SIMD Multi Format Floating-Point Unit on the IBM z15(TM)

Stefan Payer
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
spayer@de.ibm.com

Cedric Lichtenau
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
lichtenau@de.ibm.com

Michael Klein
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
michael_klein@de.ibm.com

Kerstin Schelm
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
SCHELMKN@de.ibm.com

Petra Leber
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
PLEBER@de.ibm.com

Nicol Hofmann
Compute Unit Development
IBM Deutschland R&D GmbH
Böblingen, Germany
NHOFMANN@de.ibm.com

Tina Babinsky
Compute Unit Verification
IBM Deutschland R&D GmbH
Böblingen, Germany
tina.babinsky@de.ibm.com

***Abstract*— The IBM z Systems(TM) is the backbone of the insurance, banking, and retail industry. Innovation in these markets is driving the demand for new and additional applications to better serve the customers. These workloads like machine learning, data analytics, AI, etc. require a rapidly increasing number of computations in smaller precision formats. With IBM z15(TM) we completely redesigned the binary and hexadecimal floating-point unit to efficiently implement SIMD operations at 5.2GHz while maintaining the industry leading reliability, availability and serviceability standard. This paper describes the new design and special techniques used to achieve these goals like reusing the existing double precision unit pipeline for lower precision parallel SIMD, new approaches to formally verify the design, and improving error detection for the 14nm technology node.**

Keywords*— *Floating Point Unit, SIMD, Multi Format, Formal Verification, Binary, Hexadecimal, Machine Learning, AI, IBM z15(TM)

I. Introduction

Traditional workloads on the IBM z Systems rely on double and high floating-point precision and the z15(TM) machine keep delivering excellent performance in that area. With the rise of the second AI era, additional workloads in the area of data analytics and insights are deployed on z Systems to better handle tasks like fraud detection or customer interaction as well as system optimization. The new AI algorithms and models used to reach a better prediction accuracy are significantly more compute intensive; and data scientists have recourse to lower precision arithmetic to leverage SIMD and execute more computations in parallel per cycles. IBM z15(TM) is acknowledging this trend and provides a completely redesigned binary floating-point unit to meet these new workloads' requirements.

The Vector and Floating-Point Unit (VFU) is the main execution engine of the IBM z15(TM) processor shipped in September 2019 [1]. Manufactured in IBM's 14 nm technology, the VFU supports a core frequency of 5.2 GHz (same as z14, but more cores) [2] and supports 2-way simultaneous multi-threading. The design point of the VFU was enhanced to accelerate workloads for data analytics, machine learning, AI, which includes an ever-increasing amount of computations in smaller precision formats.

The execution engines in the VFU comprise two symmetrical pipes, where each pipe contains two binary floating point units (BFU), one Decimal and Quad Precision Engine (DQE), a divide and square root engine, a short latency decimal fixed-point engine, and a vector fixed-point and string engine which support IBM's Single Instruction Multiple Data (SIMD) architecture for IBM z Systems™ [3].

There are also two load/store pipelines that can read or write to a Vector Register File (VRF) and two FXU read and write ports that can access the VRF. [4]

Each VFU pipeline is a total of 10 cycles deep. The depth of the pipeline was determined by the longest pipeline depth, which is the Decimal and Quad-Precision Engine (DQE). When operations complete in shorter pipelines, the result are sent to a forwarding network where they can be sourced by a dependent operation as soon as the data is available; while freeing up associated resources to increase performance. [4]

The new BFU design doubles the throughput for single precision floating point operation and reduces the latency by 14% compared to the IBM z14(TM). Section 2 gives an overview of the binary and hexadecimal floating-point unit and its pipeline. Section 3 details the design decisions and implementation to support the high frequency and throughput. Section 4 discuss the approach to error detection in the

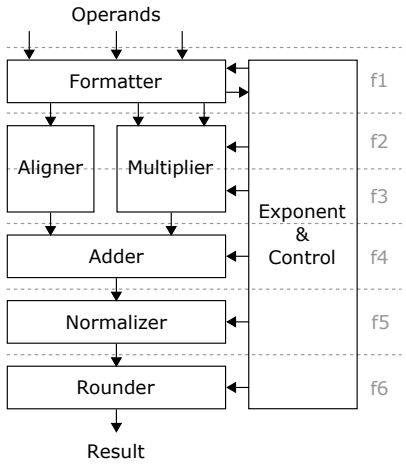

Fig. 1 BFU Pipeline Structure

arithmetic circuits. Finally Section 5 presents the testing and coverage of the more than 160 arithmetic instructions implemented in the unit.

