Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic

Jonathan Ying Fai Tong, *Member, IEEE*, David Nagle, *Member, IEEE*, and Rob A. Rutenbar, *Fellow, IEEE*

***Abstract*—Low-power systems often find the power cost of floating-point (FP) hardware prohibitively expensive. This paper explores ways of reducing FP power consumption by minimizing the bitwidth representation of FP data. Analysis of several FP programs that manipulate low-resolution human sensory data shows that these programs suffer no loss of accuracy even with a significant reduction in bitwidth. Most FP programs in our benchmark suite maintain the same output even when the mantissa bitwidth is reduced by half. This FP bitwidth reduction can deliver a significant power saving through the use of a variable bitwidth FP unit. Our results show that up to 66% reduction in multiplier energy/operation can be achieved in the FP unit by this bitwidth reduction technique without sacrificing any program accuracy.**

***Index Terms*—Analysis, computer-arithmetic, digital CMOS, low-power design, special low-power99.**

I. Introduction

FLOATING-POINT (FP) hardware provides a wide dynamic range of representable real numbers, freeing programmers from writing the cumbersome manual scaling code required of fixed-point representation. Unfortunately, FP hardware is very power hungry, with FP multipliers in particular being expensive components in a processor's power budget. This has limited the use of FP hardware, with most low-power processors not including any FP hardware. However, for an increasing number of embedded applications, such as voice recognition and image/vision processing, FP hardware's performance, simplified programming model, and adaptability over a wide dynamic range makes it a desirable feature.

One important characteristic of many of these mobile/portable applications is that they work primarily with low-resolution (4–10 bits) human sensory data, notably digitized speech and images. FP arithmetic allows many signal processing algorithms to process these acquired voice or vision data without concern about either the precision or range of intermediate results. Indeed, it remains common design practice [26] to prototype these algorithms in FP, then (usually manually) to translate them into an appropriate set of integer operations. In this study, we suggest that instead of rendering these computations as integer operations, we should translate them into custom-precision FP.

Specifically, we examine how software can employ the *minimal* number of mantissa and exponent bits in FP hardware to *reduce* power consumption, yet *maintain* a program's overall accuracy. We study experimentally the relationship between the accuracy of FP programs and the number of bits used in the representation of their data. Our central results demonstrate that many programs which manipulate human sensory inputs, e.g., speech and image recognition, suffer no loss of accuracy with reduced bitwidth in the mantissa or exponent. We use this result to explore various methods to minimize power dissipation in the FP unit. The fundamental principle is to reduce waste—in this case, unnecessary bits in the FP representation and computation.

Our results show that programs such as speech recognition and image processing use significantly less power with our reduced bitwidth FP representation than with an IEEE-standard FP representation. For example, the speaker-independent continuous-speech recognition program Sphinx [10] requires only 5 mantissa bits and 6 exponent bits to maintain 90% recognition accuracy—the same level of accuracy achieved with a 32-bit IEEE-standard FP representation. Compared to a conventional 32-bit FP multiplier, which uses 2078 pJ per multiplication in our implementation, our reduced bitwidth FP multiplier uses only 705 pJ, roughly one third the energy. It has long been known that many such signal processing applications can get by with less precision/range than full FP. Our central contribution here is precise documentation of the extremity of such allowable reductions across several real benchmarks. In many cases, fewer the half the bits, in an appropriate custom FP format, suffice.

The remainder of the paper is organized as follows. Section II provides background material on FP representation and algorithms. Section III constructs an ASIC-style FP multiplier for use as a baseline in subsequent experiments, and characterizes the power dissipation of its component pieces. Section IV offers a taxonomy of custom FP format variations, categorized by their potential impact on power. Section V describes experimental results from analysis of several real benchmark programs executing (in emulation) with reduced precision/range FP formats. This section also examines the utility of digit-serial variable-precision multipliers as a possible compromise when low-precision operands dominate, but higher precision operands cannot be entirely avoided. Section VI describes related work and examines in particular how transition-gating schemes recently proposed for integer arithmetic optimization perform in our FP workloads. Finally, Section VII offers some concluding remarks.

Manuscript received February 13, 1999; revised September 22, 1999. This work was supported in part by the National Science Foundation and by DARPA under Contract DAAL01-95-K-3527.

J. Y. F. Tong was with the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213 USA. He is now with the M-Core Technology Center, Motorola, Austin, TX 78731 USA (e-mail: Jonathan.Tong@mot.com).

D. Nagle and R. A. Rutenbar are with the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213 USA (e-mail: dnagle@ece.cmu.edu; rutenbar@ece.cmu.edu).

Publisher Item Identifier S 1063-8210(00)04346-8.

II. Background

A. FP Representation Versus Fixed-Point Representation

FP representation is by far the most widely used representation for real numbers. FP numbers provide a wide dynamic range of representable real numbers, freeing programmers from the manual scaling code necessary to support fixed-point operations. For instance, an IEEE single-precision number and an integer on a 32-bit machine both require 32 bits of storage for their representation. However, the dynamic range provided by the FP representation ($1.999\,999\,88 \times 2^{-126}$ to $1.999\,999\,88 \times 2^{127}$) is substantially wider than that provided by the integer representation ($1–2^{31}$).

On the other hand, FP hardware is very power hungry. In addition to performing the basic addition and multiplication operations as in the integer unit, the FP unit must process the exponents of the input operands as well as provide rounding to the final result. Typically, FP multipliers are expensive components in a processor's power budget. This has limited the use of FP operations in embedded systems, with many low-power processors not including any FP hardware. For example, recent digital signal processors (DSP's) introduced with FP hardware support seem to have met with limited success in the signal processing community [26].

For an increasing number of embedded applications such as voice recognition, vision/image processing, and other human-sensory-oriented signal-processing applications, FP's simplified programming model (in contrast with fixed-point systems) and large dynamic range makes FP hardware a desirable feature. Furthermore, many recognition applications achieve a high degree of accuracy starting from fairly low-resolution input sensory data. Leveraging these characteristics by allowing software to use the minimal number of mantissa and exponent bits, standard FP hardware can be modified to significantly reduce its power consumption while maintaining a program's overall accuracy.

B. The IEEE 754 Standard

There are two different IEEE standards for FP computation. IEEE 754 is a binary standard that requires the implied radix (base) to be two. IEEE 854 allows either radix 2 or radix 10 representation. By far, IEEE 754 is more popular and most desktop microprocessors support the IEEE 754 standard.

One of the main concerns of the IEEE 754 Floating Point Standard [2] is the accuracy of arithmetic operations. IEEE-754 specifies that any single-precision FP number be represented using 1 sign bit, 8 bits of exponents, and 23 bits of mantissa. With double precision, the bitwidth requirements of exponent and mantissa go up to 11 and 53 bits, respectively. An IEEE single-precision number is represented in Fig. 1. Note that the exponent is a signed number represented using a bias method; for single-precision numbers the exponent is stored in excess-127 form.

In addition to specifying the bitwidth requirement for FP numbers, IEEE-754 incorporates several additional features, including delicate rounding modes and support for gradual underflow to preserve the maximal accuracy of programs. The implementation of an IEEE-compliant FP unit is not easy. In addition to the design complexity and the large area it occupies, an FP unit is also a major consumer of power in microprocessors. Many embedded microprocessors such as the StrongARM [3] and MCore200 [4] do not include an FP unit due to its significant implementation cost.

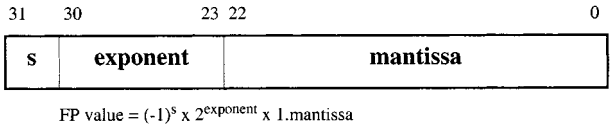


Fig. 1. Representation of single-precision IEEE FP number.

III. Power Comsumption of a Single-Precision FP Multiplier

To attack the power problem of the FP unit, we need a reasonable estimate of the power dissipation in different blocks of the unit. Since multiplication is one of the most frequent operations in signal processing applications and we claim that the multiplier is by far the most power hungry arithmetic unit, this section dissects the power dissipation of a single-precision FP multiplier. Subsequent sections use this FP multiplier as an illustration of our power reduction techniques, but the techniques apply to other blocks of the FP unit as well.

