The Vector Fixed Point Unit of the Synergistic Processor Element of the Cell Architecture Processor

N. Mäding, J. Leenstra, J. Pille, R. Sautter, S. Büttner, S. Ehrenreich and W. Haller
IBM Entwicklung GmbH, Boeblingen, Germany
nmaeding, leenstra, pillej, r_sautter, buettnes, sehrenreich, haller@de.ibm.com

Abstract

A Vector Fixed Point Unit (FXU) is designed to speed up multi-media processing. The FXU implements SIMD style integer arithmetic and permute operations. The adder, rotator and permute structure enables the use of static circuits only. The FXU was fabricated using IBM 90nm CMOS SOI technology.

1. Introduction

High degrees of parallelism are achievable for streaming data intensive processing tasks with high frequency Single Instruction Multiple Data (SIMD) style execution units. For the Synergistic Processor Element (SPE) of the Cell Processor [1] a FXU is designed to combine high frequency with a short pipeline latency and low area/power.

A short FXU pipeline is of crucial importance to prevent performance degradation by running out of registers in heavily unrolled loops. Techniques used to reach the goals include (1) new add/rotate/permute structures, (2) new instruction set architecture (ISA) targeted at the 11 FO4 design frequency, (3) floor plan balancing instruction execution delay with result distribution and (4) use of clock gated pulsed clock non-scannable latches with low area/power.

2. FXU Overview

The FXU organization is illustrated in Figure 1. The FXU enables dual instruction issue by having 2 subunits. One subunit is the ALU (with its Adder/Compare, Logical, Rotator and Count Leading Zero (CLZ) macros). The other includes Permute and Table-lookup macros. Both units operate on up to three 128-bit source vectors and produce a 128-bit vector result.

The 128b SIMD ALU operates in parallel on multiple of 8b, 16b or 32b wide elements according to the instruction executed. Data types are integer, single precision floating point for compare and true/false (all bits "1"/"0", to provide predicates for 'equal to' and 'greater than'). The permute unit implements the 128b wide inter-element manipulation instructions. For the permute instruction, the 3rd operand specifies how bytes of the other two source operand compose 16 result bytes. In addition, the unit handles 128b wide rotate/shifts and load/store alignment and floating point estimate instructions (by table lookup).

All operands are latched and implemented by scanable dynamic 5-way multiplexer (MUX) latches. The dynamic MUX selects between the result of the register file, forward macro, local store, FXU and floating point. The staging latches inside the FXU are clock pulsed latches that enable a 1FO4 cycle stealing using the latch transparency during the clock pulse. The FXU result is distributed in 0.5 cycle to all operand latches and the forward macro, leaving the other 0.5 cycle (i.e. 5-6 FO4) for logic. Operand latches are duplicated for enabling fine granular dynamic power saving and to balance unit logic delay with result distribution delay based on the floor plan and macro input cap load.

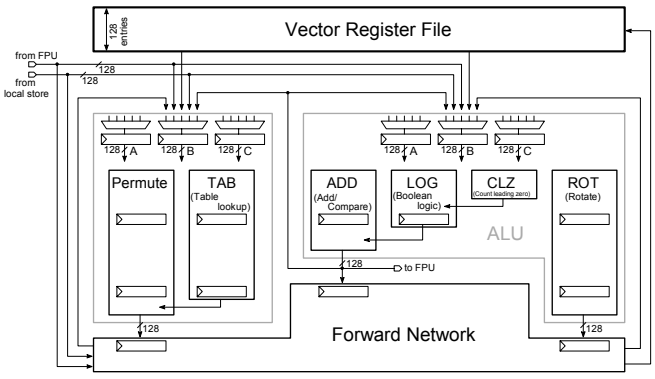

Figure 1: FXU organization

2.1 ADD macro

The total path delay of the Adder/Compare macro is 1.5 cycle. Add and compares are done in the 1st cycle. In

the 2nd cycle the results of the Logical and CLZ macros are selected or the all “1”/”0” true/false mask for compares is generated before result distribution.

A static adder is preferred over a dynamic adder for lower power consumption and better circuit stability with regard to noise and coupling immunity. Figure 2 shows, the adder implementation using a carry select structure with 8-bit wide adder blocks. The carry logic is shared between sum and compare logic. The compare logic handles 8b, 16b and 32b signed and unsigned integers and single precision floating-point numbers.

