A Fully Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor

Hwa-Joon Oh, *Member, IEEE*, Silvia M. Mueller, Christian Jacobi, Kevin D. Tran, Scott R. Cottier, *Member, IEEE*, Brad W. Michael, Hiroo Nishikawa, Yonetaro Totsuka, Tatsuya Namatame, Naoka Yano, Takashi Machida, and Sang H. Dhong, *Fellow, IEEE*

***Abstract*—The floating-point unit (FPU) in the synergistic processor element (SPE) of a CELL processor is a fully pipelined 4-way single-instruction multiple-data (SIMD) unit designed to accelerate media and data streaming with 128-bit operands. It supports 32-bit single-precision floating-point and 16-bit integer operands with two different latencies, six-cycle and seven-cycle, with 11 FO4 delay per stage. The FPU optimizes the performance of critical single-precision multiply–add operations. Since exact rounding, exceptions, and de-norm number handling are not important to multimedia applications, IEEE correctness on the single-precision floating-point numbers is sacrificed for performance and simple design. It employs fine-grained clock gating for power saving. The design has 768K transistors in 1.3 mm^2, fabricated SOI in 90-nm technology. Correct operations have been observed up to 5.6 GHz with 1.4 V and 56 °C, delivering 44.8 GFlops. Architecture, logic, circuits, and integration are codesigned to meet the performance, power, and area goals.**

***Index Terms*—Floating-point arithmetic, integrated circuit design, microprocessors, very large-scale integration.**

I. Introduction

THE synergistic processing element (SPE) [1] of a CELL processor [2] is the first implementation [3] of a new architecture designed to accelerate multimedia applications, such as three-dimensional (3-D) graphics, media streaming, and signal processing. Real-time multimedia applications demand single-precision performance significantly exceeding that of conventional processors. An SPE contains a set of 128 registers that are 128 bits wide. These registers are used by both floating-point unit for single- and double- precision arithmetic and fixed-point unit for 32-bit integer arithmetic and logical operations [1].

The single-precision floating-point unit (FPU) of the SPE is a 4-way single-instruction multiple-data (SIMD) design. Vector computing (or SIMD) has been used in supercomputers and modern microprocessors have media extensions such as SSE, MMX, and VMX/AltiVec for an SIMD design. Most instructions at the FPU in the SPE process 128-bit operands, divided into four 32-bit word slices. Each of the four slices supports 32-bit single-precision and 16-bit integer multiply–add instructions and convert instructions between single-precision floating point and integer. The single-precision floating-point multiply–add instruction consumes three register operands and produces a register result. Operands are fetched from the register file (RF) to the operand latches of the FPU. Either floating-point results are bypassed directly from the result multiplexer of the FPU to the input operand latches of the FPU to reduce the result latency or results of the FPU are sent to the forward unit (FW) from where they are distributed to other units (i.e., fixed-point unit, register file, or local-store unit). Fig. 1 shows a simplified FPU pipeline structure.

Manuscript received September 5, 2005; revised December 19, 2005.

H.-J. Oh, K. D. Tran, S. R. Cottier, B. W. Michael, and S. H. Dhong are with the IBM System and Technology Group, Austin, TX 78758 USA.

S. M. Mueller and C. Jacobi are with IBM Entwicklung GmbH, Boeblingen 71032, Germany.

H. Nishikawa is with IBM Global Services, Yasu 520-2392, Japan.

Y. Totsuka is with the Digital Imaging Group, Sony Corporation, Tokyo 108-6201, Japan.

T. Namatame, N. Yano, and T. Machida are with the Semiconductor Company, Toshiba Corporation, Kawasaki 212-8520, Japan.

Digital Object Identifier 10.1109/JSSC.2006.870924

II. Design Challenges

A. 11 FO4 Design

Recent studies [4] show that the pipeline depth for a performance optimized design is in the range of 6–8 fanout-of-4 (FO4) inverter delays per stage. Whereas for a power- and performance-optimized design, study [5] suggests an optimal delay of about 18 FO4 per stage (consisting of a logic delay of 15 FO4 and 3 FO4 latch insertion delay). The first implementation of CELL processor uses a design point of 11 FO4 per stage, since performance and power are important. The latency of the FPU is six cycles for single-precision instructions in order to meet the performance requirements of the target workloads. Since result forwarding takes 6 FO4, the whole delay budget including latch insertion delays is 60 FO4 for the single-precision logic. A state-of-the-art FPU has a latency of around 100 FO4 [6] since single-precision data are usually handled in the double-precision unit. To design a dedicated six-cycle single-precision floating-point unit with 11 FO4 cycle time required optimizations at all design levels: architecture, logic, circuits, layout, and placement.

B. Latch Insertion Delays

CMOS static gates are used to implement most of the logic; dynamic circuits are used in certain timing critical areas. A latch insertion delay of 2 FO4 to 3 FO4 occupies a significant portion of the 11 FO4 cycle. In order to minimize this delay, a special latch selection was provided (Table I). There are three types of latches: type-C, type-D, and type-E. All latches have multiple

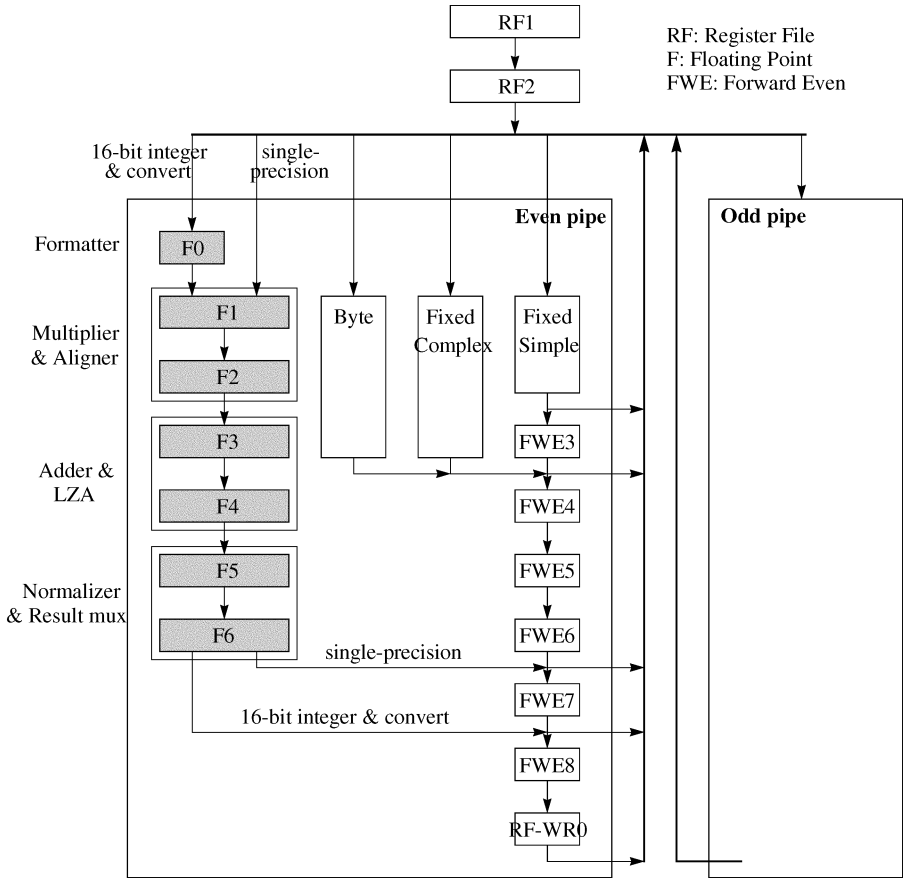


Fig. 1. Simplified pipeline structure of the floating-point unit at the SPE.

TABLE I
CUSTOM LATCH TYPES PROVIDED IN THE IMPLEMENTATION OF THE CELL PROCESSOR

Type	Characteristics	Scan or non-scan	Dynamic or static	Nominal latch insertion delay	Height of bit cell
Type-C	It combines up to 9:1 multiplexer. It is called High Performance Latch (HPL) [2]	scan only	dynamic	data: 3FO4 selection: 4.5FO4	high
Type-D	General purpose Flip/Flop	scan and non-scan	static	3FO4	middle
Type-E	Low setup time pulsed latch. Input is either an inverter or a NAND2.	non-scan only	static	2.3FO4	low

levels of driver size and pre-defined levels of local clock buffers (LCBs).

Type-C (high-performance latch [2]) latch includes a multiplexer up to eight inputs (one input is used for scan-in). It accepts static inputs, drives static outputs, and requires only 3 FO4 data delays for both multiplexing and latching function. Due to its internal dynamic circuits, the selection signal of the multiplexer requires about 4 FO4 set-up time. Sometimes, that selection path becomes critical path in the unit timing.

Type-D is the basic transmission-gate flip-flop. It includes a normal and a scan signal as inputs. It is the standard latch used all over the chip.

Type-E is a pulsed latch which can reduce the insertion delay by replacing the inverter with NAND2 logic with latch function. Its transparent window allows it to borrow up to 1 FO4 from the next or previous cycle; that helps balancing the delay of multi-cycle paths.

Both type-C and type-E of latches are heavily used in the FPU to improve timing.