The FP multiplier (Fig. 2) can be divided into three major functional blocks: 1) the mantissa multiplier; 2) the exponent unit; and 3) the rounding unit (including normalization). Fig. 3, which is based on [5], shows one possible mantissa data flow for an FP multiplier. The top section (Multiply) accepts two normalized mantissas and uses a Wallace-tree reduction structure that produces the product in carry-save form (two $2n$ bit numbers). These two numbers are then added in the carry-propagate adder (CPA) section to produce a complete $2n$ bit product. If the resulting product is in the range $2 \leq$ product < 4, we have a potential overflow and the constant $2^{(-n+1)}$ is added to the product and the result is truncated to $n-2$ bits to the right of the decimal point. A normalization shift (Normal) of 1 bit to the right is then necessary to restore the rounded product to the range $1 \leq$ rounded product < 2, with an appropriate adjustment of the exponent. If the original $2n$ bit product is in the range $1 \leq$ product < 2, then we need only add the constant $2^{(-n)}$. Furthermore, if the addition of $2^{(-n)}$ causes the rounded product to be equal to two, then a normalization shift of 1 bit and an exponent adjustment are necessary.

To estimate the power consumed in each of a multiplier's three main blocks (i.e., mantissa multiplier, exponent unit, rounding unit), we have adopted a standard ASIC-style design methodology and constructed a complete—though minimally optimized—FP multiplier. We synthesized a behavioral model of an FP multiplier using a standard cell-based design flow and then performed power simulation using a transistor-level simulator. Even though most FP units in microprocessors and DSP are custom-designed, our design provides usable estimates of the relative power dissipations among the different blocks inside an FP multiplier.

Our behavioral model of the FP multiplier was written in Verilog and based on the design of the Aurora III processor developed at the University of Michigan [6]. The model was

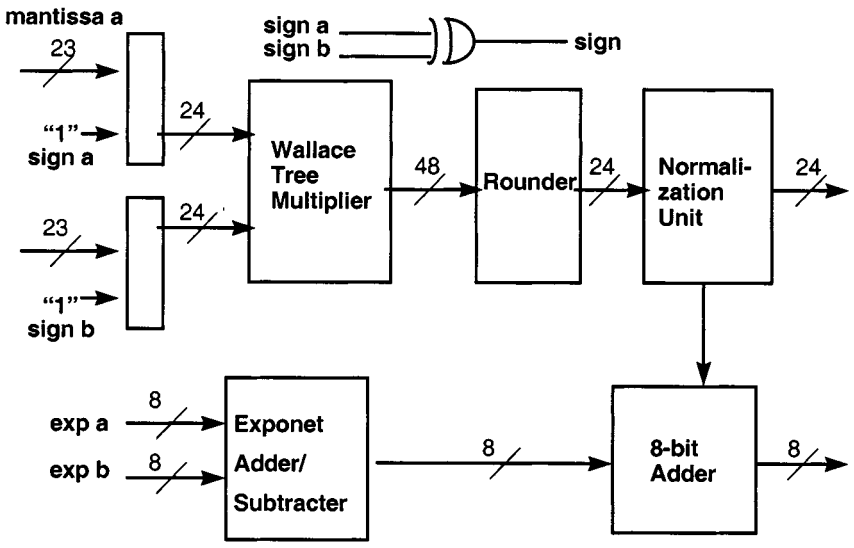


Fig. 2. Block diagram of a single-precision FP multiplier.

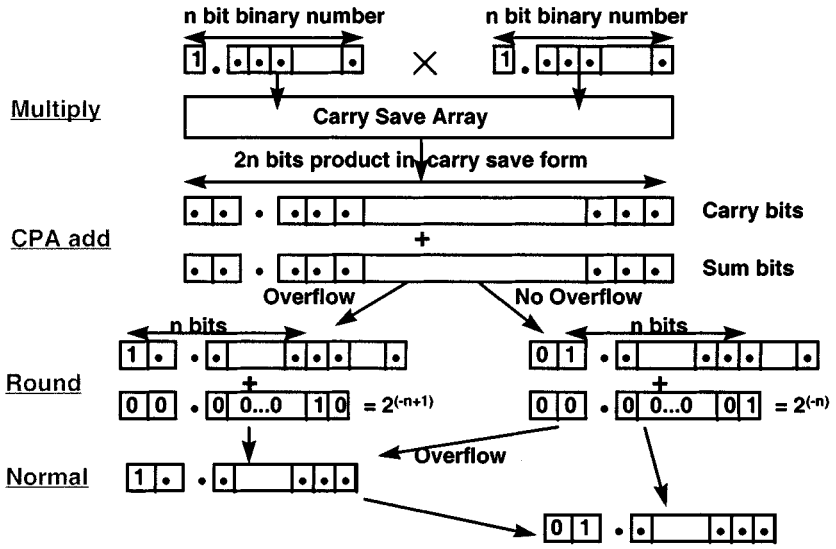


Fig. 3. Mantissa data flow of an FP multiplier.

synthesized using Synopsys' Design Compiler tool and Foundation Library. Then, the synthesized netlist was fed into Duet's Epoch physical design tool to generate the physical design in a 0.5-μm CMOS process. The layout of the FP multiplier appears in Fig. 4. A SPICE netlist with all the capacitance information back-annotated was read into Synopsys' PowerMill tool to perform full transistor-level power simulation. Determining the complete power dissipated in a multiplier requires the sensitization of all possible combinations of inputs, which means we would need to have 2^{2N} input combinations where N is the number of inputs. Fortunately, it is possible to estimate the *mean* power consumption using statistical techniques [13]. Applying these ideas, we simulated 50 batches of vectors with each batch containing 30 vectors in the Synopsys PowerMill tool; this was sufficient to ensure a 95% confidence interval. The correct cycle time was obtained from static timing analysis. The test vectors were taken from streams of actual multiplication operands found in the Sphinx III speech recognition application (discussed in Section IV-A).

Table I summarizes the average power consumption of the FP multiplier. The mantissa multiplier completely dominates the power, followed by the rounding unit which accounts for roughly one sixth of the power. We note that the power dissipation does vary significantly for different data sets; for example, in another experiment with just 100 random vectors the multiplier power consumption dropped to roughly two thirds of the total power and the rounder increased to about one quarter of the power. Nevertheless, the mantissa multiplier always consumes more power than all other blocks.

We interpret these results as follows. First, it is clear that a custom-designed FP unit, in contrast to our ASIC-style design, will have superior area, speed, power. Nevertheless, we argue that our ASIC experiment validates one clear design assumption: the mantissa multiplier itself dominates the power in an

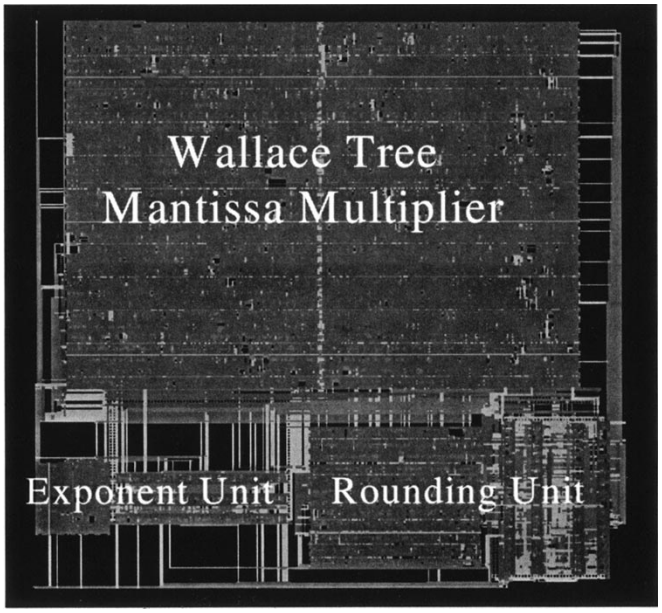


Fig. 4. Layout of the single-precision FP multiplier. This FP multiplier was synthesized and laid out using a conventional cell-based ASIC flow in a 0.5-μm CMOS process. The size of the multiplier is 1.09 $\times$ 1.15 sq mm.

FP multiplier. To reduce power dissipation, we should focus our efforts here. And, if we agree to interpret our data a bit more closely, the rounding unit is always the next most significant power consumer. We conclude that reducing power dissipation in these two units (multiplier, rounder) should be our priority. In the following sections, we examine different power reduction techniques specifically targeted at these two functional blocks.

IV. Attacking FP Power Dissipation at a System Level

Considering the IEEE representation of FP numbers as described in Fig. 1, there are four dimensions that we can explore to reduce power dissipation, which are enumerated in Table II. We regard this as a rough taxonomy of format versus power tradeoffs. The question we seek to address is: what are the *quantitative* tradeoffs between program accuracy and the power consumption for the FP unit?

