2011 20th IEEE Symposium on Computer Arithmetic

The POWER7 Binary Floating-Point Unit

Maarten Boersma, Michael Kröner, Christophe Layer, Petra Leber, Silvia M. Müller, Kerstin Schelm
IBM Research & Development, Schönaicher Str. 220, D-71032 Böblingen, Germany
{mboersma,mkroener,clayer,pleber,smm,schelmkn}@de.ibm.com

***Abstract*—The binary Floating-Point Unit (FPU) of the POWER7 processor is a 5.5 cycle Fused Multiply-Add (FMA) design, fully compliant with the IEEE 754-2008 standard. Unlike previous PowerPC designs, the POWER7 FPU merges the scalar and vector FPUs into a single unit executing three floating-point instruction sets: the single and double precision scalar set, the single precision VMX vector set, and the new single and double precision VSX vector and scalar set. Due to a compact buffer-free floorplan and several optimizations in the data and control flow, the streamlined POWER7 FPU achieves a factor of 2 area reduction over the POWER6 design, beyond the normal technology shrink. This results in a very power and area efficient FPU design, supporting a chip frequency of 4.14GHz. A single 64-bit FPU instance measures only 0.26mm^2 in 45nm CMOS SOI.**

***Keywords*-FPU; heterogeneous precision support; subnormal check; IEEE 754-2008 compliant; high-frequency design;**

I. Introduction

All major server processors nowadays have a scalar and a vector FPU, where the vector FPU is part of the Vector Media eXtension (VMX) [1–3]. Originally, the vector extension has been introduced for speeding up media and graphics applications, for which single precision floating-point (SP) was good enough. In recent years, most vector extensions have added double precision floating-point (DP) to target compute-intensive numerical applications. Introduced in the POWER7 processor [4–6], VSX is the DP vector and scalar extension for PowerPC.

When vectorizing floating-point (FP) applications, there is usually a combination of vector and scalar code segments which work on the same data. This requires a smooth data exchange between the scalar and vector unit, and therefore suggests VSX to be an extension of scalar DP. On the other hand, the scalar registers are only 64-bit wide, whereas the vector SP and vector DP both require wider 128-bit registers. Thus, VSX should actually be an extension of both scalar DP and vector SP, and the vector and scalar FPUs should be merged into a single unit. This eliminates the overhead for vector-scalar data transfer through storage. Implementation wise, such a merged FPU has a twofold benefit for chip area and power consumption: In a conventional design, a processor core which can issue two DP instructions, no matter whether they are scalar or 2-way SIMD vector, would need a total of six FPUs, i.e., two for scalar and four for vector, whereas with a merged vector-scalar FPU, the core can execute the same amount of FP instructions with only four FPUs. In an out-of-order processor core, each register file needs a large rename space. When merging the vector and scalar FPUs, the combined register file needs to hold both sets of architected registers, but the rename space can be shared between the vector and scalar operations, reducing the overall register file size significantly. Having the VSX as an extension of the scalar and vector FPU forces the merged register file to be architected, such that not only it behaves for old scalar FPU code and for VMX code like for the original split register files, but also that the new VSX instructions can access all three register sets. Besides being the first implementation of the VSX extension, the POWER7 FPU had challenging area and power targets.

The POWER6 core [7] has two binary 64-bit scalar FPUs [8, 9] and a vector FPU with four 32-bit FPU instances [1] and a vector fixed-point (FX) multiplier. Scaled into 45nm technology, these units would account together for about 2.5mm^2, and a single 64-bit FPU for about 0.46mm^2. The POWER7 replaced all these units by four 64-bit FPUs, with 0.26mm^2 each. Thus, a single 64-bit instance of the POWER7 FPU is 1.8 times smaller than the POWER6 design. The total FPU area even shrank by a factor 2.5, despite increased functionality and a two times higher DP throughput.

II. Architecture and Micro-Architecture

The POWER7 Vector-Scalar Unit (VSU) implements the original scalar binary FPU architecture and the VMX and VSX extensions. Architecturally, it consists of a 128-bit wide vector register file VSR with 64 entries, a scalar and a vector binary FPU and various vector FX units. The latter perform vector FX arithmetic, data permutations and bit manipulations.

