was executed 640 times per circuit, was fully automated. The CAD tools developed are based around the Chortle lookup table mapper. An iterative approach to utilizing the dedicated multiplexers and three-state buffers was chosen to yield excellent results while sacrificing compute-time efficiency. In a practical application, the identification of multiplexers would be built into the LUT mapper algorithm. A pre-processor (step 2) and a post-processor (step 5) are used to separate out the logic which must be mapped and then to repack all of the resources to form complete logic blocks.

TABLE I Logic block options.

Parameter	Possible Values	Description
D	0, 1	direct input to flip-flop
E	0, 1	dedicated flip-flop enable circuitry
K	2–6	number of lookup table inputs
M	0, 1	dedicated multiplexer
O	1–16	output configuration

The primary objective in developing the mapping procedure was to achieve the highest quality results possible and not to bias the mapping towards any particular architecture. The quality of the mapping in this study is essentially determined by the quality of the Chortle program, which has been shown to yield good results [12]. The other main steps which influence the results are either done by hand or done in an exhaustive manner. A limitation in our use of the dedicated multiplexers was that no attempt was made in step 4 to use them to implement other functions such as AND, OR or XOR gates. Our intention here is to determine if replacing multiplexers with hardwired logic is beneficial. In the future, we may develop more sophisticated tools to use the multiplexer to implement these functions.

Since the block sizes differ, an area model is used to compare the estimated sizes of the blocks. We consider the logic block area and routing area separately. Our model of logic block area is similar to the model described in [1], but is extended to incorporate the use of programming bit sharing and to account for the various optional resources. An area cost is assigned to all of the main components of the block using a number of parameters. A nominal value (assuming a 1.2 μm CMOS process) is assigned for each parameter, representing our estimate of the cost. Each parameter can be varied to determine its effect on the total area. Table II summarizes the parameters used. Since bit sharing is used, some components will be shared across four slices, while others will be replicated four times. The fixed area accounts for components such as the carry chain and flip-flops which are common to all architectures. Although the shift block is consid-

ered a routing resource, its area is modeled in the same manner as the logic block since its structure is similar.

We use Hill's technique of estimating routing costs by assigning a cost for each logic block pin [11]. Since control lines have less flexibility than data lines, we suggest that a reasonable value for the control pin cost is eight memory cells, while that for data pins without sharing would be fourteen. These costs are doubled for pins with bit sharing to account for the larger number of transistors. These routing parameters correspond to our initial layout studies of the DP-FPGA.

3.3. Experimental Results

Using the procedure outlined in the previous section, the number of logic blocks required to implement each of the eight benchmark circuits in all 640 logic block architectures are determined. These values are translated into silicon area estimates using the area model. The different logic block architectures are compared using normalized averages to give each circuit equal weighting. If the implementation area of circuit i using logic block x is represented as A_x^i , then the corresponding normalized area N_x^i , as defined in [8] is given by

$$N_x^i = \frac{A_x^i}{\min_{\text{all } y} \{A_y^i\}}$$

A summary of results using the nominal values for all area parameters is provided in Table III. The architectures listed were chosen such that the best block for each value of each parameter, except the output configuration, is included. These results show that no block is clearly superior, with several in the range of experimental variation. Thus, rather than treating these figures as absolute, they are used as a guide to study the interrelationships between the parameters and to determine why some features might be more useful than others.

Table II Logic Block area parameters.

Parameter	Description	Nominal Value (μm^2)
BA	SRAM cell	500
FA	fixed area	12000
MA	multiplexer	150
TA	three-state driver	360

Table III Summary of results.

Architecture Tag	Normalized Area	Amount Larger Than Best Block
d1e1k4m0o7	1.202	0.0%
d1e1k3m0o7	1.228	2.2%
d1e1k4m1o6	1.228	2.2%
d0e0k3m0o5	1.256	4.5%
d1e1k5m0o6	1.281	6.6%
d1e1k6m0o6	1.642	36.6%
d0e0k2m1o5	1.939	61.3%

First, we examine the use of dedicated flip-flop enable logic and direct flip-flop inputs, both individually and in combination. Figure 11 plots the optimal architecture for each of the four cases versus K . The use of enable and no direct input does not appear to be a good idea. For small values of K , the increased block cost due to the additional control pin exceeds the benefits, while for larger tables the usefulness of the extra circuitry is reduced due to the increased lookup table functionality.