A. Change of Implied Radix

The implied radix of the IEEE 754 standard is two. Nevertheless, a different implied radix, usually a higher radix that is a power of two, can be used. For example, historically, the IBM 360 mainframes used a radix of 16 to reduce the complexity of shift logic [25].

From a power perspective, a higher radix means less types of shifts and (potentially) fewer normalization shifts. There are basically two types of shift operations in FP operations: normalization shifts and prealignment shifts. Normalization shifts are needed after any arithmetic operations to put the final representation in a canonical format. Prealignment shifts are needed only in FP additions and subtractions to ensure the two operands have the same exponent. Other researchers [7] have extensively evaluated the best shifter configuration for doing prealignment shifts.

As a preliminary experiment to assess the impact of using a higher radix, we used a software FP emulator to measure the number of normalization shifts when different radices are used. Our benchmarks were a large set of signal processing applications focusing on human sensory data, described in Section IV-A. The result shows that the number of normalization shifts is reduced by less than 10% for base 16, suggesting the potential for only a modest power reduction in the normalized shifter. We conclude that changing the implied radix is not a high-priority target for power optimization.

On the other hand, there does exist potential power savings in the prealignment shifter. Both [7], [8] show that a log shifter has a lower energy-delay product. For single-precision addition and subtraction, three muxes are needed in series for radix 2 representation while only two muxes are needed for radix 16 representation. This reduces the area, delay, and power of the barrel shifter.

B. Simplifications of Rounding Modes

Whenever a FP number cannot be exactly represented, the number is rounded, introducing an error that is less than the value of the least significant bit (depending on rounding method). Because this error can be significant, especially for small numbers or when accumulated over a series of computations, the IEEE FP standard specifies four rounding modes for programmers to choose from. The default rounding mode is "round to nearest/even," which means always round to the nearest representable number, and in the case of a tie, round to the even representation. This rounding mode often involves a carry propagate addition that lies in the critical path of most FP operations. The other three optional modes are: "round to plus infinity," "round to minus infinity," and "chopped" (i.e., round-to-zero) which is equivalent to truncation. The hardware required to support the first two optional rounding modes is similar to the default "round to nearest," so there is little tradeoff in terms of power among these three rounding modes.

Chopped-mode (i.e., truncation), however, is an excellent candidate for potential power reduction. As Table I shows, the rounding unit (using the default rounding mode) consumes about one sixth of the FP multiplier's total power when running operands from a speech recognition benchmark. Using chopped-mode would eliminate almost all of this power consumption. However, truncating all results could compromise the final output accuracy of a program. As a simplistic initial experiment to assess the impact that chopped-mode has on accuracy, we evaluated the output of the SPEC95 benchmark suite [27], [28] compiled to use "chopped" rounding mode. Somewhat surprisingly, results show that seven out of the eight SPEC95fp double-precision benchmarks are independent of the rounding modes used (Table III). Therefore, disabling the rounding unit (e.g., by clock gating) could reduce total power.

Of course, we must emphasize that the effect of rounding will depend *strongly* on the input set and application. In mission critical applications, a sensitive numerical analysis of the required precision will be required. However, the experiment does suggest that there may be situations where a program's output is insensitive to the rounding modes and thus rounding may be

TABLE I
POWER CONSUMPTION OF SINGLE PRECISION FP MULTIPLIER

Functional Block	Power Dissipation (% of total)
Mantissa Multiplier	81.2
Rounding Unit	17.9
Exponent Unit	0.833
Others (Exception handling etc.)	0.066

TABLE II
DESIGN DIMENSIONS FOR FP REPRESENTATION

Dimension	Description
Change of the implied radix	Increase the implied radix from 2 to 4 (or 16). This provides greater dynamic range but lower density of floating point numbers, potentially leading to power savings since fewer normalizing shifts are necessary.
Simplification of rounding modes	Full support of all the rounding modes is very expensive in terms of power. Some programs may achieve an acceptable accuracy with a modified low power rounding algorithm.
Reduction in mantissa bitwidth	Reduce the number of mantissa bits at the expense of precision.
Reduction in exponent bitwidth	Reduce the number of exponent bits at the expense of a smaller dynamic range.

TABLE III
EFFECTS OF NO ROUNDING ON SPEC95 PROGRAMS. THE SPEC95 FP APPLICATIONS WERE COMPILED ON DIGITAL UNIX 4.0 USING f70. AN EXTRA COMPILER FLAG "FPRM CHOPPED" IS ADDED SO TRUNCATION IS PERFORMED INSTEAD OF ROUNDING. "RUNSPEC-DFP" IS USED TO CHECK THE OUTPUT RESULTS

Benchmark	Inputs	# of FP operations	Correct Output
mgrid	mgrid.in	52,692 M	Yes
hydro2d	hydro2d.in	18.739 M	Yes
applu	applu.in	18,463 M	Yes
turb3d	turb3d.in	35,596 M	Yes
fpppp	natoms.in	64,807 M	Yes
wave5	wave5.in	11,435 M	No
tomcatv	tomcatv.in	14,332 M	Yes
su2cor	su2cor.in	14,630 M	Yes

disabled—perhaps at the discretion of intelligent software control—to minimize power.

C. Reducing Exponent and Mantissa Bitwidth

The most intuitive way of reducing the complexity of FP operations will be to reduce the number of bits in the FP representation. According to [1] and [9], a multiplier's power consumption decreases rapidly with the operand bitwidth. Therefore, reducing the operand bitwidth might provide huge opportunities in minimizing power.

Consider again a typical FP format: one sign bit, a relatively large fractional mantissa field, a relatively smaller exponent field. Since the sign is only 1 bit, we only examine the possibility of reducing the number of bits used in the exponent and the mantissa. As the following sections reveal, reduction in the mantissa bitwidth is the most effective means of reducing power dissipation in FP datapath elements.

TABLE IV
DESCRIPTION OF FP WORKLOADS

Workload	Description	Accuracy Measurement
Sphinx III	CMU's speech recognition program based on fully continuous hidden Markov models. The input set is taken from the DARPA evaluation test set which consists of spoken sentences from the Wall Street Journal. [10] Source code size: 15,623 lines.	Accuracy is estimated by dividing the number of words recognized correctly by the total number of words in the input set.
ALVINN	Taken from SPECfp92. A neural network trainer using backpropagation. Designed to take input sensory data from a video camera and a laser range finder and guide a vehicle on the road. Source code size: 314 lines.	The input set consists of 50 road scenes and the accuracy is measured as the number of correct travel direction decisions made by the network.
PCASYS	A pattern-level finger print classification program developed at NIST. The program classifies images of fingerprints into six pattern-level classes using a probabilistic neural network. Source code size: 11,424 lines.	The input set consists of 50 different finger print images and the classification result is measured as percentage error in putting the image in the wrong class. The accuracy of the recognition is simply (1 - percentage error).
Bench22	An image processing benchmark which warps a random image, and then compares the warped image with the original one. Source code size: 926 lines.	Percentage deviation from the original outputs is used as a measure of accuracy.
Fast DCT	A direct implementation of both 2-dimensional forward Discrete Cosine Transform (DCT) and inverse DCT on blocks of 8x8 pixels. Source code size: 1059 lines.	100 random blocks of 8x8 pixels are transformed by forward DCT and then recovered by inverse DCT. Accuracy measured as percentage of correctly recovered pixels.

V. MINIMIZING POWER VIA BITWIDTH REDUCTION

It is obvious that power dissipation in an FP unit can be reduced by using fewer bits in the FP representation. However, fewer bits reduces precision and might result in a less accurate output. In this section, we first measure empirically the bitwidth requirements of several real programs, then suggest techniques for minimizing power dissipation by using smaller bitwidth functional units.

A. Workloads and Methodology

To study the relationship between program accuracy and number of bits in FP representation, we have collected a set of five signal processing applications. These programs mainly deal with human sensory data such as digitized images and speech. All of them are single-precision FP programs and they are described in Table IV. As can be seen in the table, these range in complexity from single-purpose kernels to complete applications.

To determine the impact of different mantissa and exponent bitwidths, we emulated in software different bitwidth FP units by replacing each FP operation with a corresponding function call to our FP software emulation package that initially implements the IEEE-754 standard (Fig. 5). Careful modifications to the FP emulation package allowed us to emulate different mantissa and exponent bitwidths. Then, each program was run using the modified FP package, and the results were compared to determine application accuracy.