C. Width Restrictions and Macro Placements

The physical width of each FPU slice is 46 bits wide and has to be aligned to the data flow stack of the SPE. From the register file through fixed-point units down to the FPU, the 46-bit data flow stack consists of a 32-bit data area and a 14-bit clock-bay area. The clock bay area occupies in the center area over two slices: slice0 data (32-bit), slice0 clock-bay (14-bit), slice1 clock-bay (14-bit), slice1 data (32-bit). The area of slice2 and slice3 follows the same way as the placement of slice0 and

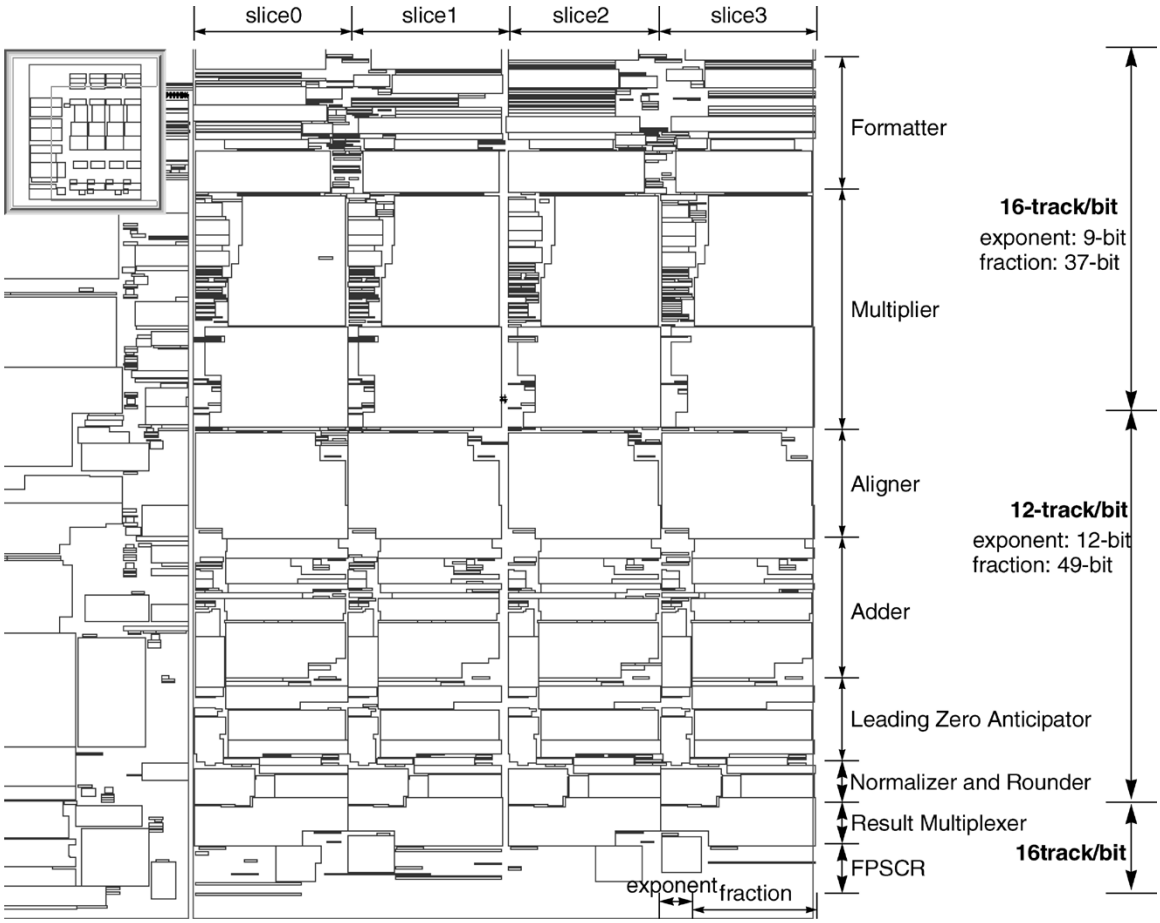


Fig. 2. Floor plan and placement of FPU macros, buffers, and stand-alone latches. Each slice has vertical dataflow of the exponent and the fraction area. There are two bit-images such as 16 track/bit and 12 track/bit.

slice1. Standard bit image is 16 tracks per bit, where a track is a Metal-2 or a Metal-4 pitch in IBM SOI technology.

The 46-bit data flow is very narrow for a FPU since intermediate results require 11-bit exponents and up to 96-bit fractions. The FPU data flow is split into an exponent (9 bit wide) and a fraction part (37 bit wide). Most fraction macros are folded to reduce their width. The FPU layouts use two different bit images, 16 track/bit and 12 track/bit. 16-track/bit regions assign 9 bits for exponent and 37 bits for fraction, whereas 12-track/bit regions assign 12 and 49 bits, respectively. The placement of macros and buffers is shown in Fig. 2.

In the CELL microprocessor, the main high-frequency clock grid covers over the FPU area [2]. Each macro has its own Metal-3 clock pins. Those clock pins are connected to global clock grid at the chip level integration step. The LCB of each macro is generally located at the sides of the latch bit cells, far left or far right. Sometimes, there is no space next to latch bit cells. In that case, the LCB is located above or below the row of latch bit cells.

In conventional designs, a multiplier and an aligner are placed side by side since they process the operands simultaneously. Due to the width restriction, the aligner was placed between the multiplier and the adder. The operands for the aligner have to fly over the multiplier, and the multiplier results have to fly over the aligner in order to get to the adder. Thus, the narrow design increases the wire length and the wire congestion. To ensure that the unit can be wired and to limit the timing impact, the designers had to have careful wire planning from the beginning of the project. Wire and re-powering delays were limited to 30% of a cycle: strict wiring assignments were applied. Macro level wiring was restricted to Metal-3 and below. Some macros have mandatory empty wiring tracks at Metal-2 for global wiring.

D. Optimizing for the Target Applications

The FPU is optimized for single-precision multiply–add type instructions [7]. In order to achieve the performance goal of the target applications and to meet their special architecture needs, the handling of single precision floating point numbers at the FPU deviates from the IEEE standard in some points [8], [9]: It only supports truncation rounding and no-trapping exception handling. De-normal numbers are flushed to zero. Numbers with the maximum exponent $\mathrm{e} = 255$ are treated as regular numbers rather than representing NaN and infinity. In media applications, NaN and infinity usually have no real meaning but the extended data set is very useful. Operations which produce an overflow saturate to the maximum representable number instead of infinity. These features are effective and useful in handling multimedia applications such as streaming and 3-D graphics. Note that a special exception flag is set at the floating-point status and control register (FPSCR) whenever the result could deviate

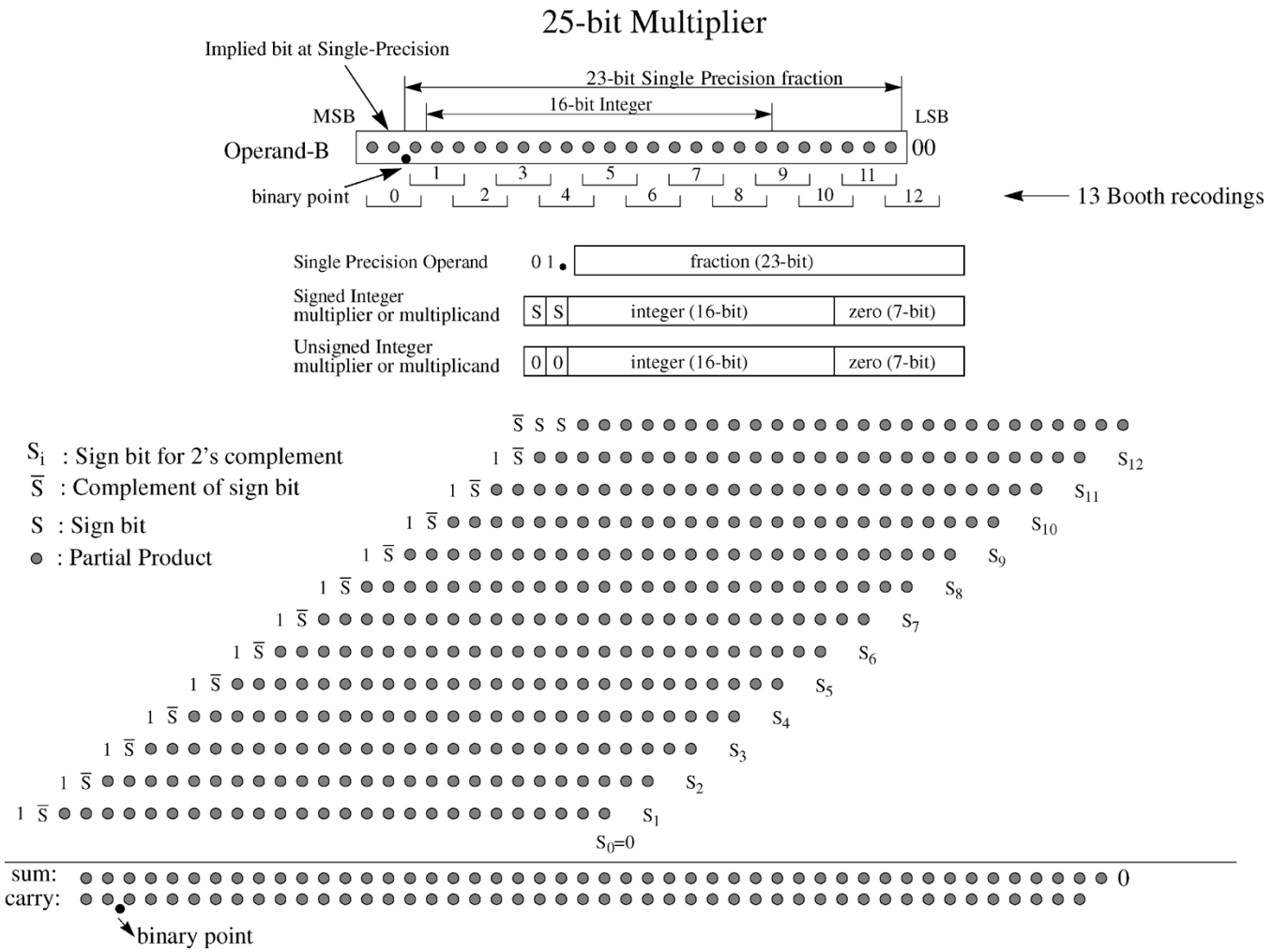


Fig. 3. Partial product of 25-bit multiplier with hot-1 encoding.

from the IEEE result. This enables software support for missing IEEE single-precision features.

