Parametric time delay modeling for floating point units

Hossam A. H. Fahmy, Albert A. Liddicoat and Michael J. Flynn

Computer Systems Lab, Stanford University, California, USA.

ABSTRACT

A parametric time delay model to compare floating point unit implementations is proposed. This model is used to compare a previously proposed floating point adder using a redundant number representation with other high-performance implementations. The operand width, the fan-in of the logic gates and the radix of the redundant format are used as parameters to the model. The comparison is done over a range of operand widths, fan-in and radices to show the merits of each implementation.

Keywords: Floating point adders, time delay modeling, signed digits, redundancy

1. INTRODUCTION

Floating point units are used an important part of general purpose processors and can be even more important in dedicated hardware for graphics and digital signal processors. There is a need to have a quick and easy way to determine the most suitable design for a specific application before committing too much time to the details of the design. This document presents a simple parametric model for the time delay of the hardware used in floating point units.

The parameters of the model are the operand width, the logic gate fan-in and the radix of the redundancy for designs using redundant representations. In special hardware for digital signal processing or graphics applications, the operand format does not have to conform to a standard and is usually dependent on the application and on the other design restrictions. The significand may be 16 bits or less. Most general purpose processors conform to the ANSI/IEEE standard.[1, 2] This standard specifies certain formats with specific operand widths. The hardware designer who already designed a floating point unit for one format of the standard or for a specific application in graphics or signal processing may find that the design is not optimal for another format. The proposed parametric model can be used to identify the best architectures based on the given constraints. The range of desirable operand widths, (n), is quite large spanning from about $n = 8$ bits to about $n = 256$ bits depending on the application. Architectures that are beneficial for small operand widths may not be for large operand widths.

The logic gate fan-in (f) is the second parameter to consider. A large number of designs use standard libraries with pre-designed components. The maximum fan-in of the standard gates is usually limited to a small number. For CMOS technologies the fan-in limitation is due to the number of series transistors that may be stacked in one pull-up or pull-down chain. Typical values for the maximum fan-in are between about $f = 2$ and $f = 6$ inputs per gate.

The third parameter used in the model is the radix (2^r) of the digits. Signed digits or other forms of redundancy may be used to improve the unit performance. In such designs, the operand width is divided into digits with some redundancy. When redundancy is present in the format, there will be more than one way to represent each number. The redundancy is introduced to eliminate the need for carry propagation.[3] The radix of the digits can be as small as 2 (binary) or as large as the operand width (non-redundant). Practical values for the radix range from radix 2 to about 256 or digit width between 1 and 8 bits.

The various floating point adder designs that were selected for comparison are briefly described in section 2. Section 3 introduces the model used and the simulations performed to validate the model. The results obtained by applying the model to the different designs are presented in section 4 and finally conclusions are given in section 5.

2. DESCRIPTION OF REPRESENTATIVE DESIGNS

A number of floating point adders that have been described in detail in the literature[4–8] are used for comparison. They are all evaluated according to the delay model mentioned above. All of the designs selected use two-path algorithms for high-performance execution. In all five examples, the far path takes longer than the close path. The results reported in this paper assume the longer far path. Each adder is labeled by the name of the first author on the publication describing the design.

The one by Nielsen is described in more details in two papers.[9,10] The critical path for this adder goes through the exponent subtract (11 bit subtraction), the significand swap, and the alignment shifter (a full length shifter for the 65 bit significand) in the first cycle. In the second cycle, the critical path includes two $[4:2]$ compressors and the adjustment logic. The third cycle requires the sticky digit calculation (implemented by a tree of multiplexers and hence having $\lceil \log_2 n \rceil$ levels) and the rounding logic. The fourth cycle introduces the delay of the final operand width (64 bits) carry propagate adder. This design has a long latency compared to other designs but it provides an improved throughput due to the redundancy used.

The adder proposed by Oberman has the exponent difference (11 bit) and the swap in its critical path for the first cycle. The second cycle includes the operand width shifter and the third cycle has the half adder, the carry propagate adder (operand width), and a multiplexer.