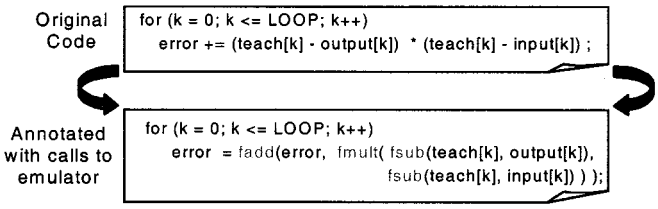


Fig. 5. FP emulation by annotating the source code.

B. Results

Fig. 6 plots the accuracy for each of the five programs across a range of mantissa bitwidths. None of the workloads display a noticeable degradation in accuracy when the mantissa bitwidth is reduced from 23 to 11 bits. For ALVINN and Sphinx III the results are even more promising; the accuracy does not change significantly with as few as 5 mantissa bits.

Fig. 7 shows that each program's accuracy has a similar trend when the exponent bitwidth is varied. With seven or more exponent bits, the error rates remain quite stable. Once the exponent bitwidth drops below six, the error rates increase dramatically and in some cases the programs could not finish properly.

These results clearly demonstrate that not all programs need the precision provided by generic FP hardware. The reason behind this result is that many programs dealing with human interfaces process sensory data with intrinsically low resolutions. Raw input data with 4–10 bits of precision is rather common in these applications. While intermediate results often require

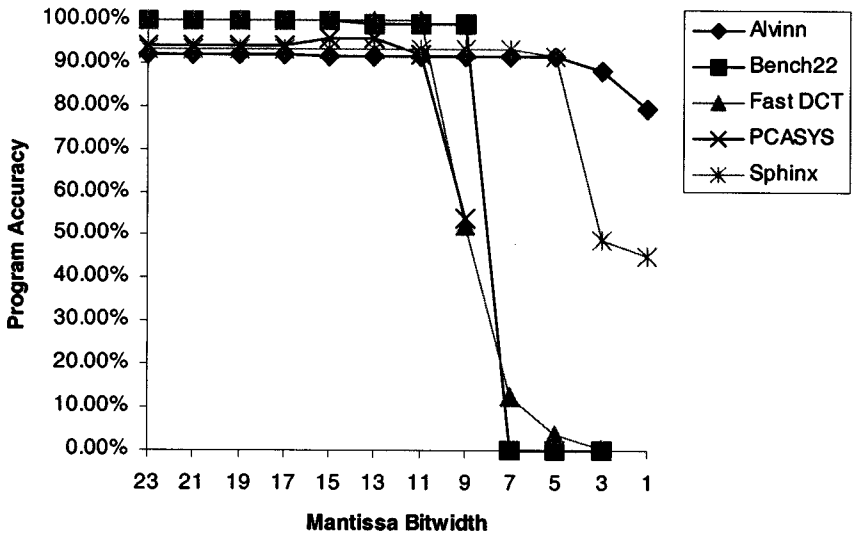


Fig. 6. Program accuracy across various mantissa bitwidths.

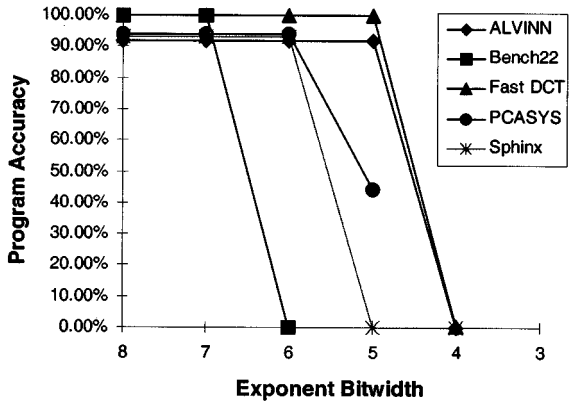


Fig. 7. Program accuracy across various exponent bitwidths. Figs. 6 and 7 show that we can reduce both the mantissa and exponent bitwidth without affecting the accuracy of the programs. This effect is especially prominent in the mantissa. This reduction of bitwidth can be turned into a reduction in power dissipation with the use of appropriate arithmetic circuits.

more dynamic range than is available with small bitwidth fixed-point computation, the programs do not require vastly more precision. This is different from scientific programs such as large-scale computational fluid dynamics or electrical circuit simulation, which not only require a huge amount of precision and dynamic range but also delicate rounding modes to preserve the accuracy of the results. What is quite clear from these experiments is that the FP format provides essential dynamic range (we can reduce, but not reduce dramatically, the number of exponent bits) but the fine precision of the 23-bit mantissa is not essential (half as many bits often suffice).

For programs that do not need the dynamic range or the precision of FP arithmetic, the use of fixed-point arithmetic will reduce chip area, operation latency, and power consumption. Indeed, common design practice for these applications is to prototype software in FP, then translate into a suitable "finite" fixed-point representation [26]. But for the workloads in Table IV, three programs require 6 bits or more in the exponent to preserve a reasonable degree of accuracy, which means they need more than the typical 32 bits of precision that fixed point arithmetic offers. Simply using fixed-point representation without additional (usually manual) scaling will not resolve the problem.

It should be noted that these complex applications were aggressively tuned by various software designers to achieve good performance using full IEEE representation. However, Figs. 6 and 7 show that significantly smaller bitwidth FP units can be used without compromising the necessary accuracy. For instance, certain FP constants in the Sphinx III code required more than 10 bits of mantissa to represent, but we modified those numbers so they could be represented using fewer bits during our experiment. Nevertheless (and perhaps somewhat surprisingly), these reductions in precision had little impact on the overall speech recognition accuracy. We believe that if the numerical behavior of these applications were tuned *explicitly* to a smaller bitwidth FP unit, we could obtain even better performance.

C. Different Bitwidth Requirements Within a Single Application

The results in the previous section assumed a single custom FP format used across all FP operations. Clearly, this is a rather strict assumption. This section analyzes changing bitwidth requirements *within* a single program. Clearly, different subroutines within one program may require different bitwidths to produce the expected results. To illustrate this, we use the Fast DCT program as an example.

The Fast DCT program naturally divides into two subprograms: the *forward* DCT and the *inverse* DCT. Forward DCT converts a block of 8×8 pixels into a block of transform coefficients, which represents the spatial frequency components of the original block. Inverse DCT converts these coefficients back to a block of 8×8 pixels.

Fig. 8 shows the mantissa bitwidth requirements for forward DCT, inverse DCT, and the entire Fast DCT program. When the mantissa bitwidth in the Forward DCT section is varied, we keep the mantissa bitwidth in the inverse DCT section at 23 bits. Results show that both the forward DCT and the entire Fast DCT program require 11 bits of mantissa to maintain 100% accuracy

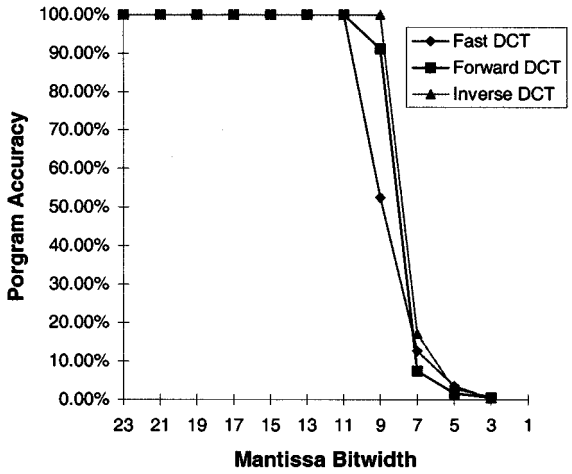


Fig. 8. Program accuracy versus mantissa bitwidth for Fast DCT.

while the inverse DCT requires only 9 bits. All three programs (or subprograms) have exactly the same behavior (same program accuracy) when the exponent bitwidth is varied from 7 to 3 bits.

Even though the mantissa bitwidth requirements of forward DCT and inverse DCT are almost the same, the result shows that different sections of a program may indeed have different bitwidth requirements. This can be exploited to reduce power dissipation: when forward DCT is being executed on a processor, an 11-bit mantissa multiplier/adder is required, while a 9-bit mantissa multiplier/adder will be enough for inverse DCT operations.

Since both the forward DCT and inverse DCT use the same basic algorithm, their bitwidth requirements differ only a little. For programs that integrate a wider portfolio of different algorithms, we expect a wider range in the bitwidth requirements within the program.