III. Main Building Blocks

A. Formatter

The main data flow is designed to execute single precision floating instructions in six cycles. An extra format stage is added to pre-process the operands of integer and convert instructions. With this extra cycle, 16-bit integer multiply–add and convert operations are executed in the main single-precision pipeline as special floating point multiply add operations. Thus, the FPU has two different latencies, six-cycle for single precision multiply–add type instructions and seven-cycle for instructions which require operand formatting. These seven-cycle FPU instructions comprise integer multiplies, FPSCR write, integer converts, and floating-point interpolation.

The mantissa of a floating-point number is an unsigned number, whereas the integers are either two's complement or unsigned numbers. In order to perform the multiplication of these data types in the same multiplier, the operands need some sign extensions: 1) The floating-point mantissa gets extended with a zero sign. 2) The signed integer multiplier operands get sign extended by two bits. 3) The unsigned integer multiplier operands get extended by two zero sign bits. The embedding method is shown in the middle of Fig. 3. After the sign extension, the floating-point operands have a 25-bit mantissa. The integer multiplier and multiplicand are 18 bits; they are mapped into the most significant bits (MSBs) of the mantissa.

B. Multiplier

The multiplier uses radix-4 Booth recording, which cuts the number of partial products in half and supports two's complement number multiplications. The recording method scans 3-bit with 1-bit overlap in the B-Operand (Fig. 3). Those 3-bit Booth digits together with the A-operand define the value of the corresponding partial products which are 0, A, $-$A, 2A, or $-$2A (Table II). Hot-1 encoding is used to deal with the sign bit extension of the negative partial products limiting their width to 28 and 29 bits.

Fig. 4 shows the details of the multiplier design. The partial products get generated by a combination of a two's complement recoding (negation in the case of $-$A and $-$2A) and shifting to obtain 2A and $-$2A. Since the generation of "double" and "single" signals is slower than "sign" signal, the negation selection is applied to the operand-B before the shifting stage.

TABLE II
PARTIAL PRODUCT GENERATION WITH HOT-1 ENCODING (MULTIPLICAND A IS 25 BITS WIDE)

$B_{i+1}B_iB_{i-1}$	Recorded value	Partial Product: PP (26bits)	Sign Bit S_i (1bit)	Partial Product at LSB $\overline{S}SS$ + PP (29bits)	Partial Product at other location $1\overline{S}$ + PP (28bits)
0 0 0	0	**0000...000**	0	1000**000...000**	110**000...000**
0 0 1	+A	A_0**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**	0	$\overline{A}_0A_0A_0A_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**	1 $\overline{A}_0A_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**
0 1 0	+A	A_0**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**	0	$\overline{A}_0A_0A_0A_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**	1 $\overline{A}_0A_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**
0 1 1	+2A	**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**0	0	$\overline{A}_0A_0A_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**0	1 $\overline{A}_0$**$A_0A_1A_2A_3$ ... $A_{22}A_{23}A_{24}$**0
1 0 0	-2A	**$\overline{A_0}\overline{A_1}\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**1	1	$A_0\overline{A}_0\overline{A}_0$**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**1	1 A_0**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**1
1 0 1	-1A	**$\overline{A}_0\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**	1	$A_0\overline{A}_0\overline{A}_0\overline{A}_0$**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**	1 $A_0\overline{A}_0$**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**
1 1 0	-1A	**$\overline{A}_0\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**	1	$A_0\overline{A}_0\overline{A}_0\overline{A}_0$**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**	1 $A_0\overline{A}_0$**$\overline{A_0}A_1\overline{A_2}\overline{A_3}$ ... $\overline{A_{22}}\overline{A_{23}}\overline{A_{24}}$**
1 1 1	0	**0000...000**	0	1000**000...000**	110**000...000**

The floating point fraction gets extended by a leading zero sign bit. Thus, the most significant Booth digit starts with a zero (0XX, X can be either 0 or 1) and the corresponding partial product is not negative. For the 16-bit integer multiply operations, the operands get extended by two additional sign bits or by two leading zeros. The most significant Booth digit then is either 00X or 111 resulting in a positive or zero partial product. Thus, the special embedding of the floating-point fraction and of the integer operands guarantees that the most significant partial product is not negative, i.e., that its sign bit is $S_0 = 0$. This reduces the latency of the multiplier by one 3:2 counter stage since it reduces the inputs of the reduction tree from 14 to 13.

The 13 partial products are reduced to two output vectors using five stages of reduction with static full adders (FAs). The static FA circuit consists of transmission-gate XOR and AOI, as shown at the bottom in Fig. 4. The static implementation of the full adder has advantages over dynamic circuits; it reduces the wiring need and allows more timing optimizations. Dynamic adders generally need true and complement signals; static circuits can cut the wire density in half. When the source is located near the input x of the full adder, it can be connected to the input of the transmission-gate XOR without additional protection. If the distance from the source to the input x of the full adder is long, an inverter stage is added to the signal x_b to generate x_t (x_true) signal, which is connected to the input of the transmission-gate XOR. This adds extra delay. The structure of our reduction tree exploits the different path delays in a static full adder [10]. Since the full adder has the slowest path from x to sum and the fastest path from z to carry, the carry output of one stage is connected to the x input of the next stage. By doing so, we are able to reduce the overall path delay of the multiplier.

The multiplier is a two-cycle design with a total block delay of 22 FO4. The staging latches between the first and second cycle are special pulsed latches (type-E). The transparent window of these latches allows utilizing slack borrowing efficiently at the different path delays in the reduction tree between the two cycles. Buffers are added to the faster paths to ensure early mode (i.e., high supply voltage and fast corner technology) timing; this is important since pulsed latches have a transparent window for data passing.

With 37-bit-width restriction, the partial product cannot be aligned to vertical data flow as shown in Fig. 3; the tree needs to be compressed in width. Partial product location is shifted in horizontal location and horizontal wires are drawn to connect the signals. The two output vectors of the multiplier, sum and carry, are 49 bits wide. They are latched using the 12-track/bit image rather than 16-track/bit image to avoid the folded latch placements.

C. Aligner

The aligner block shifts the addend based on the exponent difference in order to align it with the product. The exponent difference can be as large as 512. Such wide shifters are not a feasible solution. Fig. 5 depicts the format of the aligned addend. For the computation of the result, it is sufficient if the aligned addend sticks out 25 bits to the right of the product and 24 bits to the left. The aligner pre-shifts the addend 26 bits to the left, and then shifts it by *shift-amount* to the right. Thus, the *shift-amount* can be limited between 0 and 73.

The aligner has two critical paths. Both start with the selection of the exponents as shown in Fig. 6. 1) The first path computes the shift amount, decodes it into select signals which then control the 4:1 multiplexers performing the actual shifting of the operands. Since the multiplexers are fairly wide, the select signals have to be powered up. 2) The second path performs the shift amount saturation: it checks the shift amount for overflow or underflow, and uses this information to control the last shift stage. The last shift stage uses multiplexer-latches. Due to large setup requirements for the select signals at the multiplexer-latches, type-C, the control signals for the last shift stage are critical.