The design proposed by Smith has the exponent difference (11 bit), the swap and the operand width alignment shift in the first cycle. In the second cycle, the operand width carry propagate adder and some multiplexers fall on the critical.

The critical path of the adder presented by Seidel has in the first cycle either the partial exponent adder (7 bit), the bitwise XOR, the operand width shifter and a multiplexer or the full exponent adder, the bitwise XOR, the OR tree then the select lines to the output delay of the multiplexer. In its second cycle, there is the half adder, the compound adder (of full operand width), the one bit location shift, a multiplexer to select between the two shifted results and a final multiplexer to choose between the far and close paths.

The adder previously proposed by the current authors has the exponent difference (5 bit) in its critical path followed by the significand swap, the alignment shifter, the signed digit carry free adder, and the final multiplexer.

3. THE TIME DELAY MODEL AND ITS VALIDATION

The model proposed here gives an estimate of the number of equivalent elementary delay units in the critical path of the floating point hardware. The floating point unit delay is presented in "fanout of 4" delays, or the delay of an inverter driving a load that is four times its own size. This is commonly abbreviated as *FO4* for the "fanout of 4" inverter.

The simulation tool *irsim* (a switch level simulator for transistors) is used to simulate a number of circuits in order to validate the model. All the circuits are designed in a CMOS $0.6\mu m$ technology. The first such circuit is a chain of inverters properly scaled so that each one is four times the size of the preceding one. The chain is used to estimate the time delay of an *FO4* inverter. The pull down time of the inverter is $0.40ns$ while the pull up time is $0.44ns$. Hence, the average delay unit is estimated to be $0.42ns$.

In this model, for any integer adder the following simplified formula giving the gate delay of conditional sum addition[11] is used:

$$\tau = 5 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$$

In the formula, n is the number of bits in the adder and f is the fan-in or the maximum number of inputs for a gate in the design. $[4:2]$ compressors are assumed to take 3 *FO4* delays while a $(3,2)$ counter takes 2 *FO4* delays.[12]

A single m-to-1 multiplexer is considered to take only one *FO4* delay from its inputs to the output assuming it is realized using CMOS pass gates. This assumption for the multiplexer is acceptable as long as m is not very large so that the different CMOS pass gates are not capacitively loading each other heavily. Small m is

the usual case in VLSI design since multiplexers rarely exceed say a 5-to-1 multiplexer. Using *irsim* the 2-to-1, 3-to-1 and 4-to-1 multiplexers are simulated. They all exhibit a time delay from the inputs to the output within the range of one *FO4* delay (i.e. less than $0.42ns$). When the input lines are held constant and the select lines change, the delay from the select lines to the output is found to be larger than one *FO4* delay but not more than two *FO4* delays. Hence, for a single multiplexer the delay from the select lines to the output is assumed to be bound by 2 *FO4* delays. A series of m to 1 multiplexers connected to form a larger n-bit multiplexer is assumed to load its select lines heavily. Hence there is even a larger delay form the select lines to the output in this case. To keep up a balanced design with a fanout of four rule, each four multiplexers should have a buffer and form a group together. Four such groups need yet another buffer and form a super group and so on. The delay of the selection then is assumed to be $\lceil \log_4(n) \rceil + 1$. This last formula is applicable even in the case of a single multiplexer since it yields 2 as given above.

The carry free signed digit adder used in the design by Fahmy is a number of parallel adders each taking digits composed of $r+1$ bits ($radix = 2^r$) and adds them producing their sum, sum plus one and sum minus one. Then a choice is made between those three outcomes and a possible correction made to compensate for carries into the higher up digit. Because of the more complicated carries in this scheme it is assumed that they take an additional *FO4* delay. The choice of which outcome of the adder to produce is basically a multiplexer that has a delay from its select line to its output and then there is an additional *FO4* delay for the last correction. So, the total delay of the signed digit adder is $(5 + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil) + 1 + (\lceil \log_4(r+1) \rceil + 1) + 1$ *FO4* delays. This is a conservative estimate for the signed digit adder. Using over 100000 random test vectors, it is found that such an adder using $r = 4, f = 3$ and composed of three digits has a delay of $4.0ns$. This delay is less than the 10 *FO4* delays predicted by the above formula.