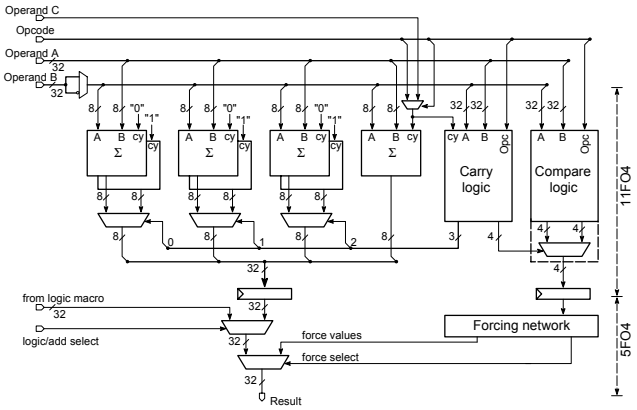

Figure 2: ADD macro block diagram

The critical paths are the carry logic outputs for sum select. A transmission gate multiplexer is used for the sum selection to reduce the carry output load.

A good overview of various adder structures is given in [2]. The *carry logic* is constructed according the parallel prefix algorithm of Kogge and Stone to have cells with low fanout and highly uniform size. Figure 3 illustrates the simplification of a Kogge-Stone adder structure towards the carry network.

In previous static adder designs the byte generate is generally computed as:

$$G8_i = G_i + P_i \bullet G_{i+1} + P_i \bullet P_{i+1} \bullet G_{i+2} + P_i \bullet P_{i+1} \bullet P_{i+2} \bullet G_{i+3} + P_i \bullet P_{i+1} \bullet P_{i+2} \bullet P_{i+3} \bullet G_{i+4} + P_i \bullet P_{i+1} \bullet P_{i+2} \bullet P_{i+3} \bullet P_{i+4} \bullet G_{i+5} + P_i \bullet P_{i+1} \bullet P_{i+2} \bullet P_{i+3} \bullet P_{i+4} \bullet P_{i+5} \bullet G_{i+6} + P_i \bullet P_{i+1} \bullet P_{i+2} \bullet P_{i+3} \bullet P_{i+4} \bullet P_{i+5} \bullet P_{i+6} \bullet G_{i+7}.$$

The use of $G_i \bullet P_i = G_i$ can reduce the fet stack height in a dynamic adder [3]. We found that by using the redundancy $G_i \bullet P_i = G_i$ our static adder can be improved as well:

$$G8_i = ((G_i + G_{i+1}) + P_{i+1} \bullet P_{i+2} \bullet (G_{i+2} + G_{i+3})) \bullet P_i + ((G_{i+4} + G_{i+5}) + P_{i+5} \bullet P_{i+6} \bullet (G_{i+6} + G_{i+7})) \bullet P_i \bullet P_{i+1} \bullet P_{i+2} \bullet P_{i+3} \bullet P_{i+4}.$$