In order to reduce the delay of the first critical path the aligner uses a sum-addressed scheme, which does adding and decoding of the shift amount in parallel. The shift amount information is acquired by a 4:2 compressor, which generates "sum" and "carry" in a carry–save redundant form. In a conventional design an adder adds these two numbers to get the shift amount. Each two-bit information of the adder result (00, 01, 10, 11) is then decoded into a hot-1 4-bit-wide selection signal for the multiplexers (1000, 0100, 0010, 0001). The sum-addressed scheme,

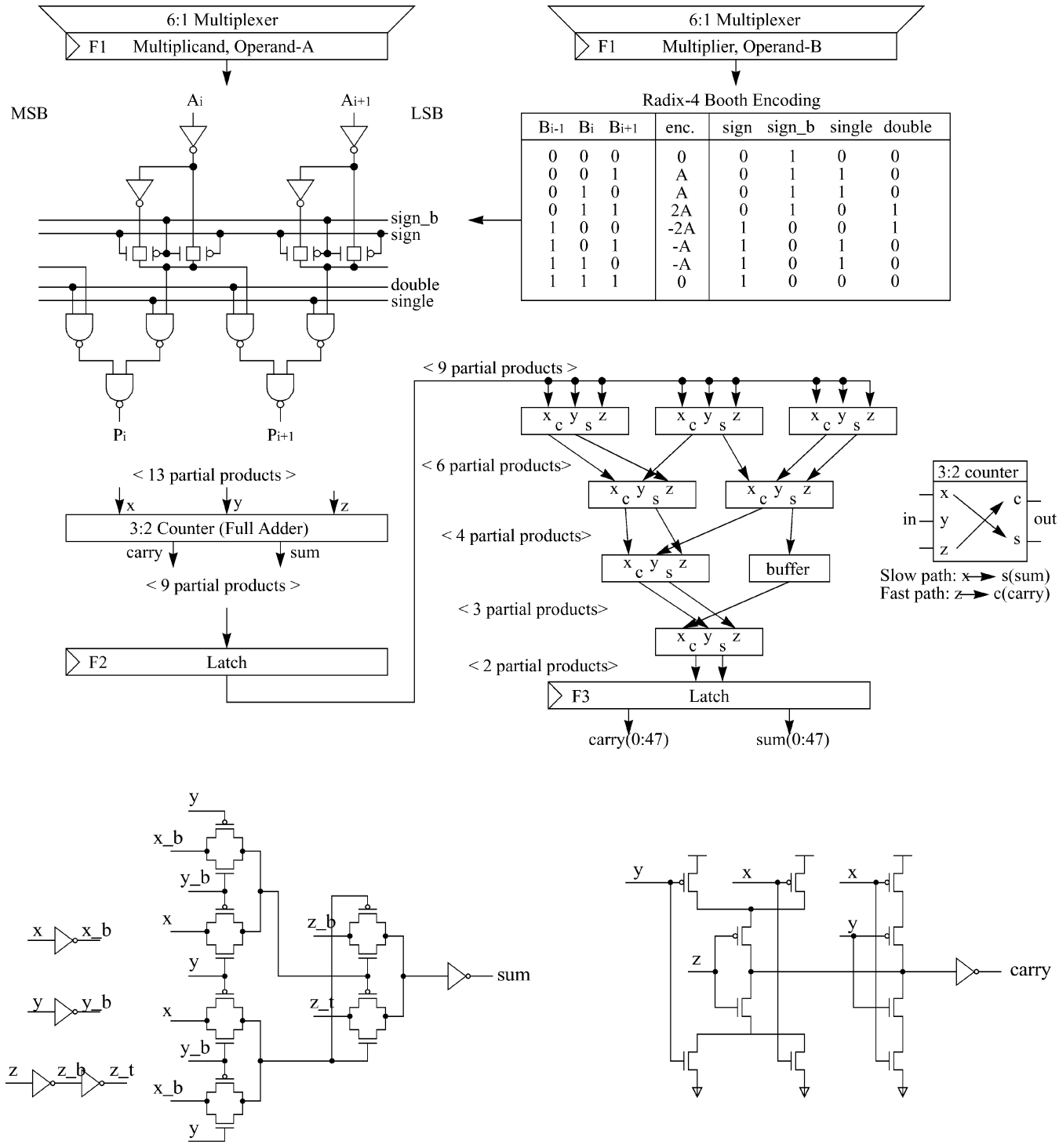


Fig. 4. Radix-4 Booth encoding table and Booth multiplexer circuits are shown. The reduction tree consists of carry–save adders. A 3:2 counter (full adder) is implemented with transmission gate and static AOI circuits.

shown in Fig. 7, uses 2-bit sum-decoders and a separate block carry generation. The sum-decoders generate for every 2-bit block the hot-1 multiplexer selection signals assuming a carry and no carry into the block. Note, that the carry case can be derived from the no-carry case by a 1-bit rotation. The output of the block carry generation is used to select one of the multiplexer selection signals. The block carry generation is usually faster than a general adder.

The main 4:1 multiplexer stages of the aligner, which are shown in Fig. 8, consist of four-input transmission-gate multiplexers (TG-MUX). Its output inverter driver has a better driving capability over the long horizontal wires than that of the AOI-type multiplexer. The short cycle time (e.g., 11 FO4) forces the insertion of nonscan staging latches in the middle of the select signal generation. To solve the reliability problem, two circuits have been implemented. At the local clock buffer, the

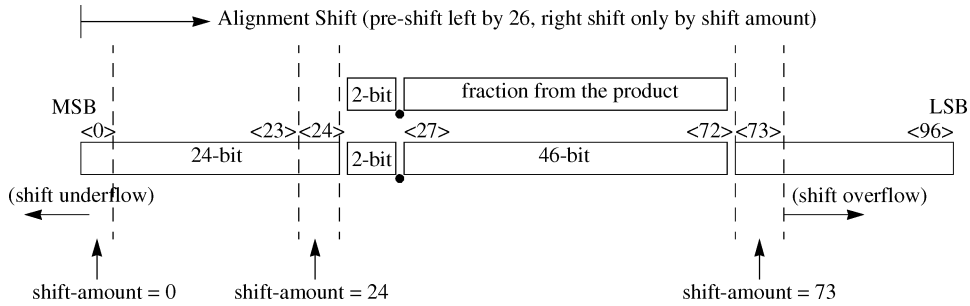


Fig. 5. Alignment of the addend fraction.

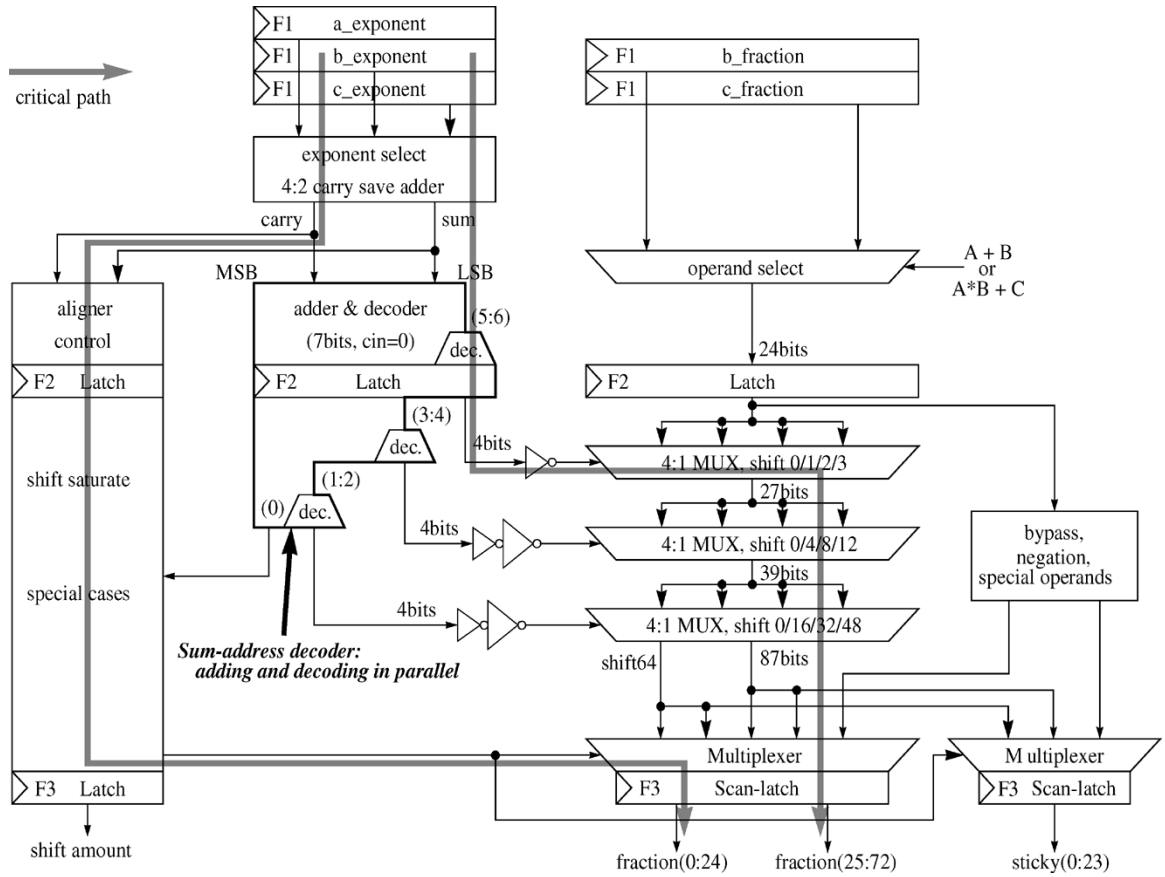


Fig. 6. Two critical paths in the aligner block diagram.

latches are forced to start the latching function as soon as possible when the VDD voltage level becomes stable and when the first clock (nclk) signal arrives. Also the fraction is gated with the power-on-reset signal (por_b) to ensure all data-inputs of the multiplexers having the same logical value, one or zero, to prevent a direct current path.

The aligner uses a 12-track/bit image in layouts. The 97-bit intermediate shifted result does not fit into the limited width of the data flow. Therefore, the 97-bit data are divided into 25-bit, 48-bit, and 24-bit data and folded into two rows.

