7.4 A Streaming Processing Unit for a CELL Processor

B. Flachs[1], S. Asano[3], S.H. Dhong[1], P. Hofstee[1], G. Gervais[1], R. Kim[1], T. Le[1], P. Liu[1], J. Leenstra[2], J. Liberty[1], B. Michael[1], H. Oh[1], S. M. Mueller[2], O. Takahashi[1], A. Hatakeyama[4], Y. Watanabe[3], N. Yano[3]

[1]IBM, Austin, TX
[2]IBM, Boeblingen, Germany
[3]Toshiba, Austin, TX
[4]Sony, Austin, Texas

The Synergistic Processor Element (SPE) is the first implementation of a new processor architecture designed to accelerate media and streaming workloads. Area and power efficiency are important enablers for multi-core designs that take advantage of parallelism in applications [2]. The architecture reduces area and power by solving scheduling problems such as data fetch and branch prediction in software. SPE provides an isolated execution mode that restricts access to certain resources to validated programs.

The focus on efficiency comes at the cost of multi-user operating system support. SPE load and store instructions are performed within a local address space, not in system address space. The local address space is untranslated, unguarded and non-coherent with respect to the system address space and is serviced by the local store (LS). Loads, stores and instruction fetch complete without exception, greatly simplifying the core design. The LS is a fully pipelined, single-ported, 256kb SRAM [3] that supports quadword (16B) or line (128B) access.

The SPE is a SIMD processor programmable in high level languages such as C or C++ with intrinsics. Most instructions process 128b operands, divided into four 32b words. The 128b operands are stored in a 128-entry-unified-register file used for integer, floating point and conditional operations. The large register file facilitates deep unrolling to fill execution pipelines. Figure 7.4.1 shows how the SPE is organized as well as the key bandwidths (per cycle) between units.

Instructions are fetched from the LS in 32 4B groups. Fetch groups are aligned to 64B boundaries to improve the effective instruction fetch bandwidth. 3.5 fetched lines are stored in the instruction line buffer (ILB) [1]. One-half line holds instructions while they are sequenced into the issue logic; as another line holds the single entry software managed branch target buffer (SMBTB) and two lines are used for inline prefetching. Efficient software manages branches in three ways: it replaces branches with bit-wise select instructions; it arranges for the common case to be inline; it inserts branch hint instructions to identify branches and load the probable targets into the SMBTB.

The SPE can issue up to two instructions per cycle to seven execution units organized in two execution pipelines. Instructions are issued in program order. Instruction fetch sends double word address-aligned instruction pairs to the issue logic. Instruction pairs are issued if the first instruction (from an even address) is routed to an even pipe unit and the second instruction to an odd pipe unit. Loads and stores wait in the issue stage for an available LS cycle. Issue control and distribution require three cycles.

Figure 7.4.5 details the eight execution units. Unit to pipeline assignment maximizes performance given the rigid issue rules. Simple fixed point [4], floating point [5] and load results are bypassed directly from the unit output to input operands reducing result latency. Other results are sent to the forward macro where they are distributed one cycle later. Figure 7.4.2 is a pipeline diagram for the SPE that shows how flush and fetch are related to other instruction processing. Although frequency is an important element of SPE performance, pipeline depth is similar to those found in 20 FO4 processors. Circuit design, efficient layout and logic simplification are the keys to supporting the 11 FO4 design frequency while constraining pipeline depth.

Operands are fetched either from the register file or forward network. The register file has six read ports, two write ports, 128 entries of 128b each and is accessed in two cycles. Register file data is sent directly to unit operand latches. Results produced by functional units are held in the forward macro until they are committed and available from the register file. These results are read from six forward macro read-ports and distributed to the units in one cycle.

Data is transferred to and from the LS in 1024b lines by the SPE DMA engine which allows software to schedule data transfers in parallel with core execution thereby overcoming memory latency and achieving high memory bandwidth. The SPE has separate 8b wide inbound and outbound data busses. The DMA engine supports transfers requested locally by the SPE through the SPE request queue or externally either via the external request queue or external bus requests through a window in the system address space. The SPE request queue supports up to 16 outstanding transfer requests. Each request can transfer up to 16kb of data to or from the local address space. DMA request addresses are translated by the memory management unit before the request is sent to the bus. Software can check or be notified when requests or groups of requests are completed.