Direct flip-flop inputs allow the flip-flop to be used independently of the lookup table. The output configuration must allow access to both output signals simultaneously. This feature saves a logic block each time it is used since a lookup table would have to be used as a buffer otherwise. The main cost is an addi-

tional data input pin. It is not intuitive that this option would be frequently used as normally the flip-flop input is generated from combinational logic (lookup table). Without enable circuitry, it was used in the benchmark circuits in two situations: (i) the flip-flop input comes directly from a flip-flop output (common in pipelined designs); (ii) the output configuration does not allow all needed signals to be accessed, forcing the registered value to be generated elsewhere. Figure 11 shows that direct input without enable logic is not beneficial for small K (pin cost is too high relative to total block size), but is marginally useful for K greater than three.

While these two features are not beneficial individually, the combination offers an advantage. The reason for this is that the dedicated enable logic creates the following additional situations for using direct inputs: (iii) case (i) above where the flip-flops have enables (e.g. pipelines which can be stalled); (iv) registered inputs which use load signals; (v) signals fanning out to two registers with different enables. With the exception of situation (ii), the use of direct inputs is independent of K . Since the relative cost of this feature decreases with increasing K , it becomes increasingly beneficial. It is important to keep in mind that for datapaths if a feature is used in one slice, it will likely be used in all slices. Thus, each situation in which the direct input is used saves a logic block for every four bits, which becomes increasingly significant for wider datapaths.

We now consider the use of dedicated multiplexers and three-state buffers, again both individually and in combination. Figure 12 plots the average normalized area for the best blocks with these options against lookup table size. It shows that the dedicated multiplexer is not beneficial. The main costs associated with the dedicated multiplexer are one data pin and one control pin. Rose et al. concluded that the most area efficient blocks are those with the largest amount of functionality per connected pin [8]. When implementing datapaths, the two additional input pins associated with the inclusion of a dedicated multiplexer are too expensive to justify the amount of additional functionality they provide when used strictly as a multiplexer.

FIGURE 11 Area vs. K when using dedicated enable logic and direct flip-flop inputs.

FIGURE 12 Area vs. K when using dedicated multiplexers and three-state buffers.

Interestingly, the results show that the inclusion of a dedicated three-state buffer, which is used to construct distributed multiplexers, may be beneficial. Since the data input is hardwired, only an additional control pin is needed. Our mapping scheme uses three-state buffers to build two-to-one multiplexers whenever it is beneficial to do so. As well, three-state buffers can be used to build wide multiplexers efficiently (this transformation was done by hand). For example, using four three-state buffers to build a four-to-one multiplexer instead of three three-input lookup tables will save three logic blocks. Of course, more control logic is required to generate the additional two control signals (which is not accounted for in our experiments), but we expect this cost to be small compared to the savings gained across a wide datapath.

There are several considerations in determining the configuration of logic block output signals. A decision must be made as to which signals should be available as possible outputs. As well, the number of logic block output pins must be determined. This involves a trade-off. Having a large number of outputs improves the utilization of the block, but requires more data pins. If the number of output pins is less than the number of output candidates, a multiplexing scheme must be used.

Figure 13 plots average normalized area against number of output pins for some interesting blocks. In cases where there are several configurations having the same number of pins, the best one is chosen for each architecture. This plot shows that having two output pins is the best choice when the architecture contains both direct input capabilities and dedicated logic for flip-flop enables. This is because the majority of blocks require only one or two outputs. Using three outputs is a close second choice, as the extra output allows the three-state buffers to be more heavily utilized.