Shifters can either be done by a successive use of multiplexers or as a barrel shifter realized in CMOS pass transistors. In either case, the delay of an n-way shifter from its inputs to its outputs is assumed to take $\lceil \log_2(n) \rceil$ *FO4* delays. The select lines are heavily loaded as in the case of multiplexers. However, if the same idea of grouping four basic cells is used then the delay from the select lines is the same as for the multiplexers. This is smaller than the delay from the inputs to the outputs in the shifter. Hence the input to output delay dominates and is the only one used. A 16-way shifter is designed using NMOS transistors and simulated with *irsim*. The model predicts that its delay must be less than $\lceil \log_2(16) \rceil = 4$ *FO4* delays. Using a set of random inputs to stimulate the simulation, the time delay from the inputs to the outputs is found to be less than $1.2ns$. This delay is equivalent to 3 *FO4* so the model is on the conservative side in this case.

For other pieces of combinational logic where a specific design is reported in the published papers, the delay can be estimated. If the design is not known, and the logic has n inputs then its time delay is assumed to be $\lceil \log_f(n) \rceil$ *FO4* delays.

The different parts of the model are summarized in Table 1. Using units of *FO4* delays makes the model independent of the technology scaling to a large degree since this elementary gate scales almost linearly with the technology.[13] Such units also make the model take into effect the time delay associated with the small local wires inside the *FO4* inverter as well as those connecting it to neighboring gates. However, the model does not include any assumptions about long wires across the chip and the time delay associated with them. Hence, obviously, it does not give an accurate estimate of the absolute delay of a logic unit. However, the model can be used to compare different architectures to estimate their relative speeds. The reason that the model does not differentiate between the delay time of the different types of gates is that the designers usually change the sizes of the transistors in order to equalize the time taken by all gates on the critical path.

4. RESULTS

The time delay of the various designs discussed in the previous section can now be estimated. Since the significand width and exponent width are closely related in the formats used for floating point units, it is assumed here that the exponent width is eight bits for any significand width that is 24 bits or less. Otherwise, the exponent width is assumed to be 11 bits. For the design of Fahmy this translates to 11 bits with the small significand width and 15 bits otherwise. This difference in the exponent width leads to a sudden jump in the time estimated for the delay at the point where the significand width is 24. The assumption of such a step

Part	Delay
Adder	$5 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$
$[4:2]$ compressors	3
$(3,2)$ counters	2
Multiplexer, input to output	1
Multiplexer, select to output	$\lceil \log_4(n) \rceil + 1$
Signed digit adder	$8 + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil + \lceil \log_4(r+1) \rceil$
Shifter	$\lceil \log_2(n) \rceil$
Other (no design details)	$\lceil \log_f(n) \rceil$

Table 1. Time delay of various components in terms of number of *FO4* delays. f is the maximum fan-in of a gate and n is the number of inputs.

change is used instead of a complex relation between the exponent width and significand width in order to simplify the derivation of the delay estimates. In the following discussion, the symbol *expW* is used for the exponent width for the "traditional" designs and the symbol *expWF* is used for the design by Fahmy. Another simplification used is to ignore small increases in the operand width (for example by one bit due to recoding in the design of Nielsen) along the critical path of the design.

For the design by Nielsen, the exponent subtract takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil$ *FO4* delays. The significand swapping is done with a multiplexer that has the select lines coming from the exponent difference. So, $\lceil \log_4(n) \rceil + 1$ *FO4* delays is added for the swap. Next, the shifter takes $\lceil \log_2(n) \rceil$ *FO4* delays. The two $[4:2]$ compressors in the second cycle add 6 *FO4* delays. The adjust logic is not described in enough detail to make a good delay estimation. The adjust logic is based on the signs of the two numbers and the difference of the exponents, hence we assumed it to take only 2 *FO4* delays. The adjustment itself is a multiplexer whose select lines come from the adjust logic and takes $\lceil \log_4(n) \rceil + 1$ *FO4* delays. The signed sticky computation of the third cycle uses a tree of multiplexers[9] and hence takes $\lceil \log_2(n) \rceil$ *FO4* delays. The rounding logic is not specified and it has 10 inputs along with the mode (four possible modes) and the sign of the result. The rounding logic is then assumed to take $\lceil \log_f(15) \rceil$ *FO4* delays. The final adder takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$ *FO4* delays. To sum all of this, the design by Nielsen has the following delay:

$$\begin{aligned} \tau_{Nielsen} &= 20 + \lceil \log_f(15) \rceil + 2 \times \lceil \log_4(n) \rceil + 2 \times \lceil \log_2(n) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil \end{aligned}$$

The design by Oberman includes the exponent subtract and the significand swap on the critical path, similar to the case for the design of Nielsen. This takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil$ and $\lceil \log_4(n) \rceil + 1$ *FO4* delays respectively. The shifter adds $\lceil \log_2(n) \rceil$ *FO4* delays. The half adder of the third cycle takes 2 *FO4* delays. Although not explicitly reported in the published paper, there must be a negation of the shifted operand in the case of a subtraction operation. This negation is assumed to take one *FO4* delay. The compound adder is assumed to take one *FO4* delay longer than a regular adder of the same width since it has an additional multiplexer to choose between the sum and the sum plus one. So the compound adder is assumed to take $6 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$ *FO4* delays. The final multiplexer to choose between the far and close path adds one more *FO4* delay. So, the design proposed by Oberman has the following delay:

$$\begin{aligned} \tau_{Oberman} &= 16 + \lceil \log_4(n) \rceil + \lceil \log_2(n) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil \end{aligned}$$

The critical path of the design by Seidel has two possibilities. The first is the 7 bit exponent difference (not the full 11 bit), the bitwise XOR, the shifter and finally the multiplexer (from inputs to output). This first option takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{7}{f} \rceil - 1) \rceil + 1 + \lceil \log_2(n) \rceil + 1$ *FO4* delays. The second possibility is the full

11 bit difference, the bitwise XOR, the OR tree, the delay from select lines to the output of the multiplexer. This latter option takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{11}{f} \rceil - 1) \rceil + 1 + \lceil \log_f(5) \rceil + \lceil \log_4(n) \rceil + 1$ *FO4* delays. The second option is slightly longer and is the one used for the delay calculations if the exponent width is assumed to be 11 bits. Obviously, if the exponent width is only 8 bits then the first option (assuming that the whole exponent difference is evaluated by one adder) is the one determining the critical path. The second cycle of this design starts with the half adder and the compound adder which both add $8 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$ *FO4* delays. The one bit location shifters are multiplexers whose select line is one of the bits produced by the compound adder, hence there is a delay of $\lceil \log_4(n) \rceil + 1$ *FO4* delays. The multiplexer used to choose between the two shifter outputs adds just one more *FO4* delay (the delay of its select line is in parallel with the delay of the select line of the previous shifters). Finally, there is a multiplexer to choose between the far and close path that adds one more *FO4* delay. To summarize, the design of Seidel has the following delay:

$$\begin{aligned} \tau_{Seidel,expW=11} &= 18 + \lceil \log_f(5) \rceil + 2 \times \lceil \log_4(n) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{11}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil \\ \tau_{Seidel,expW=8} &= 18 + \lceil \log_4(n) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{8}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil \end{aligned}$$

Similar to the designs of Nielsen and Oberman, the design of Smith starts as well with the exponent difference and the significand swap. This takes $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil$ and $\lceil \log_4(n) \rceil + 1$ *FO4* delays respectively. The shifter adds $\lceil \log_2(n) \rceil$ *FO4* delays. The negation for the case of subtraction takes one *FO4* delay. The prefix adder produces several outputs that pass through logic to set some fields in the case of an exception (zero, NaN, ...). This is assumed to add one *FO4* delay to that of the adder giving $6 + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil$ *FO4* delays. Then, there is a multiplexing stage to choose between the possible outcomes and a final multiplexer to choose between the far and near path. This is assumed to add two more *FO4* delays. The total delay for the design of Smith is thus:

$$\begin{aligned} \tau_{Smith} &= 15 + \lceil \log_4(n) \rceil + \lceil \log_2(n) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{expW}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{n}{f} \rceil - 1) \rceil \end{aligned}$$