The SPE programs the DMA engine through the channel interface which is a message passing interface intended to overlap I/O with data processing and minimize power consumed by synchronization. Channel facilities are accessed with three instructions: read channel, write channel, and read channel count which measures channel capacity. The SPE architecture supports up to 128 unidirectional channels which are configured as blocking or non-blocking.

Figure 7.4.3 is a photo of the 2.5×5.81mm^2 SPE. Figure 7.4.4 is a voltage versus frequency shmoo that shows SPE active power and die temperature while running a single precision intensive lighting and transformation workload that averages 1.4 IPC. This is a computationally intensive application that has been unrolled four times and software pipelined to eliminate most instruction dependencies and utilizes about 16kb of LS. The relatively short instruction latency is important. If execution pipelines are deeper, this algorithm requires further unrolling to hide the extra latency. More unrolling requires more than 128 registers and thus is impractical. Limiting pipeline depth also helps minimize power; the shmoo shows SPE dissipates 1W at 2GHz, 2W at 3GHz and 4W of active power at 4GHz. Although the shmoo indicates operation to 5.2GHz, separate experiments show that at 1.4V and 56°C, the SPE achieves to 5.6GHz.

Acknowledgements:
S. Gupta, B. Hoang, N. Criscolo, M. Pham, D. Terry, M. King, E. Rushing, B. Minor, L. Van Grinsven, R. Putney, and the rest of the SPE design team.

References:
[1] O. Takahashi et al., "The Circuits and Physical Design of the Streaming Processor of a CELL Processor," submitted to Symp. VLSI Circuits, June, 2005.
[2] D. Pham et al., "The Design and Implementation of a First-Generation CELL Processor," *ISSCC Dig. Tech. Papers*, Paper 10.2, pp. 184-185, Feb., 2005.
[3] T. Asano et al., "A 4.8GHz Fully Pipelined Embedded SRAM in the Streaming Processing of a CELL Processor," *ISSCC Dig. Tech. Papers*, Paper 26.7, pp. 486-487, Feb., 2005.
[4] J. Leenstra et al.," The Vector Fixed Point Unit of the Streaming Processor of a CELL Processor," submitted to Symp. VLSI Circuits, June, 2005.
[5] H. Oh et al., "A Fully-Pipelined Single-Precision Floating Point Unit in the Streaming Processing Unit of a CELL Processor," submitted to Symp. VLSI Circuits, June, 2005.

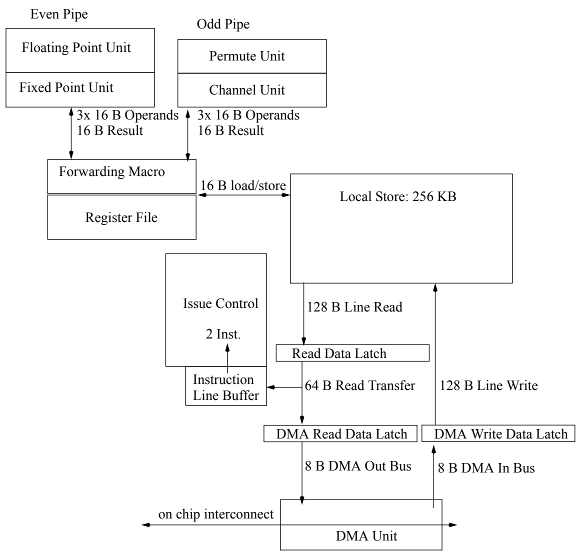


Figure 7.4.1: SPE organization.