Figure 13 shows that if the logic block does not have direct input and enable options, then only one output should be used. These results suggest that extra output pins are only needed to allow the lookup table and flip-flop to be used independently.

If no three-state buffer is included in the block, then only the lookup table and flip-flop outputs are needed. If three-state buffers are used, then multiplexing logic is needed to get at the four possible signals. Our studies have shown that since the estimated cost of the connection interface is larger than that of the multiplexing logic, it is best to make the output configuration highly flexible. However, full flexibility would require two three-state buffers. Since they would rarely be used simultaneously, the

FIGURE 13 Area vs. number of output pins.

extra cost is not justified. Thus, a better solution is to provide the most flexibility possible with only one three-state buffer, as shown in Figure 10.

Most of the results have been presented as a function of K . All of the curves display the same general shape, with the minimum areas occurring at values of either three or four, with the results slightly favouring four. Based on the results presented in [13], we expect four input lookup tables to provide better performance as well, making $K = 4$ a good choice.

Our model makes assumptions about the area of circuit components which may not correspond exactly to an actual implementation. To study the effect of inaccuracies in our estimates, we have varied the parameter values around their nominal values. We have found that reasonable sized variations do not change the results significantly. In particular, we investigated the influence of the pin cost estimates as these parameters are the least well known. We found that varying the data pin cost by $\pm 50\%$ did not change the best block. As the control pin cost is raised above 25% of its nominal value, the use of the three-state buffer (which adds a control pin) becomes inferior. The block that produced the smallest average area over a wide range of parameter variations is the one shown in Figure 10, except no dedicated multiplexer is included (D1E1K4M007).

3.4 Comparison with Commercial Architectures

Having described the DP-FPGA architecture and discussed the experimental results, we now make a qualitative comparison with existing commercial devices, namely the Xilinx XC4000 [2], Altera Flex [4], and AT&T ORCA [5] series FPGAs. These three were chosen due to their similarities with the DP-FPGA (SRAM technology, lookup tables) and due to their added support for datapaths. All three have included a dedicated carry chain which has widely been found to be essential for datapaths. However, their ripple carry techniques cannot match the four-bit carry skip scheme of the DP-FPGA in terms of performance, which can be critical for wide datapaths.

All three commercial devices include a flip-flop per lookup table output to build data registers. The XC4000 and ORCA architectures also have direct input and enable capabilities which we have shown experimentally to be good for datapaths. Altera provides only one block output (either D or Q of the flip-flop) which Figure 13 suggests is the proper number when no direct input and enable are used. The XC4000 allows both D and Q to be available simultaneously, but AT&T does not (for four-bit buses). This means that in the ORCA architecture it is not possible to use the direct input to build a four-bit data register and independently use the lookup tables for four bit-slices of datapath logic (they can be used for a control function).

While many of the features of existing FPGAs are similar to those of the DP-FPGA, there is a fundamental difference which is the amount of resource sharing that is exploited. The Flex block is one bit wide, allowing no sharing between slices. The XC4000 block can share control pins for flip-flop enables, set/resets and clocks across two slices. The ORCA block can implement four slices and utilizes the highest degree of sharing. Enables, set/resets and clocks are shared by all four flip-flops. In addition, three lookup table inputs are shared between pairs of 4-LUTs. However, this also limits the lookup table functionality for datapaths as the four 4-LUTs cannot implement functions of 2 data and 2 control inputs or 3 data and 1 control input, making them more comparable to the 3-LUT version of the DP-FPGA. Control inputs to lookup tables are shared across only two slices, as compared to four in the DP-FPGA. However, the ORCA structure is exceptional at building wide multiplexers.

The DP-FPGA not only shares control signals across four slices, but programming information as well. For example, the four 4-LUTs used in a four-bit wide section of datapath logic require 64 SRAM cells in each of the three commercial FPGAs, whereas the DP-FPGA requires only 16. The same ratio holds true for all logic block features requiring programming cells and most importantly in the data routing fabric, giving the DP-FPGA superior density for datapaths.