A. Embedding the Register Files

The 128-bit wide VSR combines the 32 64-bit wide Floating-Point Registers (FPR) and the 32 128-bit wide VMX Vector Registers (VR) into a single uniform register set. As depicted in Fig. 1, the FPR is mapped to the bits 0 to 63 of VSR[0:31] and the VR is mapped to VSR[32:63]. The VSX scalar and vector instructions can access all 64 VSR. This doubles the architected register space compared to previous designs and allows for more aggressive compiler optimizations. For backward compatibility, the original scalar FPU instructions can only access bits 0 to 63 of

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.21

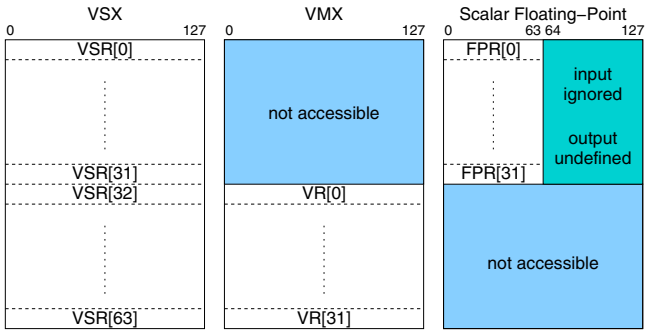

Figure 1. Mapping of the three FPU register files onto the VSR

VSR[0:31]. On a scalar read access, bits 64 to 127 of the source registers are ignored, and after a scalar write, bits 64 to 127 of the target register become undefined. The POWER7 implementation partitions the VSR in two double-word parts, one for bits 0 to 63 and the other for bits 64 to 127, which can be updated separately. Scalar operations update the high part (bits 0 to 63) of the register only, and a valid flag is indicating whether the low part of a given register is valid or undefined. When a vector operation reads a register with invalid bits 64 to 127, these bits are cleared. The valid bit for the second double word allows the VSU to save power on scalar writes, since only half the register file is updated, and still prevents undefined data from being observed by other instructions or programs. This is important for application isolation and for data protection.

B. Heterogeneous Precision Arithmetic

The VMX and VSX vector FP instructions support 2-way SIMD double-precision data and 4-way SIMD single-precision data in the VSR. Each SP and DP elements are in 32-bit and 64-bit IEEE-754 memory format respectively [10]. For the vector instructions, register file and memory use the same data format. For scalar FP instructions, data are stored in 64-bit DP format in the register file, no matter whether SP or DP. Thus, when programs combine SP scalar and vector register data, explicit data conversions between 64-bit scalar SP and 32-bit vector SP format are needed.

On a scalar SP-load instruction, the memory data in 32-bit SP format is converted to the 64-bit DP format before being written to the register file, whereas a scalar SP-store instruction converts the 64-bit DP register file data to 32-bit SP format before writing to memory. The operands of scalar SP arithmetic instructions are 64-bit DP register file data. The intermediate result is rounded to SP with a single rounding error and then converted to 64-bit DP format before being written to the register file. All scalar arithmetic FP operations come in two flavours, either rounding to SP or DP precision. Since 64-bit operands can be SP or DP values, this is a straightforward way for supporting the heterogeneous precision arithmetic required by the IEEE-754-2008 standard [10].

C. Efficient Support of Subnormal Numbers

Given the specific register file format for scalar FP data, conversion from SP data into 64-bit DP format occurs within any scalar SP load or arithmetic operation. For SP numbers in the normal SP data range, the conversion is very fast and achieved by padding the fraction with 29 trailing zeros and rebiasing the exponent. The new DP exponent E' is the SP exponent E plus the difference of the two biases:

$$E' = E - \text{bias}_{SP} + \text{bias}_{DP} = E - 127 + 1023 = E + 896$$

The bias difference 896_{10}=380_{16} can be applied by inserting three copies of the inverse of the most significant exponent bit: $E' = \langle E_0 \bar{E}_0 \bar{E}_0 \bar{E}_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 \rangle$.

For SP subnormal numbers, the exponent is rebiased to 381_{16}, then the fraction is normalized and the exponent is adjusted by the normalization shift amount. This would require an additional normalization stage for SP results on the FPU execution pipeline and on the load datapath. Former implementations [11] avoid the extra normalization step by storing SP subnormal numbers in an intermediate 65-bit format including the implied integer bit, in which the exponent is rebiased to 381_{16} and the fraction is padded with 29 trailing zeros. This additional information makes it possible to distinguish between SP subnormal and DP normal numbers, both with an exponent of 381_{16}. This scheme worked well when scalar FP data were only consumed by the scalar FPU. Now that scalar DP data are also accessed by vector FX and vector SP instructions, the register file needs to store the data with the exact architected bit pattern. Thus, the POWER7 VSU and FPU need at least an indicator to distinguish between three data types: vector data in proper architected format (type-1), scalar DP data in intermediate format matching the architected pattern (type-2), and scalar DP data not matching the architected pattern (type-3).