The design by Fahmy starts with the exponent difference. This is a 15 bit adder and not an 11 bit one because of the special format used in this design. In fact, the exponent width in this format depends on the radix (equal to 2^r) chosen for the digits and not just on n as in the previous designs. In the following derivation, this dependence on r and n is ignored and $expWF$ is assumed to be 15 for $n > 24$ and 11 for smaller n. The significand in this format is also larger than the corresponding significand for the previous designs because of the redundancy. The significand width is $\lceil \frac{n}{r} \rceil \times (r+1) - 1$. The swapping multiplexers must be as wide as the signficand and the output of the exponent difference is used to drive the select lines. Up to this point, the delay is estimated to be $5 + 2 \times \lceil \log_{f-1}(\lceil \frac{expWF}{f} \rceil - 1) \rceil + \lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil + 1$ *FO4* delays. The operand then passes through a $\lceil \frac{n}{r} \rceil$-way shifter which adds $\lceil \log_2(\lceil \frac{n}{r} \rceil) \rceil$ *FO4* delays. The following multiplexer adds one more *FO4* delay. The signed digit adder takes $8 + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil + \lceil \log_4(r+1) \rceil$ *FO4* delays. The select lines of the last multiplexer partially depend on the output of the adder in order to determine if there is a need to adjust to the right by one bit. Hence, there is a delay from the select lines to the output equal to $\lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil + 1$ *FO4* delays. The total delay for this design is thus:

$$\begin{aligned} \tau_{Fahmy} &= 16 + 2 \times \lceil \log_4(\lceil \frac{n}{r} \rceil \times (r+1) - 1) \rceil + \lceil \log_2(\lceil \frac{n}{r} \rceil) \rceil \\ &\quad + 2 \times \lceil \log_{f-1}(\lceil \frac{expWF}{f} \rceil - 1) \rceil + 2 \times \lceil \log_{f-1}(\lceil \frac{r+1}{f} \rceil - 1) \rceil + \lceil \log_4(r+1) \rceil \end{aligned}$$