Unfortunately, there is a penalty for using such a degree of sharing, which is the inability to implement control function with reasonable cost. Distinct control logic blocks are required.

4. EXPERIMENTAL ANALYSIS OF BIT SHARING

Without having fully defined the routing architecture at this point in our research, we cannot make a numerical comparison of the DP-FPGA architecture with presently available commercial architectures. We would, however, like to get an indication of the potential of such an FPGA. To gain some insight into the usefulness of bit sharing, we compare the best block architecture determined in section 3 with the same architecture without bit sharing (independent 1-bit blocks).

For the purpose of the following discussion, we define an ideal datapath as one whose width is a multiple of four and which contains no irregularities. An ideal datapath mapped into logic blocks without bit sharing requires exactly four times the number of blocks. If no shifting is allowed, the 4-bit block will provide much better density since approximately one-fourth the number of SRAM cells are required. These savings are reduced for real datapaths due to the following three factors: (i) datapath widths need to be extended to multiples of four; (ii) irregularities require additional logic; (iii) shift blocks are required. The reduction in flexibility due to bit sharing in factors (i) and (ii) may require additional blocks in the bit sharing case, reducing the 1-bit to 4-bit block ratio below four. Factor (iii) is an overhead cost associated with bit sharing, regardless of regularity.

To see how these factors influence the implementation area, the benchmark circuits were mapped into the 1-bit blocks using the procedure described in section 3, except all bits are mapped independently and no additional logic is added for datapath extension or irregularities. A modification of the area model is used to estimate the area of the one-bit block. This

involves the straightforward scaling of the logic block area components and the removal of the shift block. The routing area modifications require an estimate of the relative cost of a 4-bit data pin to a 1-bit data pin. While the number of tracks per slice and thus the number of SRAM cells per pin will not change, each SRAM cell in the 4-bit version will control four pass transistors or multiplexers rather than one. As a conservative estimate we set the cost of the 1-bit data pin as half that of the four-bit pin. The cost of a control pin does not change. Table IV summarizes the results.

If each circuit consisted of only ideal datapaths, the number of blocks in the 4-bit case would be $\lceil A / 4 \rceil$. The column titled "Increase over Ideal" represents the percentage increase of the actual number of 4-bit blocks over this ideal value. It is an indication of how far each circuit is from ideal regularity due to factors (i) and (ii). Table IV shows that an average of 13% more blocks are required. Despite these costs and the shift block overhead, the total area is still estimated to be 2 to 2.5 times larger if bit sharing is not used. While this is only a rough approximation since the actual values depend heavily on the physical layout, it suggests that significant improvement in density can be achieved through the use of bit sharing. The effect of bit sharing on other architectures or using other area models will depend on the cost of the SRAM cells and control circuitry relative to the total tile cost, as the area of these resources is reduced by a factor approaching four.

Table IV The effect of bit sharing on the number of logic blocks.

Circuit	A # Blocks Without Sharing	B # Blocks With Sharing	$\frac{A}{B}$	Increase over Ideal	$\frac{\text{Area (A)}}{\text{Area (B)}}$
awsim	287	92	3.1	28%	2.0
coder	673	193	3.5	14%	2.2
csamult	516	162	3.2	26%	2.0
filter	396	100	4.0	1%	2.5
fpadder	395	123	3.2	24%	2.0
hubnet	199	53	3.8	6%	2.4
mult	64	16	4.0	0%	2.5
viterbi	310	82	3.8	5%	2.4

The preceding discussion considers only the effectiveness of bit sharing. The optimizations of the logic block and routing architectures discussed are also expected to contribute significantly to the DP-FPGA's density advantage.

5. CONCLUSIONS AND FUTURE WORK

This paper has proposed a new FPGA architecture which provides a significant improvement in the density of datapaths compared to general-purpose architectures. Different logic and routing resources are utilized for the datapath, control and memory portions of a system, allowing them to be optimized separately. Datapath density gains are made primarily through the use of a technique referred to as programming bit sharing. Experimental estimates show that the use of bit sharing can improve densities by a factor of two over an identical architecture that contains no sharing. Further optimizations of the logic block and routing architectures are expected to contribute to the density improvements as well.