Data of the first two types can be shared among all the subunits of the VSU, but type-3 data are only understood by the FPU and actions need to be taken before they can be used by other parts of the VSU. Instead of using the implied bit, POWER7 adds to each register a "dirty" flag which is zero for type-1/2 data and one for type-3 data. Type-1/2 are held in architected format. For type-3 data, the dirty flag replaces the implied bit enabling the FPU to properly handle data when performing scalar FP or vector DP operations. When such an operand is either accessed by another unit, used as vector SP operand, or as FX operand, then a flush-request is triggered. The hypervisor then executes normalizing vector-DP-moves on every register file entry, converting type-3 data into architected format. After that, the instruction causing the flush request is reissued.

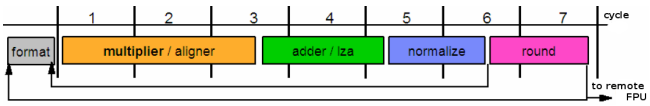

Figure 2. Pipeline Partitioning of the POWER7 floating-point unit

III. Pipeline Structure

The POWER7 VSU comprises four DP FPUs, a vector FXU and a 128-bit wide permute unit, all assigned to two execution pipes. Both execute scalar SP and DP instructions, as well as vector DP instructions. Pipe 1 also executes permute instructions, hence using one or two FPUs. Pipe 0 also executes vector FX instructions and vector SP instructions, hence requiring one FPU for scalar code, two for vector DP, and all four FPUs for vector SP. The FPU data-flow is derived from the POWER6 design, in which 64-bit FX multiplications were also executed in addition to scalar FP operations. These required 33 partial products in the Booth multiplier, i.e., 6 more than needed for a conventional 53×53-bit DP multiplier. The extra partial products caused an additional compression stage in the multiplier reduction tree. With a very high frequency design, this resulted in inconvenient cycle boundaries throughout the whole pipeline (Fig. 2). The POWER6 FPU therefore achieves the performance critical 6-cycle back-to-back latency only by forwarding unrounded results, only within an FPU instance. Forwarding to the remote FPU takes an extra cycle and the fully rounded result can be used after 8 cycles at the earliest. Forwarding unrounded results requires a fix-up step inside of the multiplier and cannot handle overflow and underflow cases. These forwarding cases require flushes.

In order to simplify the issue rules for the out-of-order execution, the POWER7 core requires symmetric forwarding latencies for local and remote FPU. For performance reasons, a 6-cycle back-to-back latency is mandatory and flush conditions must be kept to a minimum. As a consequence, the POWER7 FPU needs to provide the fully rounded result after 5.5 cycles, so that there is half a cycle for distributing it to local and remote FPU. This requires shortening the POWER6 FPU pipeline by roughly one stage. The 64-bit FX multiplication is moved back to the scalar FXU, to reduce the number of partial products and consequently reduce the delay of the multiplier. Though, without any extra partial product, the short vector FX multiplies are handled by the FPU. This way, much more convenient cycle boundaries could be achieved, as depicted in Fig. 3. Multiplier and aligner fit in the first two stages of the pipe, followed by two stages for the addition, one stage to normalize the intermediate result, and one stage for rounding and packing. Such a pipeline structure allows forwarding at the end of the sixth cycle.

The adder handles rounding and saturation for the integer results, which is not needed on the FP arithmetic path, so