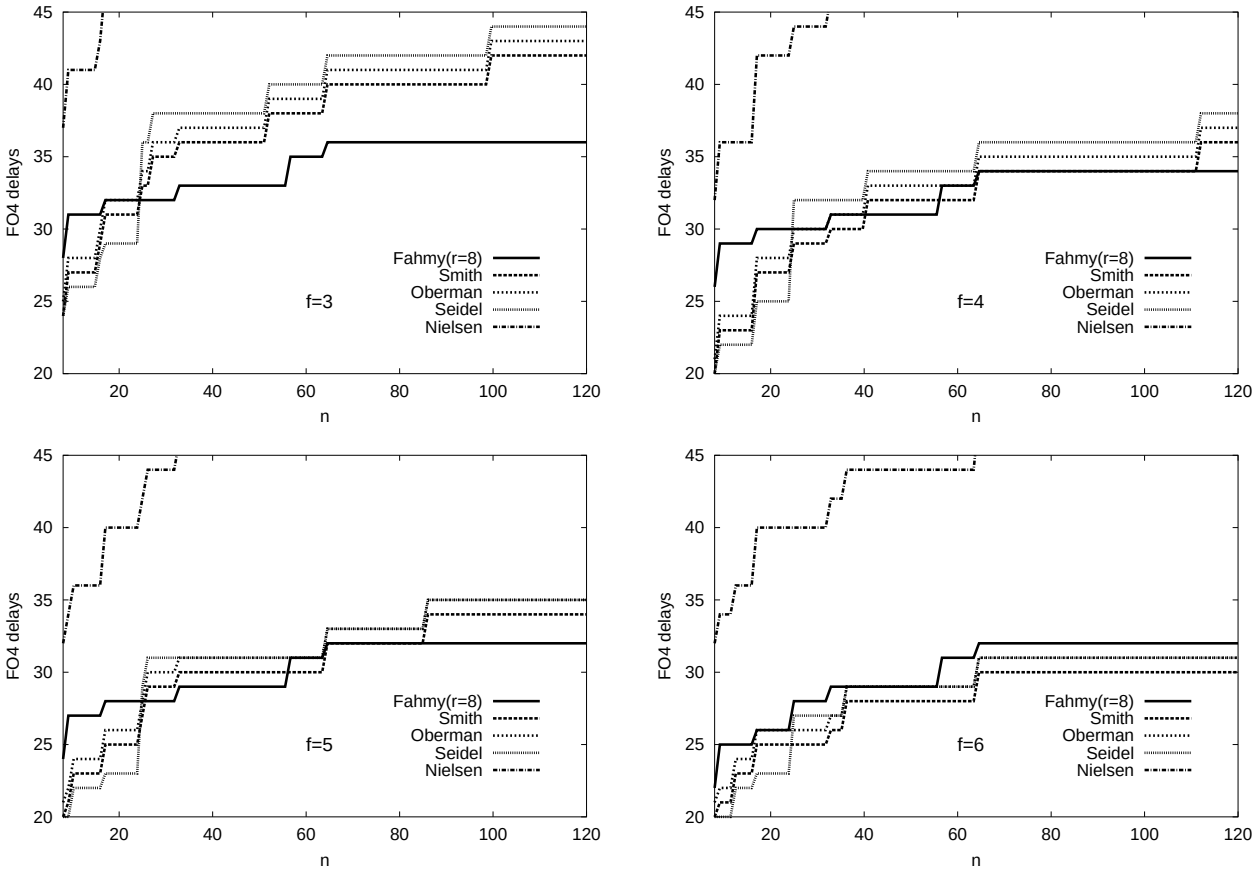

Figure 1. Time delay versus significand width for different fan-in values.

Figure 1 shows the time delay of the different designs when the significand width varies from 8 to 120 bits for different values of fan-in from $f = 3$ to $f = 6$. For the design by Fahmy, r is kept constant at $r = 8$. It is clear that the redundancy in the design of Fahmy makes it the fastest design for smaller fan-in values and large significand width. This is intuitively meaningful since for large significand widths the other designs suffer from a long time delay due to the longer carry propagation in the adders. As the fan-in increases, the long carry delays can be made better by using larger groups of bits in the conditional-sum or carry-lookahead adders. Hence, the improvements in performance due to the redundancy become less important and the overhead due to the larger significand size make the design with redundancy less desirable.

In the design proposed by Fahmy, considering the dependence of the exponent width on the radix and the other blocks (specially the shifter), the practical values for r should be multiples of 2. In order to minimize the additional cost of the redundancy (register storage, extra hardware,...) a large r is desirable. However, increasing r reduces the redundancy available and increases the time delay.

In Fig. 2, a comparison is presented between having $r = 4$ and $r = 8$ with the fan-in being either 3 or 4. Having a larger fan-in obviously improves the performance but it also decreases the relative advantage of this design compared to the other conventional ones. To clarify this point further, Table 2 compares the design proposed by Fahmy to that of Smith at $n = 80$ showing the relative improvement and the effect of r on the improvement.