Each logic block in the DP-FPGA operates on N bits of data, with typically N equal to four. An investigation of single output lookup table-based logic blocks within the DP-FPGA framework was conducted. We concluded that a dedicated carry chain and a four-bit register are essential components of the logic block. Based on experimental results, we determined that a lookup table with four inputs per bit-slice would be a good choice of lookup table granularity. We also found that allowing the register to be used independently of the lookup table is beneficial provided that dedicated enable circuitry is included. While the use of dedicated three-state buffers inside the logic block proved to be marginally useful for implementing distributed multiplexers, the inclusion of a multiplexer with one input hardwired to the table output was not.

In the future, we will investigate a number of issues concerning the datapath routing architecture, such as the use of nearest-neighbour connections, the

channel widths, the segment length distribution and the pin-to-channel and channel-to-channel flexibilities. As well, the control and memory sections as well as their interconnections to the datapath will be explored. While this paper focused primarily on reducing area, we will also study its ability to improve performance in more detail.

References

- [1] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
- [2] Hung-Cheng Hsieh et al., "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *IEEE 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.2.1–31.2.7.
- [3] Mike Ahrens et al., "An FPGA Family Optimized for High Densities and Reduced Routing Delay," *IEEE 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.5.1–31.5.4.
- [4] *Altera FLEX Programmable Logic Data Sheet*, Altera Corp., 1992.
- [5] *AT&T ORCA Data Sheet*, AT&T Bell Laboratories, 1992.
- [6] Manisha Agarwala and Poras T. Balsara, "Application Specific Logic Module Architecture for FPGAs," *IEEE 1992 Custom Integrated Circuits Conference*, 1992, pp. 4.1.1–4.1.4.
- [7] D.C. Chen and J.M. Rabaey, "A Field Programmable Architecture for High Speed Digital Signal Processing Applications," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 117–122.
- [8] Jonathan Rose, Robert J. Francis, David Lewis and Paul Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid-State Circuits*, Oct. 1990, pp. 1217–1225.
- [9] Benjamin Tseng, Jonathan Rose, Stephen Brown, "Using Architectural and CAD Interactions to Improve FPGA Routing Architectures," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 3–8.
- [10] Jack L. Kouloheris and Abbas El Gamal, "FPGA Area versus Cell Granularity—Lookup Tables and PLA Cells," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 9–14.
- [11] Dwight Hill and Nam-Sung Woo, "The Benefits of Flexibility in Look-up Table FPGAs, FPGAs," W. Moore and W. Luk Eds., Abingdon 1991, edited from the *Oxford 1991 International Workshop on Field Programmable Logic and Applications*, pp. 127–136.
- [12] R.J. Francis, Jonathan Rose and Zvonko Vranesic, "Chortlecrf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proceedings of the 28th ACM/IEEE Design Automation Conference*, June 1991, pp. 227–233.
- [13] Satwant Singh, Jonathan Rose, Paul Chow and David Lewis, "The Effect of Logic Block Architecture on FPGA Performance," *IEEE Journal of Solid-State Circuits*, Mar. 1992, pp. 281–287.

Authors' Biographies

David Lewis (M '87) received the B.A.Sc. degree with honours in engineering science from the University of Toronto in 1977, and the Ph.D. degree in electrical engineering in 1985. 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, and associate professor since 1991. His research interests include logic and circuit simulation, logarithmic arithmetic, signal processing field programmable hardware, and VLSI architecture.

Dr. Lewis is a member of the IEEE and the ACM.

Don Cherepacha received his B.A.Sc. in Engineering Science from the University of Toronto in 1991. He is currently a master's candidate in Electrical Engineering at the University of Toronto. He joined LSI Logic as a Design Engineer in 1993, where he is developing chipsets for symmetrical multiprocessing servers. His interests include architectures and CAD for FPGAs, computer architectures, and multiprocessing systems.