Field Programmable Gate Arrays and Floating Point Arithmetic

Barry Fagin and Cyril Renard

Abstract—**We present empirical results describing the implementation of an IEEE Standard 754 compliant floating-point adder/multiplier using field programmable gate arrays. The use of FPGA's permits fast and accurate quantitative evaluation of a variety of circuit design tradeoffs for addition and multiplication. FPGA's also permit accurate assessments of the area and time costs associated with various features of the IEEE floating-point standard, including rounding and gradual underflow. These costs are analyzed, along with the effects of *architectural correlation*, a phenomenon that occurs when the cost of combining architectural features exceeds the sum of separate implementation. We conclude with an assessment of the strengths and weaknesses of using FPGA's for floating-point arithmetic.**

I. INTRODUCTION

We have recently implemented an IEEE Standard 754 compliant floating-point multiplier and adder using field programmable gate arrays. We report our results below. We assume the reader is familiar with antifuse based FPGA's and the IEEE floating point standard. For more information on these topics, the reader is referred to [1]–[3].

We began our experiment with the development of a core design, shown in Fig. 1. We then divided the addition and multiplication algorithms into three stages, shown in Fig. 2. Once the basic algorithm was developed, we began investigating circuits for the adder and multiplier. Many possibilities were considered, with the limited resources of the gate array as the overriding concern. This is discussed below.

II. SPACE/TIME TRADEOFFS IN ADDITION AND MULTIPLICATION

Addition and multiplication circuits can be designed in a number of ways, occupying multiple points in the cost/performance plane. Different solutions attractive for different technologies. For a more detailed discussion of the issues involved in adder and multiplier design, the reader is referred to [4]. We report all our results in terms of *logic modules,* or LM's, the basic unit of area allocation in the Actel FPGA.

We considered a total of five designs for the adder: conventional ripple-carry, carry-lookahead, carry-skip, carry-select, and "hard macro" (a design based on vendor-supplied addition macros for 16 and 8-b adders). A summary of the performance and area costs of these designs is shown in Table I.

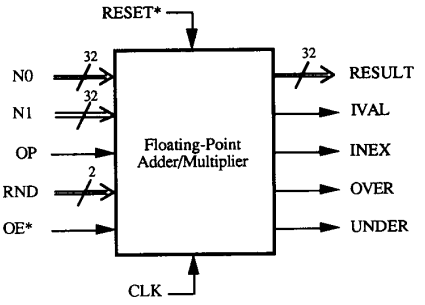
We note that the carry-select design offered the best performance of all candidates, including full carry-lookahead. This is due to the nature of the full adder macro in the Actel design suite, and the mux-based nature of the underlying technology. Although the ripple-carry design offered the best performance to cost ratio, we chose to implement the carry-select design. The improvement in performance was, we felt, more than worth the cost, particularly since our preliminary estimates indicated that the number of LM's associated with the adder was less than 5% of the total.

Manuscript received September 16, 1993; revised February 8, 1994 and March 16, 1994. This work was supported by the National Science Foundation under award numbers MIP-9222643 and MIP-9312350. Direct Imaging Incorporated, Actel Corporation, Xilinx Incorporated, Sun Microsystems, Viewlogic Incorporated, IBM, and National Semiconductor also provided support.

B. Fagin is with the Department of Computer Science, United States Air Force Academy, CO 80840 USA.

C. Renard is with IRESTE, University of Nantes, France.

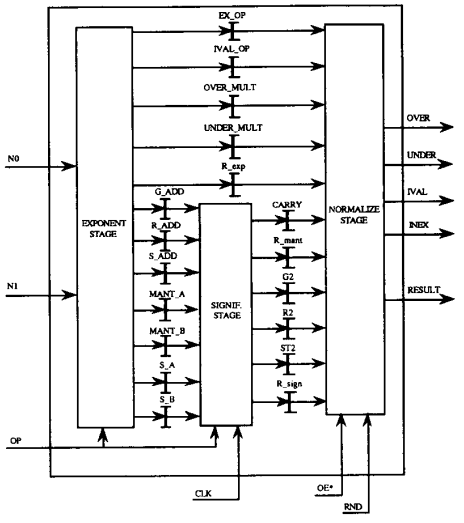
IEEE Log Number 9403173.