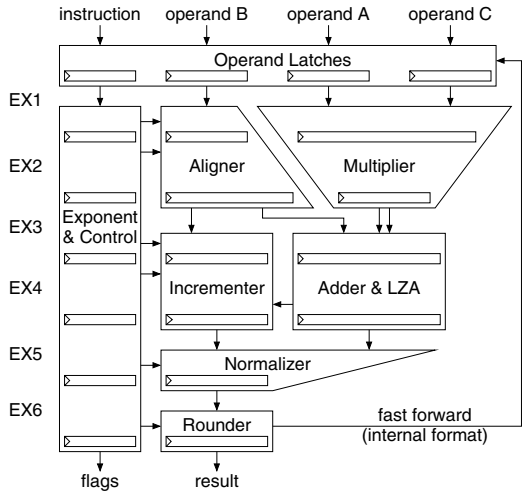

Figure 3. Pipeline Diagram of the POWER7 floating-point unit

it can be removed from the critical path. The integer results are not needed for fast forwarding and are therefore muxed into the final pack step after the exit point for 5.5-cycle forwarding. The normalizer and leading-zero-anticipator precompute the sticky bits and carry-out information for different target precisions. This is done by compressing the shifted out bits in parallel with the actual shifting. Also the leading ones are reduced to speed up the carry propagation for a potential increment during rounding. In addition it precomputes exponent wraps for overflow and underflow cases. Special exponent biasing is used to allow fast underflow detection and to free-up one exponent cycle for the rounding pre-computations (Section IV).

With these modifications, the fully rounded FP result is available after 5.5 cycles in an internal representation. Special cases like Infinity and Zero are indicated using special flags. The result and flags are forwarded to the local and remote FPU and muxed directly into the operand latches, bypassing any operand formatting. The special value flags are used to bypass the special value detection logic in the format step. At the end of the pipeline a packing circuit reformats the data from the internal FPU format to the architected register file data format. It also muxes in the integer results and potential constants into the result register. From there, the properly formatted result is distributed via a forwarding network to all subunits of the VSU, including the FPUs and the register file. Thus, the FPUs themselves can use a FP result after 6 cycles, whereas FX results produced by the FPU can only be forwarded after 7 cycles.

IV. Exponent Path

The exponent path computes the shift amounts for the mantissa data path, overflow and underflow conditions, and the result exponent. The most critical paths of the exponent logic run through the underflow checks and the computation

of the normalization shift amount. This section shows how a new internal exponent representation simplifies and speeds-up these two timing critical paths.

A. Conventional Exponent Scheme

The calculation of the final exponent of an FMA instruction A×C+B includes several steps: First, the product exponent is calculated as $e_p = e_a + e_c$. If B is much larger than the product P, then $e_s = e_b$ (case 1). If B is larger than P but their fractions overlap such that $e_p + 57 > e_b > e_p + 3$ (case 2) then the resulting exponent of the sum is $e_s = e_p + 57$. Otherwise $e_s = e_p + 3$ (case 3). Then, the Leading Zero Count LZC of the sum is computed. For cases 1 and 2, it is derived from the aligner shift amount and the leading zeros of operand B. For case 3, it is provided by a leading zero anticipator operating on the trailing 110 bits of the sum.

Depending on the underflow mask, when the result exponent e_r shall not drop below $e_{\min}$, the normalization shift-amount $nsha$ must be limited to $nsha = \min(LZC, e_s - e_{\min})$. The underflow is then detected when $LZC > e_s - e_{\min}$ and the exponent after normalization is $e_n = e_s - nsha$. If the mantissa rounding creates a carry-out (c_{out}), e_r must be incremented, i.e., $e_r = e_n + c_{\text{out}}$. Comparing e_r against $e_{\max}$ permits to detect an overflow.

Since the exponent of intermediate results of an FMA operation can be much larger than the final exponent, a wider intermediate format is needed, as seen in Fig. 4-left. The POWER6 FPU handles the conventional 13-bit exponent representation using $\text{bias}_{13b} = 4095_{10}$ making all intermediate exponents positive. The value e of the biased exponent E written in memory format can be calculated as $e = E - \text{bias}$, where the bias for SP and DP data are 127_{10} and 1023_{10} respectively. In order to check, whether an internal exponent underflows or overflows, the exponent is compared against the boundaries $E'\text{min}$ and $E'\text{max}$ of the target precision.

$$E'\text{min}_{SP|DP} = e_{\text{min}SP|DP} + \text{bias}_{13b} = 3969_{10}|3073_{10}$$
$$E'\text{max}_{SP|DP} = e_{\text{max}SP|DP} + \text{bias}_{13b} = 4222_{10}|5118_{10}$$

This scheme requires 13-bit comparators to check each intermediate value for overflow and underflow conditions.