D. Exploiting a Variable Bitwidth Multiplier

Our results show that different FP programs exhibit differing requirements for mantissa and exponent bitwidths. Hence, by reducing either precision or range from 32 to fewer bits, we should be able to create custom FP hardware which has lower power simply because of the bit reductions. For a narrow, application-specific task, a single custom FP format may be a viable option. However, to be more generally useful, we need to consider arithmetic architectures which can scale to different FP formats. Even though we may be able to assume that most of our operands can be computed successfully in limited precision, it appears inevitable that some fraction of our operands will require full IEEE-standard precision. As one potential solution, we explore in this section one architecture for a *variable bitwidth FP multiplier* to reduce power consumption. To support variable bitwidth multiplications (up to 24×24 bit), we considered a 24×8 bit *digit-serial* architecture similar to the one described in Hartley and Parhi [12]. The 24×8 bit architecture (see Fig. 9) allows us to perform 8-, 16-, and 24-bit multiplication by passing the data once, twice, or three times though the serial multiplier. A finite state machine is used to control the number of iterations through the core multiplier. The digit size of the digit-serial architecture does not have to be fixed. We chose a digit size of 8 bits because most processors support bus widths which are multiples of eight. It is already well understood that digit-serial arithmetic offers some power advantages (e.g., [14] offers a comparative analysis of power consumption among different digit-serial architectures). Hence, digit-serial multiplication seems well suited to our application.

To perform accurate comparisons, a complete 24×8 digit-serial multiplier was modeled in Verilog and then taken to layout using our previously described ASIC design flow, using a standard 0.5-μm CMOS process. Synopsys' Design Compiler tool was used to synthesize the multiplier's control logic; the complete physical design was done using Duet's Epoch layout tool.

We compare our variable bitwidth multiplier with a baseline fixed-width 24×24 bit Wallace tree multiplier. The layout of this Wallace tree multiplier was generated by Epoch's procedural cell generator in the same 0.5-μm process used in the design of the digit-serial multiplier. The two multipliers are described in Table V. Cycle time is estimated using Epoch's static timing analysis tool, Tactic, and is rounded to the nearest 5-ns interval for simplicity. The lower precision digit-serial design is slightly larger, since it includes control logic, and was also created in an ASIC style, in contrast to the larger Wallace tree, which is from an optimized layout compiler. Nevertheless, the digit-serial design is considerably faster, since the lowered precision (8×24 versus 24×24) creates shallower logic.

To compare energy or power, we need to make several assumptions. Assuming to first order that the multiplier core itself is the dominant energy consumer, we will focus here. Also, we do not consider here any global cycle-time impacts. Clearly, if we replace a 24×24 multiplier with a faster 8×24 multiplier, for 8- and 16-bit operands, our FP multiplier may well consume less of the overall cycle time. We might be able to shorten the cycle time if the FP multiplier is on the critical path, and if we know that 24×24 multiplications are rare, we can simply mandate an extra cycle. (Note again from Table V that the digit serial multiplier is only advantageous for precisions less than full 24×24 bit multiplication.) But, an alternative scenario is to leave the cycle time constant, and simply slow down the lower precision multiplier (via circuit redesign, e.g., [29]) to the same delay as the full-precision multiplier. Operating at lower precision and lower speed, this multiplier will consume even less power. We choose for now to avoid these questions and tradeoffs and simply focus on the multiplier cores themselves. To sidestep these cycle time issues, we choose to focus on multiplier *energy/operation* as our metric of comparison.

Our specific goal is a first-order analysis of the energy impact of reducing mantissa precision. To do this, we look first at energy/operation as the comparison metric and assume each multiplier simply operates at the speeds determined via static timing analysis in Table V. For each design, a SPICE netlist was generated from layout and used to estimate energy/operation, using the same statistical methodology as was used for the baseline FP multiplier in Section III (i.e., 95% confidence intervals, stimuli from the Sphinx III speech workload).

Fig. 10 plots the energy/operation and latency/operation for the digit-serial multiplier. Both the energy/operation and latency/operation increase linearly with the operand bitwidth.

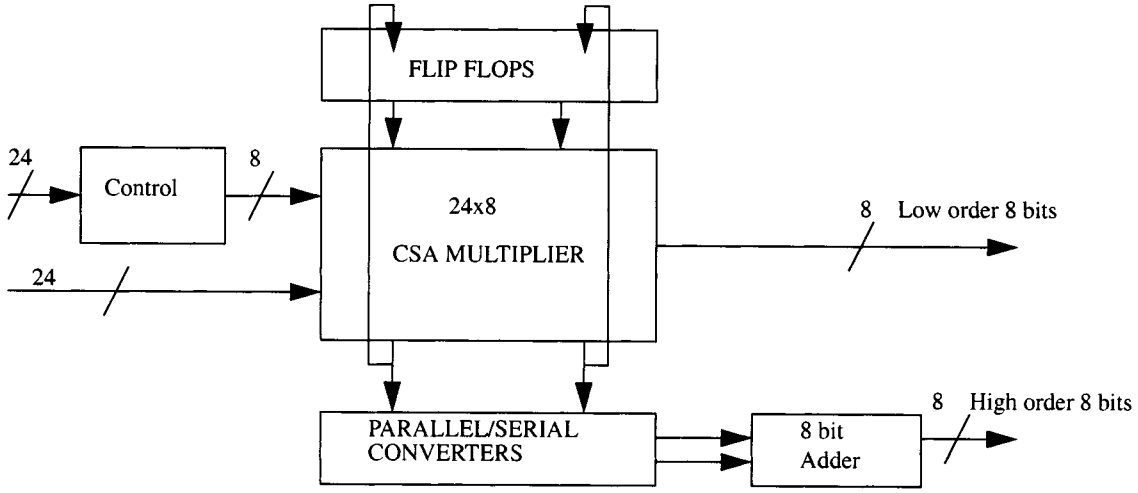


Fig. 9. Block diagram of a 24 × 8 digit-serial multiplier.

TABLE V
TIMING AND AREA OF THE TWO MULTIPLIERS. TO PERFORM 16-BIT MULTIPLICATION USING THE DIGIT-SERIAL MULTIPLIER, TWO CYCLES ARE NEEDED WHICH INCREASES THE TOTAL DELAY/OP TO 30 ns. SIMILARLY, 24-BIT MULTIPLICATION TAKES THREE CYCLES (45 ns)

Multiplier	Area	Cycle Time	Latency/op
Wallace (24x24)	.777square mm	40ns	40ns
Digit-Serial (24x8)	.804 square mm	15ns	15ns

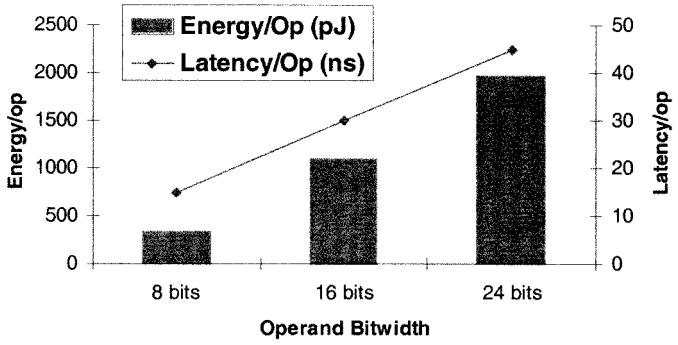


Fig. 10. Performance of the digit-serial multiplier. Both energy/op and latency/op of the digit-serial multiplier increase linearly with the operand bitwidth. Digit-serial architecture allows us to perform variable bitwidth arithmetic and save power when the bitwidth requirement is less than that specified in the IEEE standard.

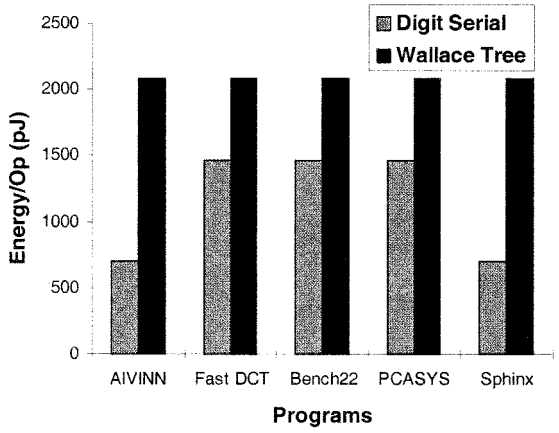


Fig. 11. Estimated multiplier energy reduction using digit-serial multiplier. A reduction of up to 66% in energy/op is attainable for Sphinx and ALVINN with the use of a digit-serial multiplier. Both Sphinx and ALVINN need only 5 bits of mantissa to be 90% accurate, and thus an 8-bit operand bitwidth is used for the digit-serial multiplier. PCASYS requires 11 bits of mantissa while Bench22 requires 9 bits, and thus a 16-bit operand bitwidth is used which results in a 30% energy/op reduction.