INPUTS:
N0, N1 32-bit IEEE standard floating-point operands
OP 1-bit operation code (0=add, 1=mult)
RND: 2-bit control signal indicating rounding mode
OE*: output enable for RESULT, see below)
RESET*: system initialization signal
CLK: system clock

OUTPUTS:
RESULT: result of OP applied to N0, N1
IVAL: indicates invalid operation
INEX: indicates inexact operation
OVER: indicates overflow
UNDER: indicates underflow

Fig. 1. Core design and input/output signal description.



INTERNAL SIGNALS:

EX_OP: 2-bit control signal indicating exceptional operation that requires specially encoded result. 00 = no exception, 01 = zero, 10 = +∞, 11 = -∞.
IVAL_OP: invalid operation
OVER_MULT, UNDER_MULT: over/underflow detection during exponent calculation of a multiplication
R_exp: exponent of result
MANTA_A, MANT_B: unpacked significands
S_A, S_B: sign of significands
CARRY: carry from new significand, needed for normalization and rounding
R_mant: significand of result
R_sign: sign of result

Fig. 2. Core design detail and internal signal description.

TABLE I
AREA COST AND PERFORMANCE OF ADDER DESIGNS

Design	Area (LMs)	Critical path (ns)	Perf (MHz)	Perf/Cost (MHz/LM)
Ripple-carry	48	150	6.67	.139
Carry-lookahead	260	100	10	.38
Carry-skip	83	115	8.70	.105
Hard macro	141	110	9.09	.64
Carry-select	112	70	14.29	.128

TABLE II
AREA COST AND PERFORMANCE OF MULTIPLIER DESIGNS

Design	Cost (LMs)	Cycle Time (ns)	Latency (ns)	Perf/Cost (KHz/LM)
Basic	260	100	2400	1.60
1 CSA	385	63	1582	1.64
8 CSA	1103	200	800	1.13
Combinatorial	2704	460	460	0.80

TABLE III
AREA COSTS OF PIPELINING, ROUNDING, AND GRADUAL UNDERFLOW

Pipelining	Rounding	Gradual Underflow	LMs
0	0	0	2642
0	0	1	2828
0	1	0	3362
0	1	1	3929
1	0	0	2805
1	0	1	2991
1	1	0	3525
1	1	1	4098

We considered four designs for our multiplier: a basic shift-and-add multiplier computing one bit per cycle, a design employing a carry-save adder, a design employing eight carry-save adders, and a combinatorial multiplier. A summary of the cost and performance of these designs is shown in Table II.

We selected the 8-CSA design for our final implementation because we felt it offered the best tradeoff between latency and resource requirements. Other designs had better performance to cost ratios if considered in isolation, but we believed their accompanying latencies would have unacceptable impacts on system performance.

III. AREA AND PERFORMANCE COSTS OF PIPELINING, ROUNDING AND GRADUAL UNDERFLOW

Any combination of pipelining, rounding, and gradual underflow can be incorporated in a design. We performed a complete examination of the design space by examining the area requirements of all possible designs, as shown in Table III. A 0 in a table entry indicates the absence of the associated feature, while a 1 indicates its inclusion.

To better view the costs of each feature separately, we can place each design at the vertex of a Boolean 3-cube, labeling each axis with the difference in area requirements. This is shown in Fig. 3. The basic design is at vertex 000, with the inclusion of features as specified in the axes on the right. Edges are labeled with the number of LM's necessary to include the associated feature, along with the accompanying percentage increase.