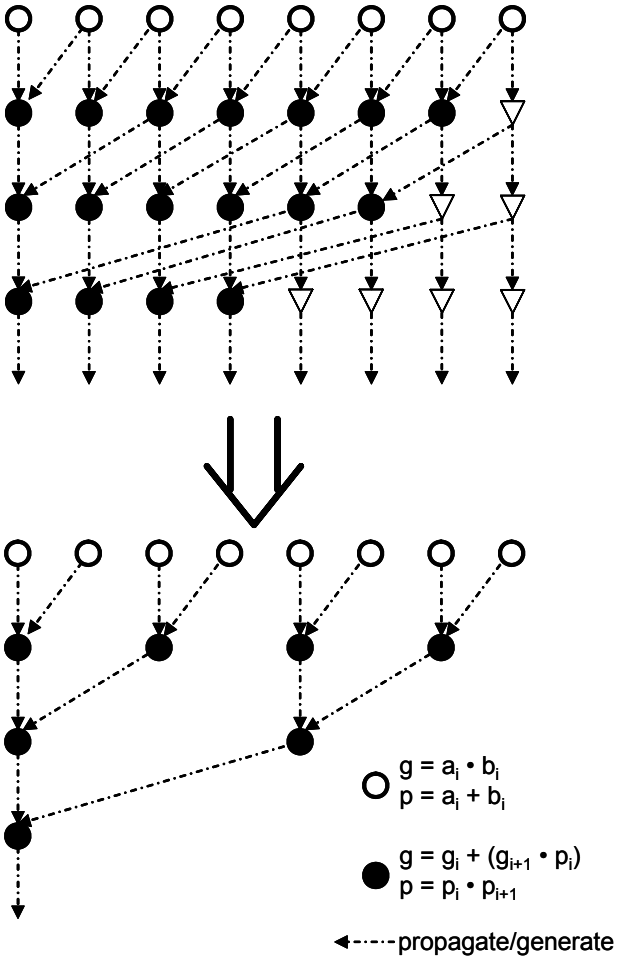

Figure 3: Kogge-Stone adder vs. carry network

This new equation allows the implementation of the first adder stages with (G_j+G_{j+1}), j=0,2,4,6 using the NAND2 circuit of Figure 4. Complemented inputs further balances the delay between propagate and generate signals.

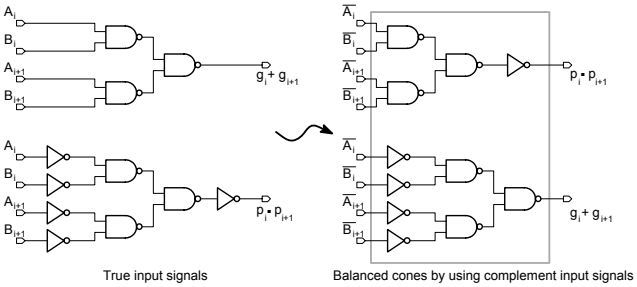

Figure 4: Balanced input stage

As the final optimization the carry block for eight bits of each input operand are structured as illustrated in Figure 5. By changing the propagate tree the timing becomes balanced. The remote propagate signal covers the wire distance while the non critical paths has additional logical effort. In the given technology this

approach has the best results in area, performance and power consumption.

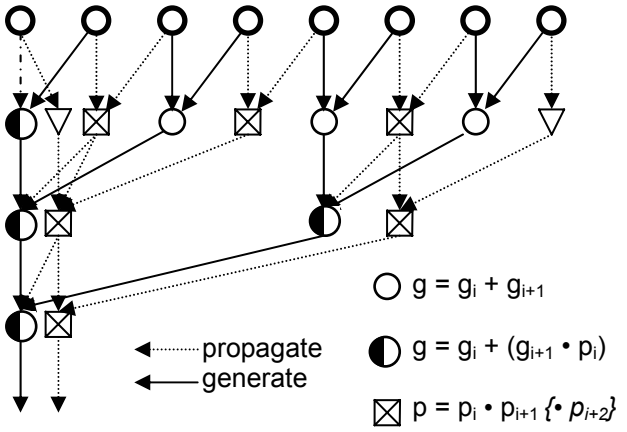

Figure 5: byte carry block

To enable SIMD operations of addition, subtraction, and compare, for various operand lengths, the carry network is exploited for additional functionality, as shown in Figure 6.

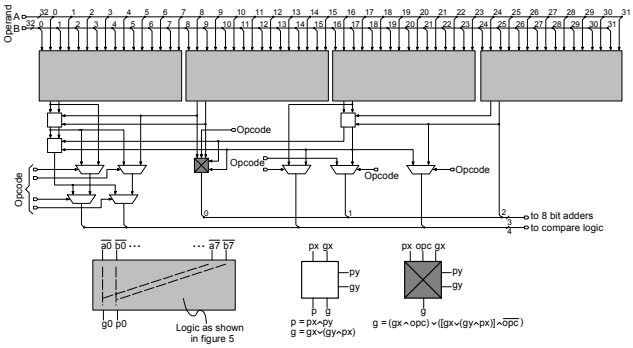

Figure 6: carry logic

The *compare logic* calculates both potential results. As for the multiplexers of the carry select adder a balanced timing between the carry network and the multiplexers at the output of the compare logic is crucial. The forcing network assures special results for the compare operations at the final result multiplexer.

The implementation of the carry in signals for the subtraction as well as some compare functions are not visualized to ensure clearness of the illustration.