B. Improved Exponent Scheme

Instead of the 13-bit format with 13-bit bias, we use a 13-bit biased two's complement format, rebiasing exponents with the minimum exponent $e_{\text{min}T}$ of the target precision, i.e., $e_{\text{min}SP} = -126_{10}$ for SP results and $e_{\text{min}DP} = -1022_{10}$ for DP results. since operands and result can have different precisions, they are rebiased as follows:

$$E'_{SP} = e_{SP|DP} - e_{\text{min}T} = E_{SP|DP} - \text{bias}_{SP|DP} - e_{\text{min}T}$$

The exponent boundaries for SP and DP targets become:

$$E'\text{min}_{SP|DP} = e_{\text{min}SP|DP} - e_{\text{min}T} = 0|0$$
$$E'\text{max}_{SP|DP} = e_{\text{max}SP|DP} - e_{\text{min}T} = 253_{10}|2045_{10}$$

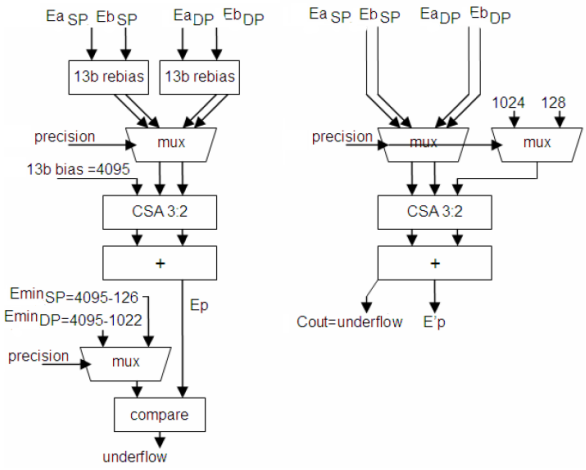

Figure 4. Computing product exponent and checking for underflow with the conventional re-biasing (left) and the new re-biasing scheme (right)

The value $e_{\min}$ is mapped to 0 for all target precisions, so that intermediate exponents smaller than $e_{\min}$ are negative. Thus, the underflow check becomes a simple inspection of the sign bit of the exponent, saving the 13-bit comparator of the conventional scheme.

$$e_p < e_{\text{min}T} \quad \leftrightarrow \quad E'_p = e_p - e_{\text{min}T} < 0$$

Given the rebiasing, $E'_s = e_s - e_{\min}$, thus limiting $nsha$ becomes $nsha = \min(LZC, E'_s)$. The right part of Fig. 4 shows the computation of the product exponent, checking it for underflow, and computing the limit $E'_p - E'\text{min}$. The new rebiasing scheme allows removing several 13-bit comparators in the exponent logic, especially on the underflow check. Hence, it results in a faster and more area-efficient exponent logic.

V. Compact Floorplan

In a high frequency design, a signal can barely cross an FPU on default wire using M1 to M5 [6]. Thus, a compact floorplan with short and efficient wiring on all major buses is required to close timing. For an FPU, a good floorplan needs to match the structure of Fig. 3 well. The operands reach the FPU via the operand latches and go in parallel through aligner and multiplier into the adder. The 160-bit wide sum is processed by normalizer and rounder feeding into the result latch. Multiplier and aligner are equally timing critical and therefore should be very close to operand latches and adder. For fast result forwarding within the VSU, rounder and result latch should be near the top of the FPU.

The two-stack U-shaped floorplan from [1] (Fig. 5, top) fulfill these requirements. However there are three aspects which can be improved: First, long wires on the return path from the rounder to the result latch over the multiplier add extra delay to the result forwarding path and worsen the timing of the multiplier. Second, two 110-bit horizontal data

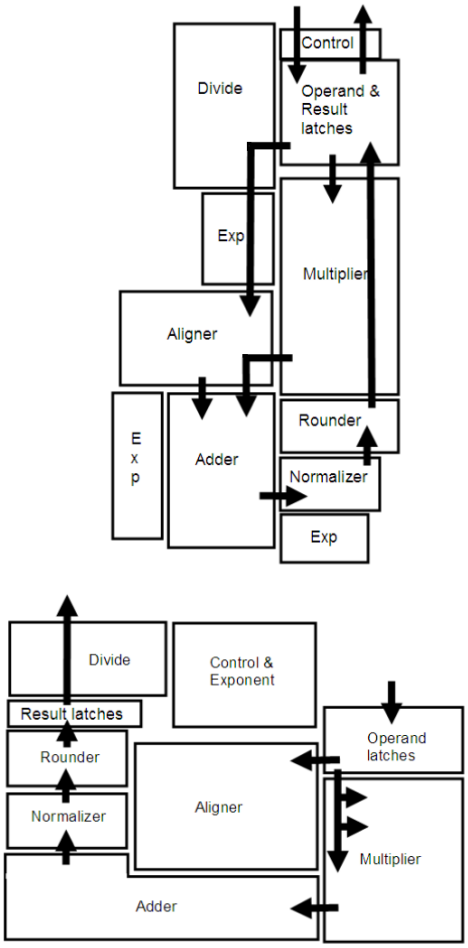

Figure 5. Floorplan of a two-stack and a three-stack U-shaped FPU

buses run over the aligner, producing horizontal wire congestion at unit level. Third, the exponent logic is distributed over the whole unit stressing the otherwise less timing critical exponent path.