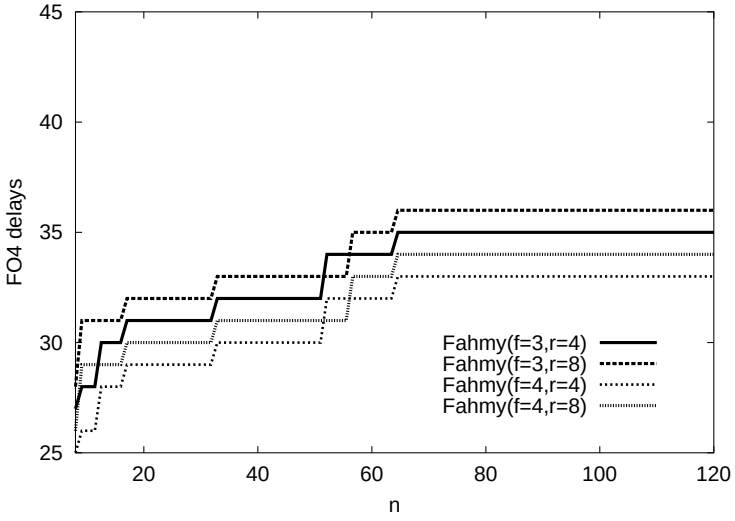

Figure 2. Comparison between $r = 4$ and $r = 8$ at $f = 3, 4$.

	$f = 3$		$f = 4$	
	$r = 4$	$r = 8$	$r = 4$	$r = 8$
Smith	40		34	
Fahmy	35	36	33	34
Relative improvement	$5/40 = 12.5\%$	10%	2.94%	0%

Table 2. Effect of r on the relative improvement for $n = 80$.

5. CONCLUSIONS

A parametric model for the time delay estimation of floating point units is presented. The time delay model accuracy is validated using *irsim*. The model then is used to compare different floating point adder designs for a large range of parameters. The time delay model can be used by designers and researchers to focus their efforts in the most profitable direction. Other parametric models that estimate power consumption and implementation area are also very important and should be considered for future extension to the current work.

For practical CMOS designs, the fan-in is usually limited to 3 or 4. The majority of the floating point adders are currently designed to handle double precision numbers ($n = 53$) or larger. For this range, the design proposed by Fahmy provides the best performance. The benefit of choosing a lower radix for the adder proposed by Fahmy is also highlighted.

REFERENCES

1. "IEEE standard for binary floating-point arithmetic," Aug. 1985. (ANSI/IEEE Std 754-1985).
2. "IEEE standard for radix-independent floating-point arithmetic," Oct. 1987. (ANSI/IEEE Std 854-1987).
3. B. Parhami, "Generalized signed-digit number systems: A unifying framework for redundant number representations," *IEEE Transactions on Computers* **39**, pp. 89–98, Jan. 1990.
4. A. M. Nielsen, D. W. Matula, C. N. Lyu, and G. Even, "An IEEE compliant floating-point adder that conforms with the pipelined packet-forwarding paradigm," *IEEE Transactions on Computers* **49**, pp. 33–47, Jan. 2000.
5. S. F. Oberman and M. J. Flynn, "Reducing the mean latency of floating-point addition," *Theoretical Computer Science* **196**, pp. 201–214, 1998.
6. A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency ieee floating-point standard adder architectures," in *Proceedings of the 14th IEEE Sympsoium on Computer Arithmetic, Adelaide, Australia*, pp. 35–42, Apr. 1999.

7. P.-M. Seidel and G. Even, "How many logic levels does floating-point addition require?," in *Proceedings of the International Conference on Circuit Design*, pp. 142–149, 1998.
8. H. A. H. Fahmy, A. A. Liddicoat, and M. J. Flynn, "Improving the effectiveness of floating point arithmetic," in *Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA*, **1**, pp. 875–879, Nov. 2001.
9. D. W. Matula and A. M. Nielsen, "Pipelined packet-forwarding floating point: I. foundations and a rounder," in *Proceedings of the 13th IEEE Sympsoium on Computer Arithmetic Asilomar, CA, USA*, pp. 140–147, July 1997.
10. A. M. Nielsen, D. W. Matula, C. N. Lyu, and G. Even, "Pipelined packet-forwarding floating point: II. an adder," in *Proceedings of the 13th IEEE Sympsoium on Computer Arithmetic Asilomar, CA, USA*, pp. 148–155, July 1997.
11. S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehard & Winston, New York, 1982.
12. V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Transactions on Computers* **45**, pp. 294–306, Mar. 1996.
13. G. W. McFarland, *CMOS Technology Scaling and Its Impact on Cache Delay.* PhD thesis, Stanford University, June 1997.