2.2 ROT macro

State-of-the-art SIMD rotator macros to rotate/shift to the left or right using logarithmic rotator arrays are based on two different approaches.

They either have independent logarithmic rotator arrays for each of the supported data types and then select between the different results, or have a more complex rotator arrays structure, which takes the width of the different data types into account. In both cases the rotate array block is followed by a masking stage for the shifts. These approaches are discarded for the presented FXU, because they are neither area economic nor performance optimal.

Figure 7 shows the proposed organization of one out of four 32b rotator macros capable to rotate/shift left/right one 32b or two 16b (un)signed integers. The ISA helps a low latency rotator implementation by specifying the shift/rotate amounts controlling the rotate array multiplexer without conversion. Two 16b array structures are used instead of a traditional 32 bit rotator array to balance delay and to reduce wiring length. The following masking logic uses a 3-way MUX to generate the result. The three MUX select lines of each MUX are controlled independently. This enables to route/mask the 16b array results for each bit as need to combine the two 16b results into 32b. Each 3:8 decoder used for shift operations has an 8b output width and is duplicated to enable the two independent 16b shifts/rotates. Using this new structure, the control and dataflow path delay is balanced between all parts.

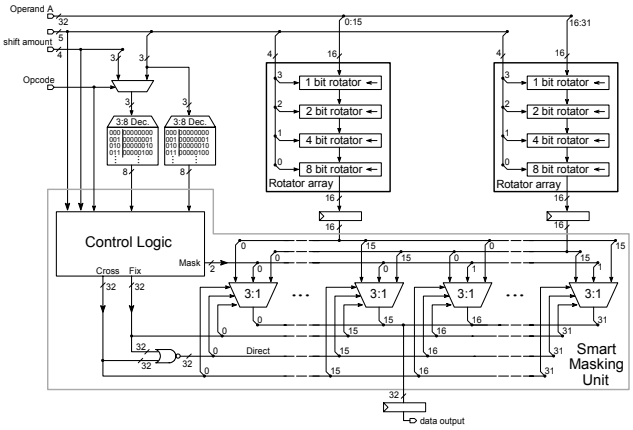

Figure 7: ROT macro block diagram

To enable a state-of-the-art logarithmic rotator array for SIMD operation additional multiplexer and wires are introduced into the data path. This makes the overall timing of the data signals heterogeneous and slower.

The proposed organization has simple logarithmic rotate arrays using 2-way multiplexer only. The timing of the data signals is homogeneous and the critical data path has not been afflicted in comparison to a non-SIMD 32b rotator/shifter. The presented method can easily be applied for 8b SIMD operation or even wider operands.

In case of the rotator the micro-architectural improvements were used to significantly reduce the area and power consumption as a trade off to a non critical performance impact.