This is due in part to the fact that we use a Wallace-tree structure, rather than an array multiplier [9] (for which energy savings are more dramatic as bitwidth decreases). Also, we require extra random logic for control of the multiple passes through the digit serial structure for larger width operands. This additional energy is the penalty we pay for the flexibility of doing variable bitwidth multiplication.

For 8-bit multiplication, the digit-serial multiplier consumes 78% less energy than the 24 × 24 Wallace tree multiplier (in the case of Sphinx and ALVINN). When 9–16 bits of the mantissa are required (in the case of Fast DCT, PCASYS and Bench22), the digit-serial multiplier still consumes 32% less energy than the 24 × 24 Wallace tree multiplier. The digit-serial multiplier does consume 19% more energy when performing 24-bit multiplication due to the overhead circuitry.

Fig. 11 shows a simple estimate of the potential energy savings in the FP multiplier for our five programs if we use the digit-serial multiplier as the mantissa multiplier. The data is based on the estimate that a 24-bit Wallace tree multiplier consumes 80% of the total energy in a single-precision FP multiplier. (We have taken the data from Table I and simply rounded the multiplier core's contribution to 80%.) A reduction of 66% in multiplication energy/op is attainable for Sphinx and ALVINN with the

```
Set Bitwidth = B_i (where B_i is the lower bound on data bitwidth)
    do {
        Run the program with B_i
        Error = Run Result - Expected Result
        Increase B_i
    } while (Error > Error Threshold)
```

Fig. 12. Calculating necessary FP bitwidth for an arbitrary program.

use of a digit-serial multiplier. Both Sphinx and ALVINN need only 5 bits of mantissa to be 90% accurate, and thus an 8-bit operand bitwidth is used for the digit-serial multiplier. PCASYS requires 11 bits of mantissa while Bench22 requires 9 bits, and thus a 16-bit operand bitwidth is used; this results in a 30% energy/op reduction.

These comparison results, though still clearly approximate, do suggest that significant energy savings are achievable by using lower precision arithmetic. Of course, our digit-serial multiplier was designed using an ASIC approach and was not as heavily optimized physically as the Wallace tree multiplier, which was compiled/optimized for the process, and digit-serial arithmetic is only one architectural alternative here. Given the decreasing cost of silicon area, and the small size of a reduced precision FP unit, one might imagine simply including both full and reduced precision FP units, and using appropriate sleep-mode circuit techniques to shut down the unused unit. Assuming a richer, less homogenous arithmetic substrate for power-optimized computation naturally suggests opportunities for compiler support, i.e., exposing a more diverse set of precision/speed/power tradeoffs and allowing software developers to choose appropriately. We briefly treat this idea in Section VI.

E. Software-Assisted Bitwidth Reduction

Since different applications require different algorithms to determine program accuracy, software support is desirable in order to harness the benefits of this bitwidth reduction technique. This can be done by the programmer—but can be assisted greatly by the compiler.

There are two approaches that the programmer can take. For applications where the bitwidth requirement is known in advance, the programmer can annotate the source code explicitly or modify the declaration of the data types. For instance, since Sphinx requires only 5 bits of mantissa and 6 bits of exponents to maintain 90% of recognition accuracy, the programmer can simply choose an appropriate, available lower precision FP format. Given this information, the compiler can then optimize the object code for power. In applications where there is no prior knowledge of the required bitwidth, the programmer can insert an error analysis procedure [15] which adjusts the data bitwidth during data profiling (analogous to self-adjusting bit-precision techniques applied in low-power digital filters [16]). This is illustrated in Fig. 12. The final B_i will be the minimum bitwidth that can satisfy the accuracy requirement for that specific data set.

In situations where the error analysis routine is too complicated or where the program users have no comprehensive knowledge of its algorithms, a compiler can be used to annotate the source code and find at least the approximate bitwidth requirement automatically, assuming a comprehensive data set is available.

To demonstrate the possibility of compiler assisted bitwidth reduction, we added a compiler pass to the SUIF research complier [30] to help us find the bitwidth requirement of FP programs. SUIF was developed at Stanford with the objective of providing a research vehicle that allows compiler researchers to implement their ideas without having to write a new compiler from scratch. After every step in the SUIF compilation process, a standard format is written out. This allows us to insert any optimization in a separate compiler pass.

We modified SUIF by inserting a pass that annotates the source code of FP programs. For every FP operation, the input operands were masked with the desired bitwidth. This is analogous to what we have done manually in Section IV-B. Of course, masking is an unsubtle approach to precision reduction: by zeroing low-order mantissa bits, we prevent the rounding mechanisms from correctly propagating bits up into the higher order bits we retain. Nevertheless, as an experiment this does provide some flavor of the sorts of optimizations we believe should be possible given a more diverse arithmetic substrate. Starting from the original bitwidth specified in the IEEE standard, the program was executed using a representative input set. The output was then compared with the expected result. If the results match, the bitwidth was reduced and the program was again executed using the same input set. This iterative process continues until the minimum bitwidth that can satisfy the output requirement is found. All of the above steps can be automated using a simple script. As a simple but concrete experiment here, we used the SUIF-generated code for the Fast DCT program and found the same bitwidth requirement as we did with the methodology described in Section IV-C.

Modern compilers are capable of performing sophisticated techniques of code optimization to improve program performance. Global compile-time optimization of the necessary precision/range for arithmetic operands may be feasible if we integrate variable-precision units, or a collection of mixed-precision arithmetic units in our hardware.

VI. Related Research

Substantial work on low-power datapath design techniques has been done at the circuit level. In particular, there have been many studies on low-power multiplier designs due to the prevalence of multiplication in DSP-type applications [17]–[19]. However, there have been few published studies targeting power reduction techniques for FP multipliers or the entire FP unit at the system or architectural level.

The basic principle behind our proposed technique is to avoid unnecessary or useless work in the hardware. This principle has been applied in other work to either improve performance [20] or reduce power consumption [21]. For example, [20] proposes the use of "memoing" to accelerate multimedia processing. By saving the input and output of previous calculations and using the output if the inputs are encountered again, the authors report an average computational speed up of more than 20%. In [21],

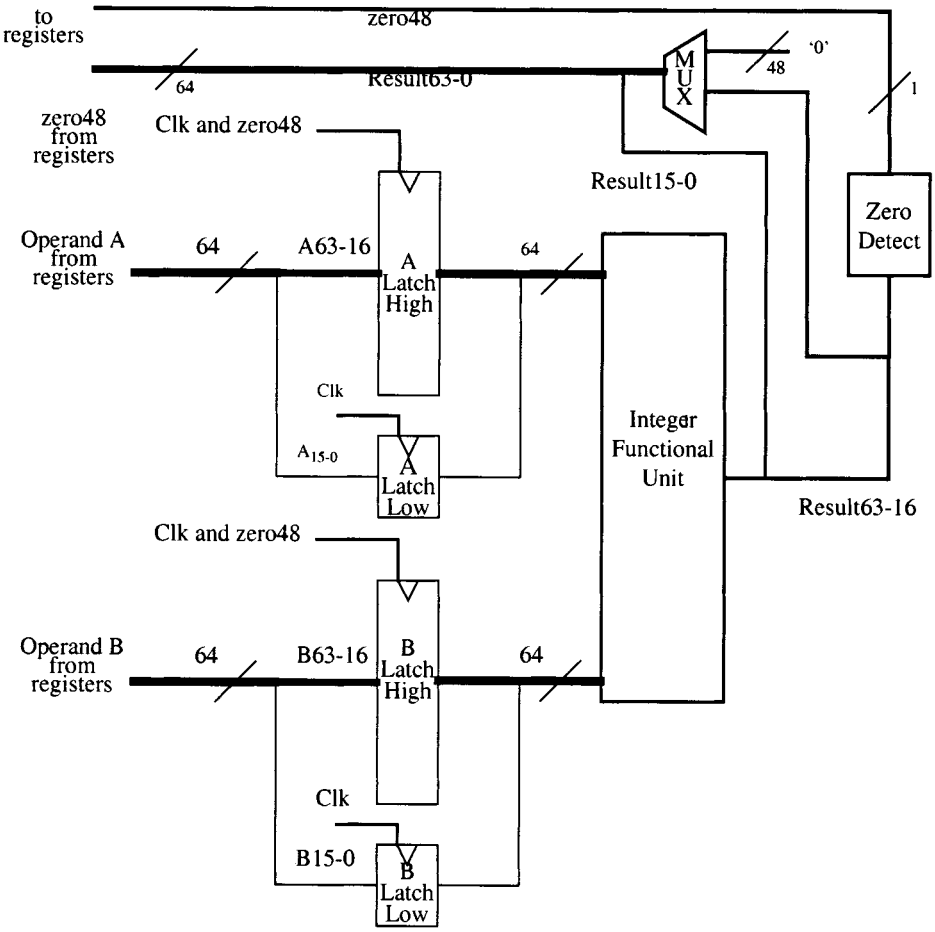


Fig. 13. Clock gating architecture proposed in [24].

the idea of reusing computation results stored in a cache-like structure was proposed. But the design parameters of the structure ("execution cache" coined by the authors) are optimized for power minimization instead of just improving the execution speed. They reported energy savings up to 60% in the integer unit.