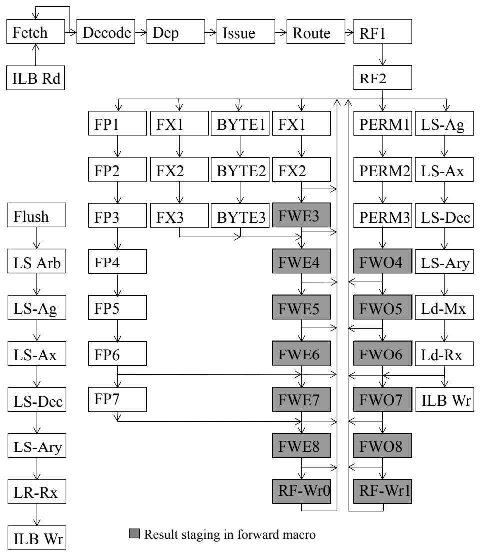


Figure 7.4.2: SPE pipeline diagram.

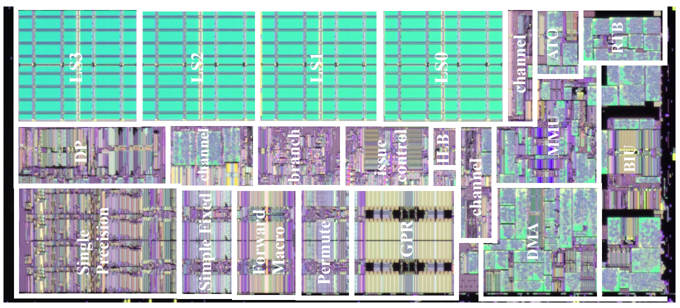
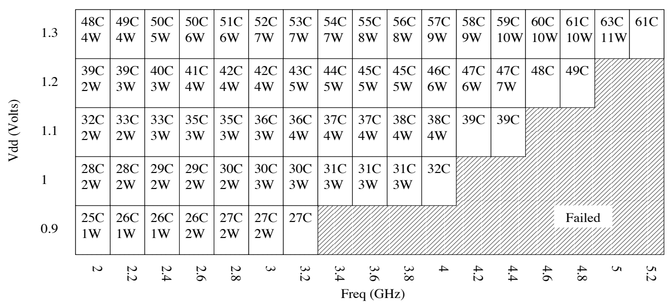


Figure 7.4.3: SPE die micrograph.



Vdd (Volts) \ Freq (GHz)	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6	3.8	4	4.2	4.4	4.6	4.8	5	5.2
1.3	48C 4W	49C 4W	50C 5W	50C 6W	51C 6W	52C 7W	53C 7W	54C 7W	55C 8W	56C 8W	57C 9W	58C 9W	59C 10W	60C 10W	61C 10W	63C 11W	61C
1.2	39C 2W	39C 3W	40C 3W	41C 4W	42C 4W	42C 4W	43C 5W	44C 5W	45C 5W	45C 5W	46C 6W	47C 6W	47C 7W	48C	49C	▨	▨
1.1	32C 2W	33C 2W	33C 3W	35C 3W	35C 3W	36C 3W	36C 4W	37C 4W	37C 4W	38C 4W	38C 4W	39C	39C	▨	▨	▨	▨
1	28C 2W	28C 2W	29C 2W	29C 2W	30C 2W	30C 3W	30C 3W	31C 3W	31C 3W	31C 3W	32C	▨	▨	▨	▨	▨	▨
0.9	25C 1W	26C 1W	26C 1W	26C 2W	27C 2W	27C 2W	27C	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Figure 7.4.4: Voltage/frequency schmoo.

Unit	Instructions	Execution Pipe	Unit Pipeline Depth	Instruction Latency
Simple Fixed	word arithmetic, logicals, count leading zeros, selects, and compares	Even	2	2
Simple Fixed	word shifts and rotates	Even	3	4
Single Precision	multiply-accumulate	Even	6	6
Single Precision	integer multiply-accumulate	Even	7	7
Byte	pop count, absolute sum of differences, byte average, byte sum	Even	3	4
Permute	Quadword shifts, rotates, gathers, shuffles as well as reciprocal estimate	Odd	3	4
Local Store	Load and store	Odd	6	6
Channel	Channel Read/Write	Odd	5	6
Branch	Branches	Odd	3	4

Figure 7.4.5: Unit and instruction latency.