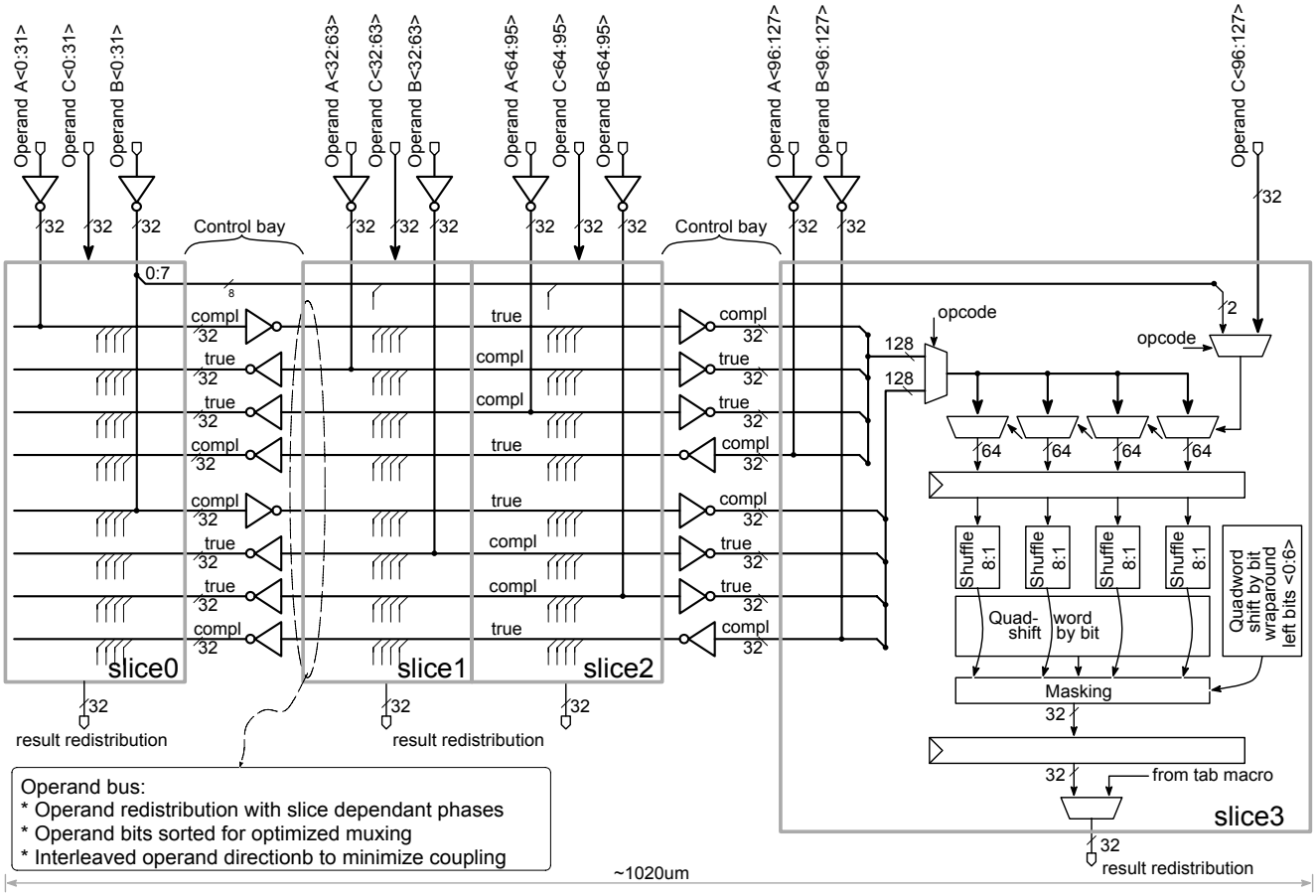

Figure 8: Permute macro structure

2.3 Permute

Figure 8 outlines the structure of the permute macro. It is capable to perform on 128b wide data three types of instructions: permutation, rotate/shift by bytes/bits.

The permute functionality implies 16 independent 32-to-1 byte multiplexer to implement the crossbar selecting the operand bytes for the 16 bytes of the 128b result. The same crossbar is used for rotate/shift implementation, using different MUX select settings. The shift/rotate amount is available only once in the 128b operand and hence the most timing critical. Therefore the crossbar was implemented such that a rotate is done with connecting the rotate amount directly to the MUX selects. The 3rd operand byte select permute values are locally converted in the first cycle into the corresponding rotate value to select each requested byte by the crossbar. The advantage of this approach is, that the crossbar is common for permute and byte shift/rotate with no timing impact. The first cycle also distributes the operands across the whole width of the 4 macros. To meet the slew requirement attention was devoted to the repowering and bit ordering. The wire direction is interleaved to minimize coupling, and takes the ordering of the multiplexer stages into account.

The boundary to the second cycle latches 4 times 256 bytes or 1K. This optimized wiring balances silicon and wiring area.

In the 2nd cycle contains the majority of the 32-to-1 byte reduction and the 128b shift by bit within a byte as well as the masking for shifts. To minimize the wires of the bit shifter the latch bank between the first two cycles is used to duplicate the bytes at the edges of the control bays and especially macro boundary. The 3rd cycle is only used to MUX the permute result with the table lookup and distribute the result to the forwarding macro.

Clock gating is very important in the permute macro due to the high latch count, which is in total only needed for the permute operation. To reduce the power consumption each latch is controlled instruction dependent.