Furthermore, the idea of reducing bitwidth to save power has also been employed in other areas of low-power research. [22] showed that an average of more than 4 bits of pixel resolution can be dropped during motion estimation to obtain a power reduction of 70%. To reduce energy consumption at the I/O pins, [23] proposes sending only those address bits that have changed. For integer applications that do not need 32 bits of precision, the Intel MMX instructions allow arithmetic operations on multiple narrow subfields of a data word simultaneously.

A closely related approach was proposed by Brooks and Martonosi [24]. In that paper, the authors found that across the SPECint95 benchmarks, over half of the integer operations require 16 bits or less. They proposed to reduce processor power consumption by using aggressive clock gating to turn off portions of integer arithmetic units when the full width of a functional unit is not required (see Fig. 13). By evaluating the percentage of narrow bitwidth operations, they reported that the optimization results in an over 50% reduction in the integer unit's power consumption.

However, [24] does not include data on FP programs. Since our work focuses primarily on FP operations, we have performed a similar analysis on SPECfp95 applications. In this case, we define an operation as a *narrow bitwidth operation* when both of its operands have mantissa values that can be packed in 16 bits or less (the lower bits of the mantissa are all "zeros"). One obvious difference here is that we are looking at the most significant 16 bits of the mantissa instead of the least significant bits as in integer operations.

Also, our methodology for measuring the power savings is different. Instead of looking at the percentage of narrow bitwidth operations, we estimate the power saving by evaluating the percentage of narrow bitwidth operations *preceded* by a *non*narrow bitwidth operation (or wide bitwidth operations). It is well known that power dissipation in CMOS circuits is proportional to the switching activity of the circuits. For instance, if the narrow bitwidth operations tend to cluster together, then for consecutive operations using the same functional unit, the upper bits of the inputs remain at "zeros" or "ones" all the time which should then generate fewer transitions in the upper portion of the datapath circuits.

To estimate the power reduction by clock gating the input operands, we need to count the number of transitions from wide bitwidth operations to narrow bitwidth operations (Fig. 14). We used the instrumentation program ATOM to instrument eight

Instruction #	*Operand A*	*Operand B*
1	0x7 f f f f f f f	0x7 f f f f f f f
2	0x0000006e	0x0000006e
3	0x0000007f	0x0000007f
4	0x0000003d	0x0000003d
5	0x5 f f f f f f f	0x5 f f f f f f f

A

B

Fig. 14. Estimation of power reduction via clock gating. The figure shows a set of consecutive integer add operations for a 32-bit machine. Assume for illustration that narrow bitwidth operations are defined as instructions where both operands use less than 8 bits. Transition A (instr 1 $\rightarrow$ instr 2) is a transition from wide bitwidth operation to narrow bitwidth operation. Transition B (instr 2 $\rightarrow$ instr 3) is a transition from narrow bitwidth to narrow bitwidth operation. With clock gating, we only save power in transition A, but not transition B.

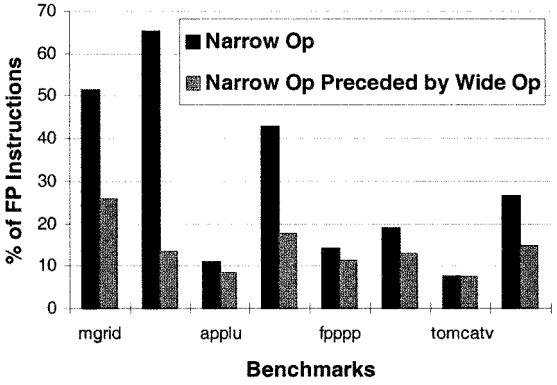


Fig. 15. Percentage of narrow bitwidth operations in SPECfp95.

SPECfp95 programs (Table III) and count the number of transitions from wide bitwidth operations to narrow bitwidth operations.

The results of the experiments are shown in Fig. 15.The percentage of narrow bitwidth instructions that are preceded by wide bitwidth instructions varies from 8 to 20%. Taking into account the power consumed in the overhead logic (latches and zero detection circuits), actual power reduction is approximately 5–10%.

Furthermore, we examined the occurrence of narrow width FP operands among the five programs in Table IV to see how a clock gating scheme might save power for these programs. Fig. 16 shows the result (notice that the y-axis is in log scale). For Sphinx and Alvinn, the percentage of narrow width operations is less than 1% of the total FP operations. These results suggest that arithmetic unit clock gating schemes, which can leverage the relatively high frequency of consecutive narrow bitwidth operands in integer datapaths, are less viable in the sort of human-sensory FP signal-processing workloads we think will be important in portable/mobile electronics. In general, these sorts of "dynamic" suppression techniques are very attractive because of their transparency: the hardware appears to the program to be doing a typical arithmetic operation at full precision, yet at circuit level, useless transitions are suppressed. But based on our observations of FP workloads, we believe there is merit in making *explicit* the fact that some operands *always* need more/fewer bits than others. Our initial experiments suggest that the availability of reduced-precision arithmetic could

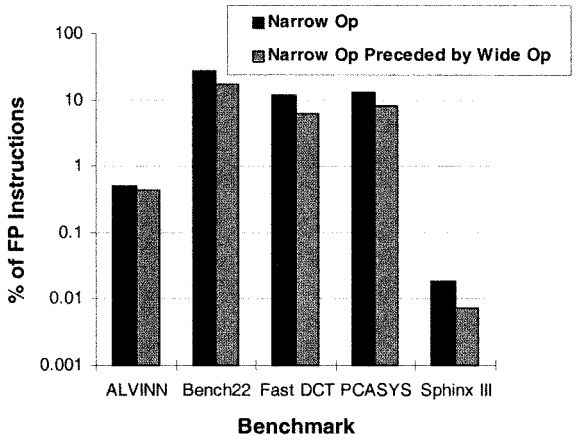


Fig. 16. Percentage of narrow bitwidth operations in our workload.

allow compile-time optimizations that simply omit the unnecessary bits and computations on them, and by exposing these variable precision possibilities to the compiler, a wide range of further system-level optimizations may be feasible.

VII. Conclusion

Many mobile/portable electronics applications will ultimately need to process human-sensory data such as speech or video imagery. This data is often acquired at relatively low bit resolutions, e.g., 4–10 bits of precision. Nevertheless, many signal processing algorithms are more easily and efficiently coded assuming an FP representation, which liberates developers from concerns over either the precision or range of the intermediate results. Common design practice is then to translate these FP algorithms into fixed precision arithmetic. The intuition underlying this study is that many of these applications—especially those that process low-resolution data and render as a result a decision, such as speech and image recognition—may be better served with a custom, reduced FP format.

By annotating source code to replace FP operators with calls to emulation routines, we were able to systematically vary the FP format across a set of speech and image processing tasks. Analysis of these FP programs shows that they suffer no loss of accuracy, even with a significant reduction in bitwidth, thus confirming our intuition. Most FP programs in our benchmark suite maintain the same output even when the mantissa bitwidth is reduced by half. Exponents can likewise be reduced, but not as dramatically. In hindsight, this result is reasonable: these codes require more dynamic range than can be achieved with fixed-width integers, but not dramatically more precision in intermediate results.

This FP bitwidth reduction can deliver a significant energy savings through the use of a variable bitwidth FP unit. By creating and comparing ASIC-style fixed-width and variable-width (reduced-width) FP multipliers, we estimated that a reduction of up to 66% in multiplier energy/operation can be achieved in the FP unit without sacrificing any program accuracy. As a result of these experiments, we suggest that the fact that some operands need different bitwidths should be exposed to compilers. This

would allow us to attempt compile-time precision/range optimizations which could then be realized using either variable-precision or a collection of mixed-precision arithmetic units in hardware. This approach has two attractive advantages: it opens another perspective on power optimization to the compiler for large-scale global optimization, and the availability of mixed-precision arithmetic always allows us to recover from extraordinary circumstances by simply repeating a computation with extra precision.