II. Binary Floating-Point Unit (BFU)

The BFU pipeline was designed to be seven cycles back to back latency in prior zSystem design. It is designed to cover binary floating point and hexadecimal floating-point single and double precision 32b/64b data type operations. The newly designed BFU in z15 does have a reduced pipeline, which is only 6 cycles long, back to back. This reduces the latency by 14% for all instructions covered by the BFU.

Fig. 1 shows a block diagram of the new BFU pipeline. Since it is designed for a system frequency of 5.2 GHz, some blocks span over two cycles to meet timing requirements. The BFU is a six -stage fully pipelined unit to execute one instruction every cycle. Each operand bus feeding into the BFU, as well as the result bus is 64b wide.

A. Formatter

The Formatter is separating the sign, exponent and fraction from the operands and aligning the exponents. It sends the fraction to the aligner and multiplier as well as sign and exponent to the Exponent & Control. The detection of special cases like zero, infinity and nan; as well as the masking based on the data format is also done in the Formatter.

B. Aligner

In parallel to the Multiplier, the Aligner does align one of the fractions according to the exponents for the adder to match the result of the Multiplier. The Aligner covers two cycles and allows for shifts by up to 255 bits. A Barrel-Shifter was chosen for timing and wireability reasons.

C. Multiplier

The Multiplier does do the radix-4 based multiplication of two operands fraction, which can be up to 56 bits wide, each, including the implicit bit. That width was chosen to support the 14 fraction digits of the hex floating-point format. It is an adder tree to sum up the 29 partial products based on booth digit calculation as well as other correction terms. It returns 116 bits sum and carry as a result.

D. Adder

The Adder adds sum and carry, as well as the shifted result from the Aligner. A carry-lookahead adder structure was chosen. In parallel, a leading zero anticipator (LZA) is estimating the number of leading zeros to be normalized.

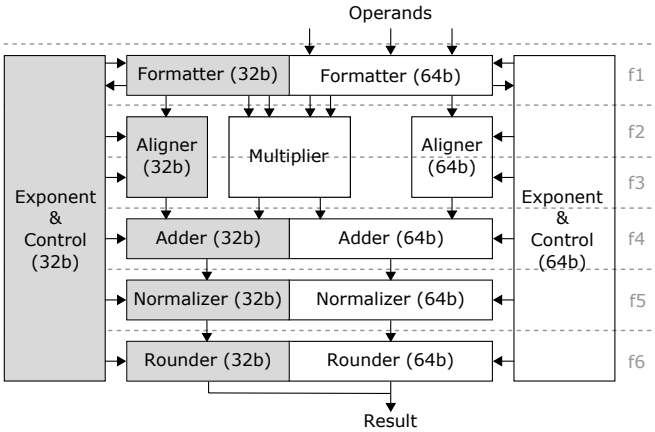

Fig. 2 BFU Pipeline with the extension to double the 32b throughput

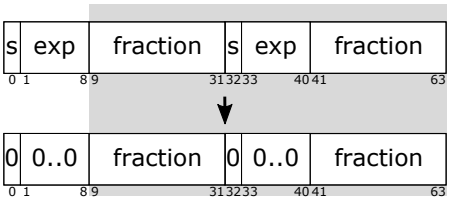

Fig. 3 Mapping of two 32b operands at one 64b multiplier input

E. Normalizer

The Normalizer is shifting the result of the Adder according the anticipated leading zeros.