We see from Fig. 3 that the cost of pipelining is not particularly significant, ranging between 4 and 6% for any design. This is not surprising, given the lack of feedback in the sequential design Fig. 2 and the ease with which floating-point addition and multiplication algorithms are pipelined. The necessary logic is essentially some extra

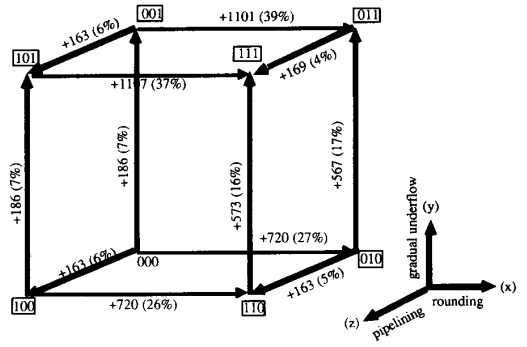


Fig. 3. The design space as a Boolean 3-cube.

registers and control circuitry, a relatively small investment for the expected improvement in performance.

We note that all arrows pointing in the 'z' direction, corresponding to the inclusion of pipelining in a design, contain values very close to one another. This suggests that architectural support for pipelining is independent of the other features in the design space. This is not true for arrows in the 'x' and 'y' directions. Support for denormals, for example, requires an extra 186 LM's if rounding is not supported, but approximately three times as many LM's if rounding is supported. This is because gradual underflow and rounding are architecturally correlated: circuitry is required when both features are present that is not required if either feature is used individually.

To see why this is so, consider the decision to add denormalized numbers to designs with and without rounding. A design that already supports rounding without denormalization will require the addition of both a variable-width shifter at the output of the normalization stage and extra logic to support the correct manipulation of the guard, round, and sticky bits associated with the shifter. A design without rounding, by contrast, requires only the extra shifter. This explains why the arrows in the y direction in the right half of Fig. 3 are labeled with a significantly larger value than those in the left half: i.e., "100" $\rightarrow$ "101" = +186 (7%) but "110" $\rightarrow$ "111" = +573 (16%). These values represent the resources necessary to add denormalized numbers to designs that already support the four rounding modes. The logic needed to manipulate the guard, round, and sticky bits associated with the denormalization shifter is only required if both rounding and gradual underflow are supported.

We may examine architectural correlation more closely by examining the effects of feature incorporation on individual pipeline stages. A breakdown of the four pipelined designs is shown in Fig. 4. We see that support for denormalized numbers affects only the normalization stage, due to the addition of the variable width shifter. Adding rounding to a base design, by contrast, requires significant changes to both the exponent and normalization stages due to the manipulation of the guard, round, and sticky bits. Finally, adding rounding to a design that supports denormals requires both the addition of the rounding logic required for the base design plus extra logic for the variable-width shifter in the normalization stage. This explains the large increase in normalization logic in design 111.

In addition to area costs, the decision to support rounding and gradual underflow has performance costs as well. Simulations indicate that the cycle time of 200 ns, initially determined by our multiplier design, is lengthened if rounding and/or denormalized numbers are supported. For designs that support rounding alone, the worst case occurs when addition is performed using the largest number and the rounding mode is towards $+\infty$. The generation of overflow from

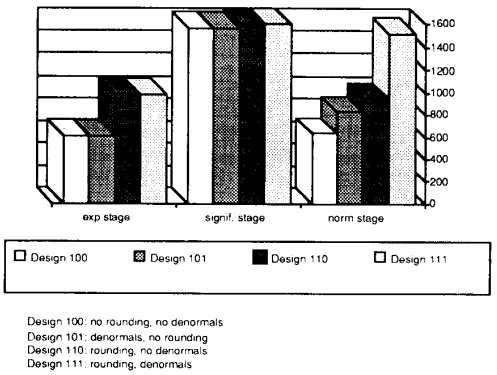


Fig. 4. Area requirements of pipelined designs.

the rounding logic increases the critical path to 245 ns, whether or not denormalized numbers are supported. For designs that support denormalized numbers without rounding, the delay through the shifter increases the critical path to 235 ns.