To date, there has been relatively little work on FP representation and computation from the perspective of power optimization. This is in contrast to a large and growing body of circuit, logic, and system-level optimizations for integer arithmetic. We argue that FP computation is inevitable as low-power systems evolve and that studies of the tradeoffs available among operand precision, dynamic range, and rounding modes can play an important role here.

ACKNOWLEDGMENT

The authors would like to thank H. Schmit of Carnegie-Mellon University for his role in the creation of the ASIC design flow used in the work, P. Meier of Carnegie-Mellon University for assistance with power and energy estimation techniques, and B. Ackland and C. Nicol of Bell Laboratories, Lucent Technologies, for enlightening discussions about arithmetic issues in DSP applications.

REFERENCES

[1] P. C. H. Meier, R. A. Rutenber, and L. R. Carley, "Exploring multiplier architecture and layout for low power," in *Custom Integrated Circuits Conf.*, 1996.

[2] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std. 754-1985, Aug. 1985.

[3] D. Dobberpuhl, "The design of a high performance low power microprocessor," in *Int. Symp. Low Power Electronics and Design*, Aug. 1996, pp. 11–16.

[4] "MCore shrinks code, power budgets," Microprocessor Rep., vol. 11, Oct. 27, 1997.

[5] M. R. Santoro, G. Bewick, and M. A. Horowitz, "Rounding algorithms for IEEE multipliers," in *10th Int. Symp. Computer Arithmetic*, 1991.

[6] T. Huff, M. Upton, P. Sherhard, P. Barker, R. McVay, T. Stanley, R. B. Brown, R. Lomax, T. Mudge, and K. Sakallah, "A high performance GaAs microprocessor," in *Proc. IEEE Laser and Optics Society Sarnoff Symp.*, Princeton, NJ, Mar. 1993.

[7] R. V. K. Pillai, D. Al-Khalili, and A. J. Al-Khalili, "Energy delay measures of barrel switch architectures for pre-alignment of floating point operands for addition," in *Int. Symp. Low Power Electronics and Design*, Aug. 1997, pp. 235–238.

[8] K. P. Acken, M. J. Irwin, and R. M. Owens, "Power comparisons for barrel shifters," in *Int. Symp. Low Power Electronics and Design*, Aug. 1996.

[9] T. K. Callaway and E. E. Swartzlander, "Power-delay characteristics of CMOS multipliers," in *IEEE 13th Symp. Computer Arithmetic*, 1997.

[10] M. Hwang, R. Rosenfeld, E. Theyer, R. Mosur, L. Chase, R. Weide, X. Huang, and F. Alleva, "Improving speech recognition performance via phone-dependent VQ codebooks and adaptive language models in SPHINX-II," in *Int. Conf. Acoustics, Speech and Signal Processing*, 1994.

[11] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 437–445, Dec. 1994.

[12] R. I. Hartley and K. K. Parhi, *Digit-Serial Computation*. Norwell, MA: Kluwer Academic, 1995.

[13] R. Burch, F. Najm, P. Yang, and T. Trick, "McPOWER: A Monte Carlo approach to power estimation," in *IEEE/ACM Int. Conf. CAD*, 1992.

[14] Y. N. Chang, Satyanarayana, H. Janardhan, and K. K. Parhi, "Design and implementation of low-power digit-serial multipliers," in *IEEE Int. Conf. Computer Design*, 1997, pp. 186–195.

[15] C. Alippi and L. Briozzo, "Accuracy vs. precision in digital VLSI architectures for signal processing," *IEEE Trans. Computers*, vol. 47, Apr. 1998.

[16] P. Larsson and C. J. Nicol, "Self-adjusting bit-precision for low-power digital filters," in *Symp. VLSI Circuits Dig. Tech. Papers*, June 1997.

[17] I. S. Abu-Khater, A. Bellaouar, and M. I. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers," *IEEE J. Solid-State Circuits*, vol. 31, Oct. 1996.

[18] B. S. Cherkauer and E. G. Friedman, "A hybrid radix-4/radix-8 low power signed multiplier architecture," *IEEE Trans. Circuits Syst. II*, vol. 44, Aug. 1997.

[19] R. K. Krishnamurthy, H. Schmit, and L. R. Carley, "A low-power 16-bit multiplier-accumulator using series-regulated mixed swing techniques," in *IEEE Custom Integrated Circuits Conf.*, May 1998.

[20] D. Citron, D. Feitelson, and L. Rudolph, "Accelerating multi-media processing by implementing memoing in multiplication and division units," in *8th Int. Conf. Architectural Support for Programming Languages and Operation Systems*, Oct. 1998.

[21] M. Azam, P. Franzon, and W. Liu, "Low power data processing by elimination of redundant computations," in *Int. Symp. Low Power Electronics and Design*, Aug. 1997, pp. 259–264.

[22] Z. L. He, K. K. Chan, C. Y. Tsui, and M. L. Liou, "Low power motion estimation design using adaptive pixel truncation," in *Int. Symp. Low Power Electronics and Design*, Aug. 1997, pp. 167–172.

[23] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," in *Int. Symp. Low Power Electronics and Design*, Aug. 1997, pp. 202–207.

[24] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *5th Int. Symp. High Performance Computer Architecture*, Jan. 1999.

[25] S. F. Anderson *et al.*, "The IBM system 390 model 91: Floating point execution unit," *IBM J. Res. Dev.*, vol. 11, pp. 34–53, 1967.

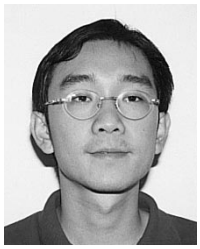
[26] P. Ackland and P. D'Arcy, "A new generation of DSP architectures," in *Proc. IEEE CICC*, May 1999.

[27] SPEC Benchmark Suite Release 1.0, SPEC, Santa Clara, CA, Oct. 2, 1989.

[28] SPECfp95 Benchmark, Available: http://www.spec.org.

[29] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1994.

[30] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion, and M. S. Lam, "Maximizing multiprocessor performance with the SUIF compiler," *IEEE Computer*, Dec. 1996.



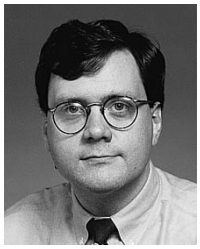
Jonathan Ying Fai Tong (S'97–M'99) received the B.S.E.E. degree from the University of Texas at Austin in 1996 and the M.S. degree from Carnegie-Mellon University, Pittsburgh, PA, in 1998.

Since 1999, he has been a Logic and Circuit Designer at Motorola's MCORE Technology Center, Austin, TX. His research interests include low-power electronics, VLSI CAD, and computer architecture.

Mr. Tong is a member of Eta Kappa Nu and Tau Beta Pi.

David Nagle (S'85–M'95) received the B.S., M.S., and Ph.D. degrees from the University of Michigan, Ann Arbor.

He is a member of the faculty at Carnegie-Mellon University and Director of Carnegie-Mellon University's Parallel Data Lab. He coleads CMU's Network-Attached Secure Disk (NASD) project and the Active Storage Networks project. His research interests include OS/architecture interactions, storage systems, networking, and hardware-based security.



Rob A. Rutenbar (S'77–M'84–SM'90–F'98) received the Ph.D. degree from the University of Michigan, Ann Arbor, in 1984.

He subsequently joined the faculty of Carnegie-Mellon University, Pittsburgh, PA. He is currently Professor of Electrical and Computer Engineering and (by courtesy) of Computer Science. From 1993 to 1998, he was Director of the CMU Center for Electronic Design Automation. His research interests focus on circuit and layout synthesis algorithms for mixed-signal ASIC's, high-speed digital systems, and FPGA's. He served on the Editorial Board of the IEEE SPECTRUM.

Dr. Rutenbar received a Presidential Young Investigator Award from the National Science Foundation in 1987. He has won Best/Distinguished paper awards from the Design Automation Conference in 1987 and the International Conference on CAD in 1991. He has been on the program committees for the IEEE International Conference on CAD, the ACM/IEEE Design Automation Conference, the ACM International Symposium on FPGA's, and the ACM International Symposium on Physical Design. He was General Chair of the 1996 ICCAD. He chaired the Analog Technical Advisory Board for Cadence Design Systems from 1992 through 1996. He is a member of the ACM and Eta Kappa Nu.