Our new three-stack FPU floorplan (Fig. 5, bottom) keeps the advantages of the two-stack floorplan and overcomes its shortcomings. Multiplier and aligner are both placed between operand latches and adder, resulting in very short data buses. The third stack is used for the backend of the pipe, i.e., adder output, normalizer and rounder which can be connected buffer-free with very short wires. Even the result bus becomes shorter. The exponent, control, and divide logic are placed outside of the main data-flow to reduce wire congestion. The exponent is a single macro which allows for very short internal wires and fast connections to operand latches, aligner, normalizer, and rounder. This results in a very compact and virtually buffer free floorplan with no wire congestion, in which all major buses are short, allowing detuning the devices within the macros, improving timing, area and power.

VI. Summary

The POWER7 FPU is the first implementation of the new VSX extension, providing vector DP capability. The scalar FPU is embedded into the VSU with its four-way SIMD SP and two-way SIMD DP. That allows for efficient mixing of vector and scalar FP code, and doubles the architected registers available to both FPU types. The implementation merges the vector and scalar FPU into a single unit, instantiated four times per core in a very area and power efficient design. Compared to the POWER6 core, the POWER7 FPUs can perform twice as many DP operations per cycle in a 2.5 times smaller area on top of the normal technology shrink. Merging the vector and scalar FPUs accounts for a factor 1.35. Streamlining the 64-bit FPU pipeline and reducing the frequency account for the remaining factor of 1.8. Its compact 3-stack floorplan requires virtually no buffers, shortens all critical buses and allows detuning of devices. Removing the 64-bit FX multiplier resulted in more natural partitioning of the pipe stages and significantly fewer registers. A different exponent rebiasing scheme allows for a smaller and less timing critical exponent logic. Precomputing rounding information throughout the pipeline is a key for achieving a half-cycle rounder and supporting 6-cycle result forwarding to both FPUs.

References

[1] L. Eisen *et al.*, “IBM POWER6 accelerators: VMX and DFU”, *IBM Jal. Research and Development*, Vol. 51, No. 6, pp. 663-683, Nov. 2007

[2] D. Bistry *et al.*, “The complete Guide to MMX Technology”, *McGraw-Hill/TAB Elec.*, ISBN 0-07-006192-0, Apr. 1997

[3] S. Oberman *et al.*, “AMD 3DNow! Technology and K6-2 Microprocessor”, *Proc. Hot Chips 10*, pp. 245-254, Aug. 1998

[4] “Power Instruction Set Architecture Ver. 2.06 Rev. B”, Jul. 2010, http://www.power.org/resources/downloads

[5] R. Kalla *et al.*, “POWER7: IBM’s Next-Generation Server Processor”, IEEE Micro, pp. 7-15, Mar. 2010

[6] D. Wendel *et al.*, “The implementation of POWER7: A Highly Parallel and Scalable Multi-Core High-End Server Processor”, *IEEE Intl. Solid-State Circ. Conf. Digest of Technical Papers*, pp. 102-103, Feb. 2010

[7] H. Le *et al.*, “IBM POWER6 microarchitecture”, *IBM Jal. Research and Dev.*, Vol. 51, No. 6, pp. 639-662, Nov. 2007

[8] S. Dao Trong *et al.*, ”P6 Binary Floating-Point Unit”, *Proc. 18th IEEE Symp. Comp. Arith.*, pp. 77-86, Jun. 2007

[9] B. Curran *et al.*, “4GHz+ Low-Latency Fixed-Point and Binary Floating-Point Execution Units for POWER6”, *Proc. IEEE Intl. Solid-State Circ. Conf.*, pp. 1728-1734, Feb. 2006

[10] IEEE Computer Society, “IEEE Standard for Floating-Point Arithmetic”, IEEE Standard 754-2008, Aug. 2008

[11] E. Schwarz *et al.*, “FPU Implementations with Denormalized Numbers”, *IEEE Trans. Computers*, Vol. 54, No. 7, pp. 825-836, Jul. 2005