D. Fraction Adder

The adder is a sign magnitude adder which always produces a positive result. The adder receives the product from the multiplier in a carry–save format (i.e., it receives two partial products, sum and carry) and the aligned addend from the aligner. A 3:2 counter compresses the two partial products and the aligned addend to a sum and carry vector. If the instruction requires subtraction the aligner provides the negated addend. The aligned addend is a 97-bit vector whereas the product is only a 48-bit vector. The aligned addend has additional 25 leading and 24 trailing bits. Therefore, the adder is split into an incrementer, adder, and sticky logic [12].

If the signs of addend and product are the same, the result fraction can be computed by a standard addition of the fractions. If the signs are different, an end-around-carry (EAC) scheme [13] has to be used for the addition of addend and product.

Due to the sign-magnitude representation, the main adder computes the sum or absolute difference of its inputs using the end-around-carry concept. Let x and y be two numbers, and

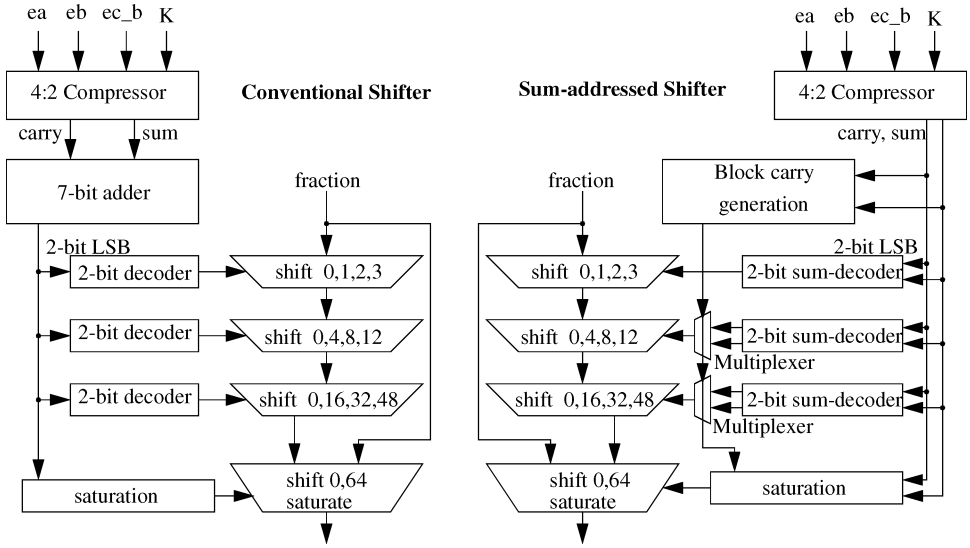


Fig. 7. Conventional and sum-addressed shifter diagram.

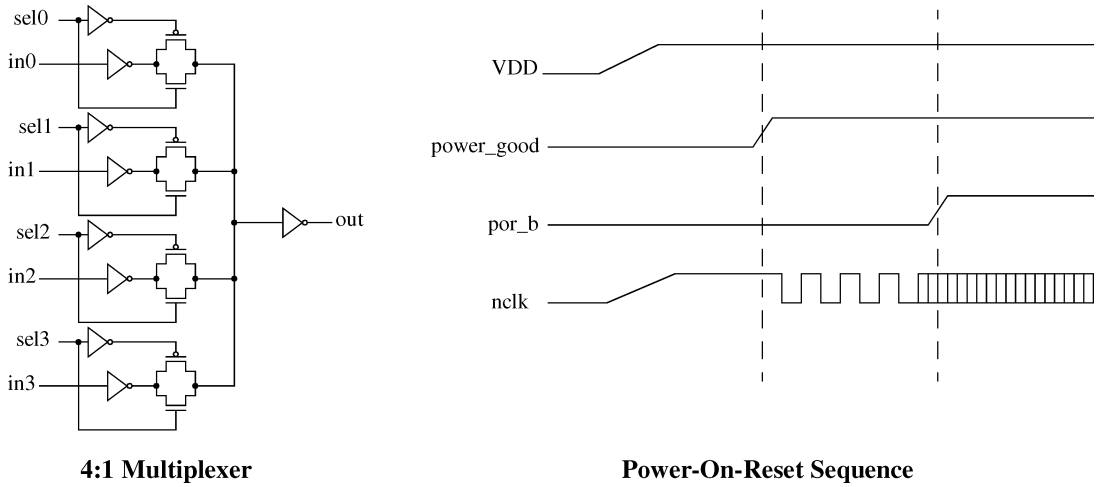


Fig. 8. Main shifting stage consists of four-input TG-MUXs. POR sequences are used to protect the direct current paths at the four-input TG-MUXs.

let $cout$ be the carry-out of $x+!y$, where $!y$ indicates the one's complement of y. The sum or absolute difference of x and y can then be expressed as $x+y$ for addition and as $(x+!y+cout)$ *XOR* $cout$ for subtraction.

Since adding the carry-out to the adder result would be way too slow, the main adder generates sum $(sum0)$ and $sum+1$ $(sum1)$. Instruction type, shift amount at the aligner, the sticky bit, and the carry-out of the adder determine whether $sum0$ or $sum1$ is chosen, and whether complement of the result is needed.

The 25-bit MSB part of addend is passed to an incrementer which generates itself $(sumi0)$ and MSB $+1$ $(sumi1)$, and re-complements both results on a subtraction. The selection of $sum0$, $sum1$, $!sum0$, $sumi0$, and $sumi1$ is implemented with the special six-port MUX latch, type-C. The path delay of this multiplexing is hidden by the latch insertion delay. The computation of the select signals based on the carry-out is the critical path of the adder, as shown in Fig. 9. An extra carry-only adder is used to speed up signal carry-out. The special six-port MUX latch also implements the first multiplexing stage of the normalizer which performs a 25-bit left shift if $sumi0$ is all zero.

To avoid fan-out problems, the main-adder and the special carry generation adder both use a Kogge–Stone parallel prefix adder [13] with binary lookahead carry adder structure [14]. The block diagram of adder is shown in Fig. 10.

E. Leading Zero Anticipator

The leading zero anticipator (LZA) determines the number of leading zeros in the fraction adder result. The fraction adder comprises an incrementer and a compound adder. Either both of them contribute to the result or only the result from the compound adder is used. The computation of the number of leading zeros is different for the two cases: 1) In the case that incrementer is not used and hence the complete result comes only from the compound adder, the LZA estimates the number of leading zeros in the adder output. The LZA receives the output of the merging 3:2 counter in front of the fraction adder. The LZA might overestimate the number by one. 2) In the case that adder result is composed of the incrementer and compound

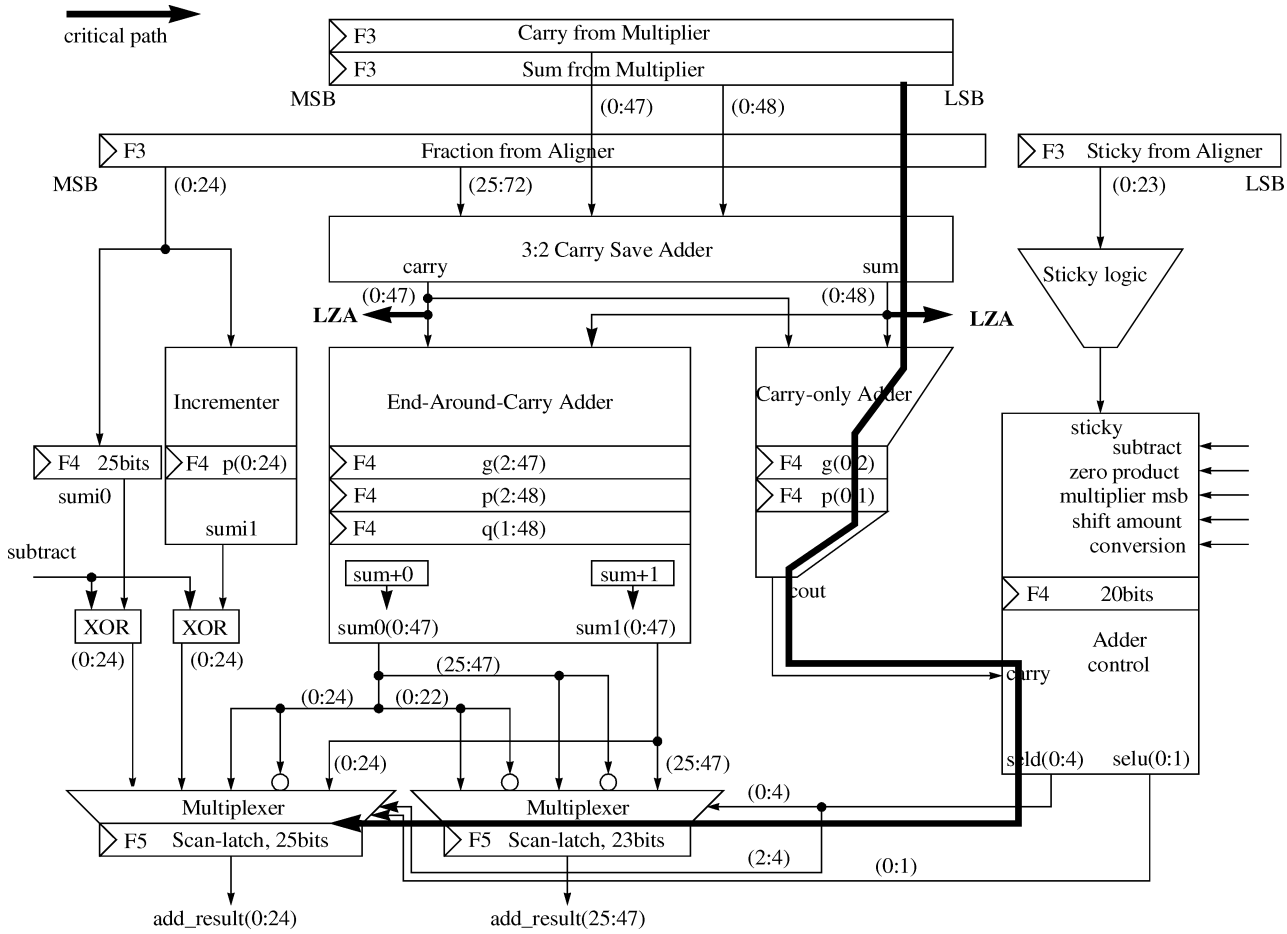


Fig. 9. Critical path of the adder: generating carry-out signal with selection signals at multiplexer-latch.