For a more detailed discussion of these tradeoffs and other issues, the reader is referred to [5]. Copies may be obtained from fagin@cs.usafa.af.mil.

IV. CONCLUSION

Our final design was partitioned over 4 Actel A1280 FPGA's, with a 3-stage pipeline and a cycle time of 245 ns. Addition has a 3 cycle latency, while a multiply requires 6 cycles: 1 for the exponent stage, 4 for the significand stage, and 1 for the normalization stage. We were unable to reduce the latency for multiplies due to the resources required for the 24-b multiplier.

The multiplier is the largest single component of our system. Based on the data in Table II and [2], the design with the best performance does not fit on the largest chip currently available. This is due to the large data width required by our system. Pin limitations do not appear to be a problem, but current chip densities need to improve by a factor between 2 and 4 before 24-b multipliers can fit on a single die. Our results indicate that this is the single most significant obstacle to floating-point calculation using FPGA's.

We were also able to use FPGA's to rapidly explore the design space in a variety of ways. Our designs for adders and multipliers indicate that technology-dependent effects predominate. The superior performance of the carry-select adder, for example, was due to the existence of efficient mappings between it and the basic cell structure of the target FPGA technology. As the technology matures, we hope that the relationships between the target device structure and optimal design will disappear. Just as compilers are now sufficiently sophisticated to permit users to write efficient programs independent of the architecture of the target machine, we hope that CAD tools will advance to the point where designers can produce efficient designs without a knowledge of the basic structure of the FPGA. Our results indicate, however, that this goal remains elusive.

Finally, our experiments suggest that FPGA's can be used effectively to evaluate design tradeoffs, and to empirically evaluate the costs of including architectural features. Users wishing to support a subset of a given set of features can obtain a better understanding of the consequences of design choices, to decide which options are right for their application. Users who wish to trade speed for accuracy, for example, may choose to truncate inexact results and reduce chip cycle times. Precision can also be sacrificed; by implementing a shorter data word, the area requirements of the multiplier are reduced and faster implementations can be employed within the resource constraints of a single chip. We have focused on the IEEE floating-point standard, but the techniques described here apply to virtually any set of architectural features to be implemented under resource constraints.

The existence of *architectural correlation,* however, complicates the design decision. Features may require greater resources when combined than when implemented separately, as indicated by our analysis of gradual underflow and rounding. Architectural correlation implies that an understanding of individual architectural features is not sufficient for cost/performance analysis. A knowledge of how features interact is essential.

For future work, alternative datapath widths can be investigated to better understand speed/precision tradeoffs. Other standards and application domains, such as data compression and image processing, can be explored using FPGA's. It is also important to see how strongly our results apply to other technologies, including reprogrammable FPGA's and EPROM-based devices. Perhaps most critically, comparison with other technologies is warranted. It is important to compare the results obtained with these experiments to similar analyses for full custom ASIC's, to better understand the effects of device technology on the area and performance cost of architectural feature support.

ACKNOWLEDGMENT

The authors thank D. Fraser, J. Erickson, and T. Truex for their assistance, and the referees for their efforts to improve the quality of the paper. We also thank Professor Y. Thomas and IRESTE at the University of Nantes.

REFERENCES

[1] A. El Gamal *et al.*, "An architecture for electrically configurable gate arrays," *IEEE J. Solid State Circ.*, vol. 24, pp. 394–398, 1989.
[2] *ACT™ Family Field Programmable Gate Array Databook.* Actel Incorporated, 1992.
[3] IEEE Task P754, "A proposed standard for binary floating-point arithmetic," *IEEE Computer*, vol. 14, no. 12, pp. 51–62, March 1981.
[4] D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufmann, 1990.
[5] C. Renard, "FPGA implementation of an IEEE Standard floating-point unit," *Tech. Rep.*, Thayer School of Engineering, Dartmouth College, NH USA.