3. Physical Implementation

The die photo shows in Figure 9 the two 128b subunits of the FXU separated by the *forward* macro and portions of the surrounding units. The upper subunit of the FXU contains the *Rotate*, *CLZ*, *Log* and *Add/Cmp* macros and has an overall size of 0.94 x 0.46 sqmm. The lower subunit contains the *Permute* und *Table-lookup* macros and utilizes 0.94 x 0.47 sqmm. Each macro is divided into four 32b wide blocks and two bays of control logic. The control logic is generated using logic synthesis and decodes locally the over 100 FXU instructions. All macros use only the lower 3 levels of metal for local wiring and other levels are used for unit and chip integration. The chip was fabricated using IBM 90nm CMOS SOI technology. All functional patterns run in the full application voltage and temperature range. The measured maximum frequency is 5.2GHz at 1.4V and 56°C. The SPE shmoo plot is included in [1].

4. Conclusion

A SIMD type processor element has been introduced with a static CMOS implementation targeted at 11 FO4. The FXU is composed of an Adder/Compare, Count Leading Zero, Rotator, Logical, Table-lookup and a Permute macro. The functional macros execute 16-way, 8-way and 4-way SIMD instructions.

With the presented structures a short FXU pipeline has been achieved. Power consumption was minimized by a instruction based clock gating. Only very few latches in the control bay are always active.

It is difficult to compare the presented solutions with state-of-the-art or academic research due to the technology implications. A product development for 11FO4 design frequency in 90nm CMOS SOI technology has not been done before.

The decision to use static circuit only supports the volume requirements of the product.

5. Acknowledgements

The authors thank M. Strohmer, U. Krauch, M. Pflanz, the complete SPE team, convergence team and management team for their support.

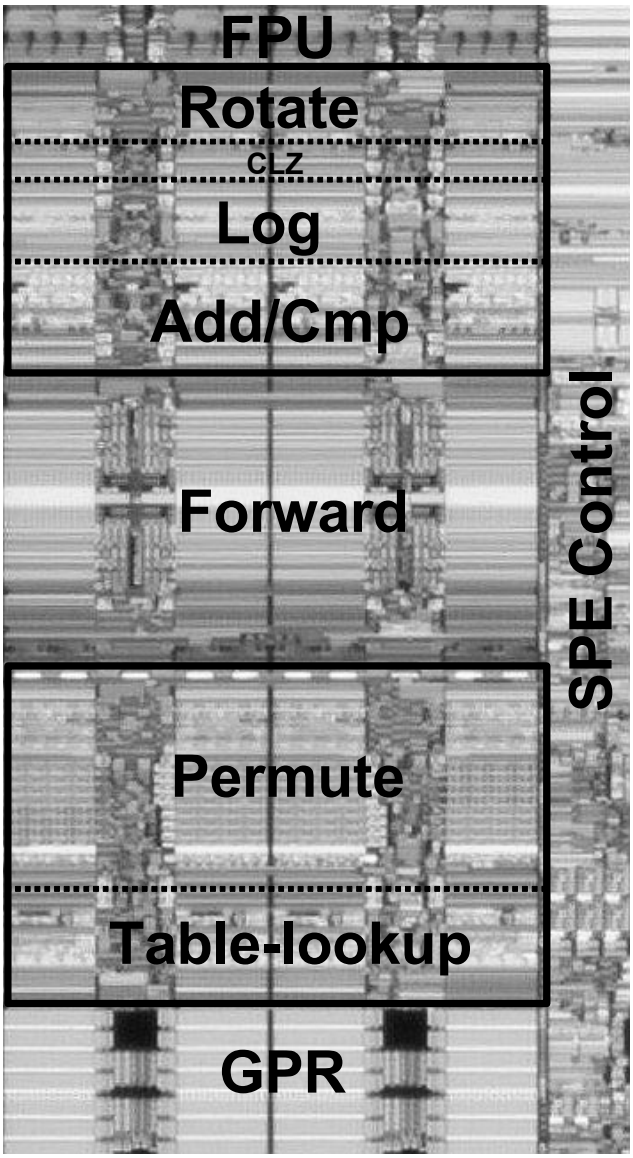

Figure 9: Die photograph

Reference

[1] B. Flachs et al., "The Microarchitecture of the Synergistic Processing Unit of Cell Architecture Processor," ISSCC 2005, pp. 134-135

[2] M. M. Ziegler et al., „A Unified Design Space for Regular Parallel Prefix Adders", DATE 2004

[3] J. Park et al., "470ps 64bit Parallel binary Adder", Symposium on VLSI Circuits, pp. 192-193, 2000