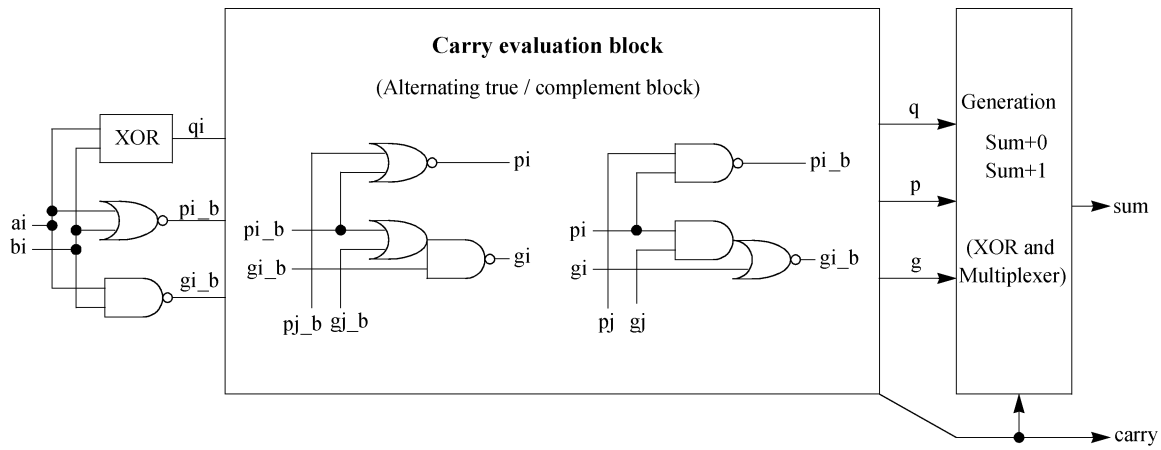


Fig. 10. Binary lookahead carry adder structure is used to build the end-around-carry adder with alternating true/complement block.

adder result, an estimate of the leading zeros can be obtained from the aligner shift amount.

In the case 1), the first step of the LZA is to compute generate, G (NAND), propagate, P (XOR), and kill, K (NOR), signals for each bit. The LZA can then detect edges in the sum using P, G, and K signals of the current bit position, and the P, G, and K signals of the two preceding bit positions. Since LZA requires the two preceding bits to detect an edge, it is necessary to extend the sign bit of the LZA/adder inputs.

The LZA generates an edge vector with set bits at the location of each possible edge in the sum. This edge vector is produced at the first pipeline stage of the LZA. The normalization shift amount is calculated by counting the number of leading zeros in the LZA edge vector. This is done in the second pipeline stage of the LZA. The final leading zero count is either exact or too large by one. Therefore, the LZA provides a mask vector which is used in the normalizer and the rounder to determine whether the estimate was correct.

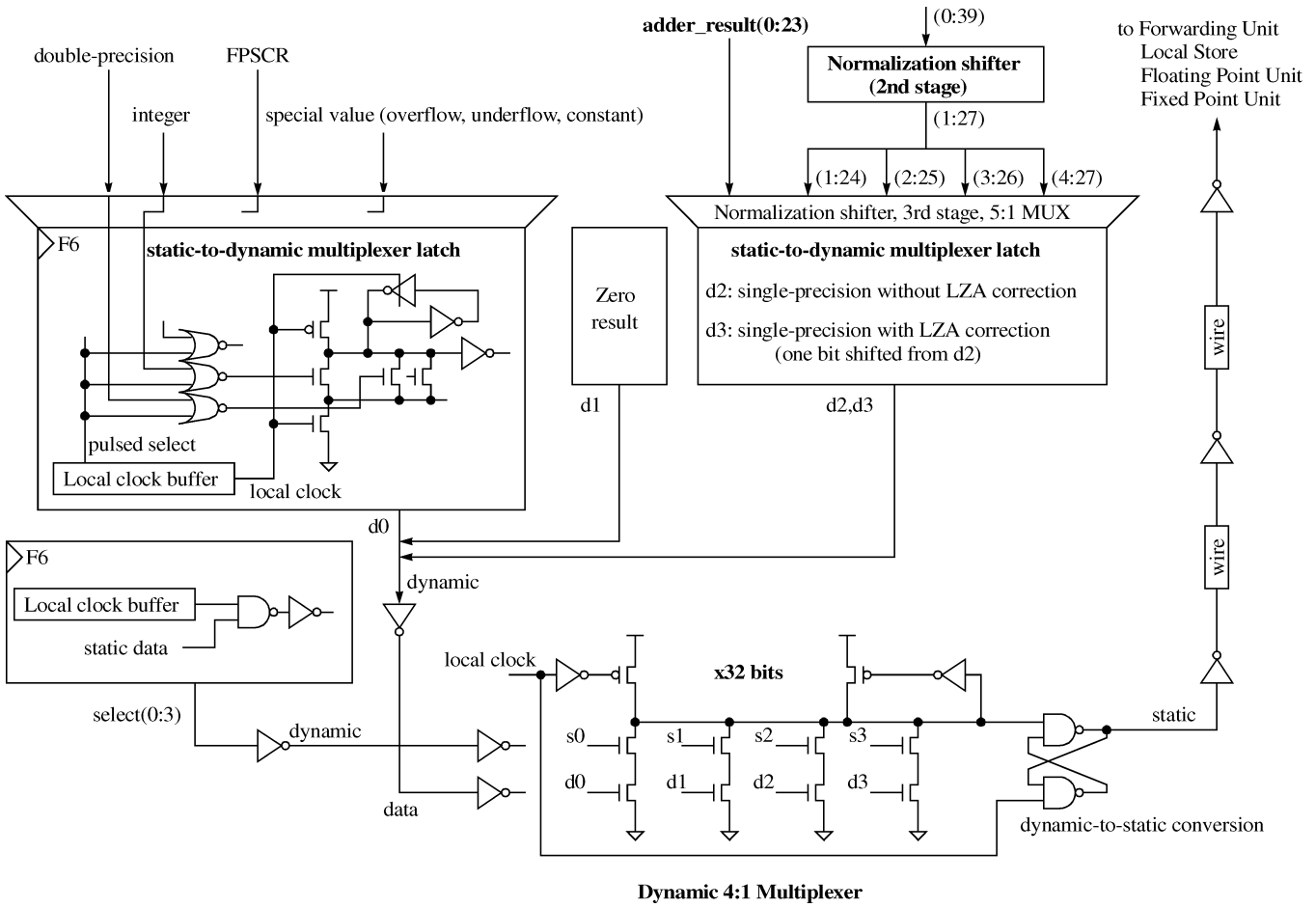


Fig. 11. Static-to-dynamic multiplexer latch generates the dynamic signals, and then the dynamic result-MUX converts the dynamic signal to static forwarding signals.

F. Normalizer

The normalization shifter receives the 48-bit fraction from the adder and the shift amount $(lzsha)$ from the LZA. There can be up to 48 leading zeros, but the shift amount is at most 47. In the case of 48 leading zeros, the whole fraction is zero; this is handled as a special case in the rounding logic. The normalizer requires two types of corrections, one is an LZA correction and the other is an adder correction. In case of an incorrect estimate by the LZA, the most significant bit got shifted out. In the simplest solution, the normalizer shifts the fraction by $lzsha - 1$ only producing a 25-bit normalized fraction. The MSB of the normalizer output indicates whether a correction is needed or not. If the MSB is one, then the LZA overestimated the number of leading zeros. If the MSB is zero, then the LZA predicted the correct number of leading zeros and one more bit needs to be shifted out. With this simple method, the correction cannot start until the results of the normalizer are available. Instead of waiting for the result of the normalizer, the mask vector from the LZA is used for faster correction. The mask vector goes to the bit-by-bit AND gate with the outputs of the adder. The output vector of the AND gate is fed to a zero checker. The mask vector has a single one at the position of $lzsha - 1$. If the LZA estimate is correct, the leading one is at the position $lzsha$ and the zero checker delivers a zero. If the estimate is off by one bit, the leading one is positioned at $lzsha - 1$ and the zero checker delivers one. The result multiplexer performs the LZA correction by either selecting the bits 0 to 23 or the bits 1 to 24 of the normalized fraction depending on the zero checker output.

The last stage of the adder already performs a first step of the normalization shift, selecting 48-bit data out of 73-bit data. When the 73-bit adder result has exactly 24 leading zeros, the adder selects the least significant 48 bits, loosing the leading one. In this case the adder sets the special signal $add\_correction$ to one. With $add\_correction = 1$, the shift amount is ignored, and the normalized result becomes a 1 followed by bits 0 to 22 of the adder result. The normalization shift is done in four stages including one stage in the adder, using the same 4:1 multiplexers from the aligner.

G. Result Multiplexer and Rounding

The FPU only supports round toward zero and nontrap exception conditions. That simplifies the rounding step considerably; the fraction is truncated and no exponent wrapping is needed. The fraction rounder is implemented as a result-multiplexer (result-MUX). The exponent rounder (ERND) adjusts the exponent, detects exceptions and controls the result-MUX. To match the fast rounder, the block delay of the exponent rounder was improved by 15FO4 using highly optimized checkers and dynamic circuits. The exponent rounder uses multiple pre-computed exponent copies such as e, $\mathrm{e}+1$, and $\mathrm{e}-1$. A dynamic MUX-latch selects the appropriate copy of the exponent rounder results.

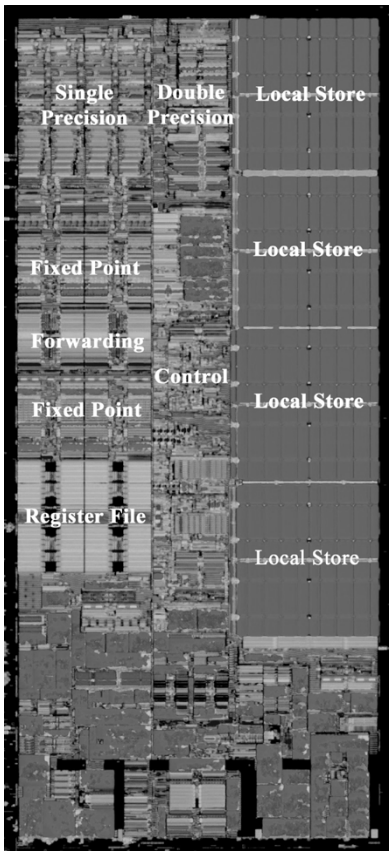


Fig. 12. SPE die photo with major functional unit location.

The result-MUX selects between four inputs: 1) results from integer or double precision; 2) a zero result; 3) single-precision results which need LZA correction; and 4) single-precision results which do not need LZA correction. Since the last cycle has only 5FO4 budget due to the result forwarding, a dynamic 4:1 MUX, which is shown in Fig. 11, was designed.

The high-performance latch (type-C) is modified to generate dynamic signals. Without set-reset latch (cross-coupled NAND2) in the type-C latch, the internal signal is dynamic. That signal is brought to the data inputs, $d0 - d3$, of the dynamic 4:1 multiplexer. The select signal inputs to the dynamic multiplexer, $s0 - s3$, are generated by gating the select signals from the exponent rounder with local clock signals from local clock buffer. The dynamic 4:1 multiplexer which is shown in Fig. 11 has an un-footed domino configuration. The internal dynamic signal at the 4:1 multiplexer is converted to a static signal through the cross-coupled NAND2 gate. Since operand data flows use 16-track/bit image, the dynamic 4:1 multiplexer has 16-track/bit image while other blocks in the normalizer and exponent rounder use 12-track/bit image.

IV. Clock Gating to Save Power Consumption

Clock-gating is extensively used to reduce the active power consumption [15]. All the registers have a clock enable terminal at the LCB. Only pipeline stages with valid instructions are activated by controlling the local clock buffer. Also, circuit blocks are clocked down based on instruction type and on operand values, e.g., the multiplier is bypassed for add-type instructions,

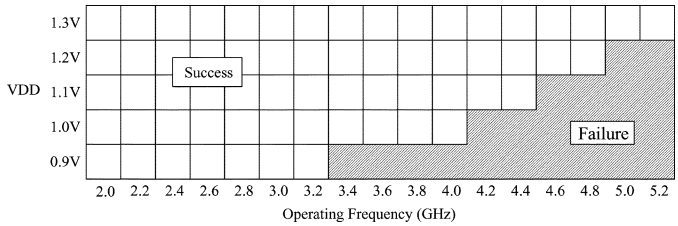


Fig. 13. Voltage versus frequency correlation graph: a synergistic processing element is running single-precision floating-point computations for lighting and transform workloads.

and the aligner is idle for multiply-type instructions. Also, integer instructions make the LZA and the normalizer idle. All those control signals are generated by control logic, which is generally built using fully automated techniques. Control logic macros are synthesized using a standard cell library consisting of only simple gate logic functions except for latches and clock buffer components.

There is a 32-bit bypass bus in the FPU. The integer operand of convert instructions and the addend of integer multiply–add instructions are 32-bit wide. The aligner and the multiplier only receive 25-bit fractions. It requires significantly less hardware to allow 32-bit data in the bypass bus to widen the aligner or multiplier. In the final multiplexer stage of the aligner, the result of the aligner can be replaced by the bypass data. The multiplier also has result multiplexers, which either select the two partial products or select the bypass data and forces the other partial product to zero.

V. Conclusion

The design has roughly 768K transistors in 1.3 mm^2 (0.937 mm $\times$1.391 mm), fabricated in 90-nm SOI technology with eight levels of copper interconnects. The die photo (Fig. 12) illustrates the SPE. The single-precision floating-point unit is located at the top-left corner of the SPE. Correct operation has been observed up to 5.6 GHz, 1.4 V of supply voltage, and 56 °C, delivering a peak performance of 44.8 GFlops. An interpolated hardware power of 1.4 W at 4 GHz (i.e., 32 GFlops), gives 43.75 mW/GFlop. Voltage versus operating frequency correlation of the SPE is shown in Fig. 13 [1], when an SPE is running single-precision floating-point computations for lighting and transforms. The operating frequency is limited by the instruction fetch and operand distribution paths. The actual operating frequency of the floating-point unit alone is expected to be higher.

There are three key enablers for this low-latency, power-efficient, and high-frequency FPU: 1) architecture and implementation are optimized for target applications, trading uncritical functions for overall performance; 2) logic, circuits and integration are codesigned; and 3) the pipeline stages are carefully balanced, achieving the maximum path delay difference of 3% between stages.

Acknowledgment

The authors would like to thank O. Takahashi, B. Flachs, P. Hostee, R. Cook, S. Yong, G. C. Fossum, and B. L. Minor for their design efforts, G. Gervais and the SPU verification team for

support, J. Preiss for editing, and L. V. Grinsven and R. Putney for management.

REFERENCES

[1] B. Flachs, S. Asano, S. H. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano, "A streaming processing unit for a CELL processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2005, pp. 134–135.

[2] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2005, pp. 184–185.

[3] O. Takahashi, R. Cook, S. Cottier, S. H. Dhong, B. Flachs, K. Hirairi, A. Kawasumi, H. Murakami, H. Noro, H. Oh, S. Onishi, J. Pille, J. Silberman, and S. Yong, "The circuits and physical design of the synergistic processor element of a CELL processor," in *Symp. VLSI Circuits Dig. Tech. Papers*, Jun. 2005, pp. 20–23.

[4] M. S. Hrishikesh, K. Farkas, N. P. Jouppi, D. C. Burger, S. W. Keckler, and P. Sivarkumar, "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," in *Proc. 29th Int. Symp. Computer Architecture (ISCA-29)*, May 2002, pp. 14–24.

[5] V. Zyuban, D. Brrok, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski, and P. G. Emma, "Integrated analysis of power and performance for pipelined microprocessor," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 1004–1016, Aug. 2004.

[6] N. J. Rohrer, M. Canada, E. Cohen, M. Ringler, M. Mayfield, P. Sandon, P. Kartschoke, J. Heaslip, J. Allen, P. McCormick, T. Pfuger, J. Zimmerman, C. Lichtenau, T. Werner, G. Salem, M. Ross, D. Appenzeller, and D. Thygesen, "PowerPC 970 in 130 nm and 90 nm technologies," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2004, pp. 68–69.

[7] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. Wellesley, MA: A. K. Peters, Ltd., 2002.

[8] S. M. Mueller, C. Jacobi, H. Oh, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. H. Dhong, "The vector floating-point unit in a synergistic processor element of a CELL processor," in *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH'05)*, Jun. 2005, pp. 59–67.

[9] H.-J. Oh, S. M. Mueller, C. Jacobi, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. H. Dhong, "A fully-pipelined single-precision floating-point unit in the synergistic processing element of a CELL processor," in *Symp. VLSI Circuits Dig. Tech. Papers*, Jun. 2005, pp. 24–27.

[10] 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 Trans. Comput.*, vol. 45, no. 3, pp. 294–305, Mar. 1996.

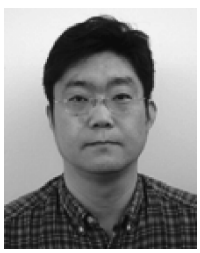
[11] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Mateo, CA: Morgan Kaufmann, 2004.

[12] S. M. Mueller and W. J. Paul, *Computer Architecture, Complexity and Correctness*. New York: Springer, 2000.

[13] B. Parhami, *Computer Arithmetic, Algorithms and Hardware Design*, 1st ed. New York: Oxford Univ. Press, 2000.

[14] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A System Perspective*, 1st ed. Reading, MA: Addison-Wesley, 1985.

[15] O. Takahashi, "The power conscious design of the synergistic processor element of a CELL processor," in *Proc. IEEE Symp. Low-Power and High-Speed Chips (COOL Chips VIII)*, Apr. 2005, pp. 446–457.



Hwa-Joon Oh (S'87–M'98) received the B.S. and M.S. degrees from Yonsei University, Seoul, Korea, in 1987 and 1989, resepctively, and the Ph.D. degree from Michigan State University, East Lansing, in 1996.

From 1996 to 1998, he was with Hyundai Electronics America, San Jose, CA, where he was involved in research and development of DRAM designs. In 1998, he joined IBM Austin Research Laboratory (ARL), Austin, TX, where he worked on POWER4 microprocessor design. From 2000 to 2001, he briefly worked on short random cycle embedded memory design for IBM microelectronics. Currently, he is with the IBM System and Technology Group with the Sony-Toshiba-IBM Design Center (STI-DC), Austin, TX, where he is involved in architecture, logic, circuits, and physical implementations of the CELL microprocessor. He has authored or coauthored several journal papers and patents. His main research area is artificial neural networks, SRAM and DRAM design, and broadband microprocessor design.

Silvia M. Mueller received the B.Sc. and M.Sc. degrees in mathematics and the B.Sc. and Ph.D. degrees in computer science from the University of Saarland, Saarbuecken, Germany.

From 1992 to 1993, she was a PostDoc at University of California, Berkeley, where she was involved in the research of a supercomputer for speech recognition. Until fall 1999, she held teaching and research positions at the University of Saarland and became a Privatdozent at the Computer Science Department. She also spent two short sabbaticals at the Computer Science Department of MIT, Cambridge. She conducted research in several fields of computer architecture, studying schedulers for pipelined processors with out-of-order execution and precise interrupt handling, IEEE compliant floating-point units (FPUs), and supercomputers. Her key contributions are on quantitative analysis, performance prediction, and construction and correctness of processor designs. She joined IBM's Server Division in Boeblingen, Germany, in 1999 as a Staff Hardware Engineer where she was involved in performance analysis and processor studies for the S/390 mainframe. From 2001 to 2003, she was on an international assignment in Austin, TX, joining the Sony-Toshiba-IBM Design Center. She was the lead logic designer for the FPUs of the synergistic processing engine of the CELL processor. Her logic team was responsible for the micro-architecture, logic design, verification, and floorplan of the FPUs. Since 2004, she has been leading two multi-site teams developing high-performance FPUs and vector media extension units (VMXs) for the PowerPC processors. In this role, her major focus is on power–performance tradeoffs, area reduction, and a more efficient way to handle the design complexity. She is an IBM Senior Technical Staff Member, a Privatdozent at the Computer Science Department of the University of Saarland, and an affiliate of IEEE Computer Society. She is the coauthor of two textbooks on computer architecture, many technical publications, and several patents.

Christian Jacobi received the Master's degree in computer science (Diplom-Informatiker) and the Doctoral degree in computer science from Saarland University, Saarbruecken, Germany, in 1999 and 2002, respectively.

He has worked on formal verification techniques, mainly for floating-point units. He joined IBM in 2002 at the development laboratory in Boeblingen, Germany, where he worked on floating-point logic design, logic verfication, and physical implementation for various IBM microprocessors. He is now working on cache designs for future zSeries processors. He is the author of several journal and conference papers.

Kevin D. Tran received the B.S.E.E. degree from the University of Texas at Austin in 1995.

He was with a startup company in Austin, TX, from 1995 to 1997, where he was engaged in the design of a 200-MHz microprocessor. He was with the SmartCard America group and Somerset Design Center, Motorola, Austin, from 1997 to 2000, where he was a memory circuit designer. He joined IBM Microelectronics Division, Austin, in 2000, where he has been engaged in circuit design of register files and floating point units.

Scott R. Cottier (M'05) received the B.S. degree in electrical engineering from Texas A&M University.

He is responsible for memory arrays in the SPE of the CELL processor at the Sony-Toshiba-IBM Design Center, Austin, TX.

Brad W. Michael received the B.S. degree in computer engineering from Iowa State University in 1989. He joined IBM in that year, and is now involved in microprocessor logic design in the STI Design Center in Austin, TX.

Hiroo Nishikawa received the Bachelor's and Master's degrees in instrumentation technology from Kobe University, Kobe, Japan, in 1980 and 1982, respectively.

In 1982, he joined IBM Japan, working on EDA tools, analog circuit design, and logic design. From 2001 to 2003, he designed the floating-point unit of the CELL processor. He is currently with IBM Global Services, Yasu, Japan.

Yonetaro Totsuka received the B.S. and M.S. degrees in information science from the University of Tokyo, Japan, in 1993 and 1995, respectively.

Since 2000, he has been with Sony in Tokyo. He worked at the STI Design Center in Austin, TX, on CELL BE microprocessor design in 2002–2005. His interests include processing architecture and its design and implementation.

Tatsuya Namatame received the B.S.E.E. degree from the Shibaura Institute of Technology, Tokyo, Japan, in 1992.

He joined SEGA Enterprises, Ltd., Tokyo, in 1992, in the Hardware Research and Development Department, where he was engaged in development of the game console system Sega Saturn, mainly working on bring-up of the whole system. In 1997, he was involved in development of the next game console system, Dreamcast. His main role there was to design the bus interface in the 3-D graphics chip. From 1999 to 2000, he was engaged in designing the pixel setup engine for 3-D graphics and also studying the floating-point arithmetic algorithm. He moved to Toshiba Corporation Semiconductor Company, Kawasaki, Japan, in 2001, and then joined the Sony-Toshiba-IBM (STI) Design Center, Austin, TX, to work on the development of the CELL processor as a member of the Floating Point Unit team, where he was engaged in logic design and verification of the FPU. He is currently working on the CELL processor derivative chip design.

Naoka Yano received the Bachelor of Arts and Science degree in pure and applied sciences from the University of Tokyo, Tokyo, Japan, in 1991.

She joined the Semiconductor Device Engineering Laboratory, Toshiba Corporation, Kawasaki, Japan, in 1991. She was engaged in the research and development of high-performance microprocessors, including the Emotion Engine. In 2001, she moved to Toshiba America Electronic Components, Inc., where she was involved in the development of the CELL processor. She is currently with the Broadband System LSI Development Center, Semiconductor Company, Toshiba Corporation, Kawasaki, Japan.

Takashi Machida received the B.S. and M.S. degrees in chemistry from the Tokyo Institute of Technology, Tokyo, Japan, in 1996 and 1998, respectively. He joined the RISC Processor Development Group, Toshiba Corporation, Kawasaki, Japan, in 1998. From 1998 to 2001, he engaged in the development and product engineering of the Emotion Engine. From 2001 to 2004, he was with Toshiba America Electronic Components, Inc., where he was involved in the development of the CELL processor. He is currently with the Broadband System LSI Product Engineering Group, Toshiba Corporation Semiconductor Company, Kawasaki, Japan.

Sang H. Dhong (M'76–SM'99–F'01) received the B.S.E.E. degree from Korea University, Seoul, Korea, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley.

He joined IBM's Research Division, Yorktown Heights, NY, in 1983 as a Research Staff Member where he was involved in the research and development of silicon processing technology, mainly bipolar devices and reactive-ion etching (RIE). From 1985 to 1992, he was engaged in research and development of DRAM cell structures, architectures, and designs, spanning over five generations of IBM DRAMs, from 1-Mb DRAMs to 256-Mb DRAMs. His key contributions in this area are high-speed DRAMs, low-power DRAMs, and NMOS-access transistor trench DRAM cells. After spending three years in development of one of IBM's PowerPC microprocessors as a Branch/Icache circuit team leader of 15 designers, he worked on a simple but fast processor core based on the PowerPC architecture and on high-speed embedded DRAM (eDRAM) in the Austin Research Laboratory of the IBM research division, leading, managing, and growing the high-performance VLSI design group to a peak of 25 people from 1995 to 1999. The work resulted in setting a major milestone for the microprocessor industry by prototyping 1-GHz PowerPC processors. Also, his work on the high-speed eDRAM provided the justification for the logic-based eDRAM foundry/ASIC technology offering by IBM as well as the design basis for eDRAM macros of DRAM-like density with SRAM-like high speed. Since becoming the chief technologist of the Austin Research Laboratory in 1999, he has worked in three areas: fast low-power embedded PowerPC, super-pipelined multi-gigahertz PowerPC servers, and high-speed eDRAM. In 2000, he joined the Sony-Toshiba-IBM (STI) Design Center as one of the key leaders, primarily concentrating on a 11-FO4 coprocessor design, streaming process (SPE). As the partition leader of SPE team, he defined, executed, and delivered the technology, circuits and latch styles, floor plan, and basic lower-power micro-architecture and led technically a multi-discipline team of 70 or more engineers. Currently, he is the Chief Hardware Engineer/SPE Partition Lead, responsible for productization of BE chip from the STI center side. In this role, his major focus is on power-frequency yield tradeoff, interacting and directing manufacturing, PE, and design teams in a matrix organization of more than 100 engineers. He holds more than 125 U.S. patents as well as many technical publications.

Dr. Dhong is an IBM Distinguished Engineer, a member of IBM Academy of Technology, and a Fellow of IEEE, and has received five Outstanding Innovation/Technical Achievement Awards from IBM.