F. Rounder

The Rounder is doing the correction of the case the leading zero anticipation was wrong by one, increments in rounding case, forces to maximum positive and negative values.

G. Exponent & Control

This block handles the calculation of the exponent and sets the control bits for all base blocks along the pipeline.

The reduced pipeline could be achieved by several different aspects. Since the initial design point, some generations ago, multiple technology enhancements were giving margin on the timing. Arithmetic improvements, e.g. faster shift amount calculation in the Aligner by using better pre-processing for the least significant bits of the shift amount, improving the adder structure from a manually written one with a latch boundary in it to removing the latch boundary and an optimized synthesized adder structure, streamlining the special case handling, and combining leading zero correction and rounding in one step. The previous BFU design was prepared for multi cycle ops like square root and divide. They were moved out, some of them generations ago, also published in [4]. By removing the feedback paths and optimizing for a pure pipelined design, the critical path could be shortened. The physical design was optimized for a small block approach. By moving to large synthesis blocks more improvements could be achieved, for example by optimizing the latch boundaries.

III. Detailing the SIMD Floating Point Implementation

The BFU in z15 was enhanced to double the throughput for 32b binary floating-point. As power and area are a critical resource in modern microprocessors, we chose an approach to share the existing multiplier and add smaller 32b variants of all other blocks.

Fig. 2 shows all new hardware blocks added to support the doubled throughput in grey. The Multiplier is reused, therefore both 64b operands are fed into the Multiplier, but the sign and exponent are forced to 0. All white blocks were existing and support the 64b wide data flow already.

The Adder (64b) is supporting the 64x64 bit mode and is about 116 bit wide. The Adder (32b) does only support the smaller precision 32x32 bit and is therefore 56b wide.

Fig. 3 shows how two 32b operands per one 64b input, α_0, α_1, β_0 and β_1, are having their exponent and sign forced to zero at the input of the multiplier. If both input operands of the multiplier are treated that way, the result P will look like:

$$P = (\alpha_0 * 2^{32} + \alpha_1) * (\beta_0 * 2^{32} + \beta_1) \quad (1)$$

$$P = \alpha_0 * \beta_0 * 2^{64} + \alpha_0 * \beta_1 * 2^{32} + \alpha_1 * \beta_0 * 2^{32} + \alpha_1 * \beta_1 \quad (2)$$

The result would consist of four parts added to each other. The first part, $a0*b0*264$, and the fourth part, $a1*b1$, are the parts needed for the result of two multiplications of 32b operands. If we suppress part two and three of the sum, there is no interference, and one can just pick the two result fractions at the proper bit position from the result bus. Therefore, a suppression mode was introduced, and each partial product of the Multiplier is divided into a high and low part. For each, the high and low part can be suppressed separately.

We carefully weighted what to pick as base for the mixed arithmetic precision design: the z14 existing double precision arithmetic engine and extend it to support two single precision path [6] or coupling two single precision units to support double precision. Our decision was mainly guided by:

- The existing double precision arithmetic design has been tuned over the years for area, power and timing efficiency; and has a running verification environment.
- Double precision arithmetic – binary but especially hexadecimal – is the most commonly used format in existing workload for the machine.
- IBM z Systems(TM) support subnormal handling in hardware for all precisions. A lot of extra wiring and logic was expected when starting from a single precision design.
- Trimming down double precision logic to single precision logic for elements of the design we duplicated in order to balance the design complexity vs. power/area gain can be done easily. In a number of cases it was down to changing generics/constants in the design and verification environment.

Fig. 4 shows the structure of the 64b Multiplier and its partial products, how they are aligned, and which parts get masked to have no interference of both multiplications of FP32 numbers in parallel.

In the new IBM z15(TM) microprocessor several instructions support that double bandwidth throughput. Those instructions are:

- Vector Load FP Integer
- Vector FP Multiply And Add / Subtract
- Vector FP Add / Subtract / Multiply
- Vector FP Negative Multiply And Add / Subtract
- Vector FP Load Lengthened / Rounded

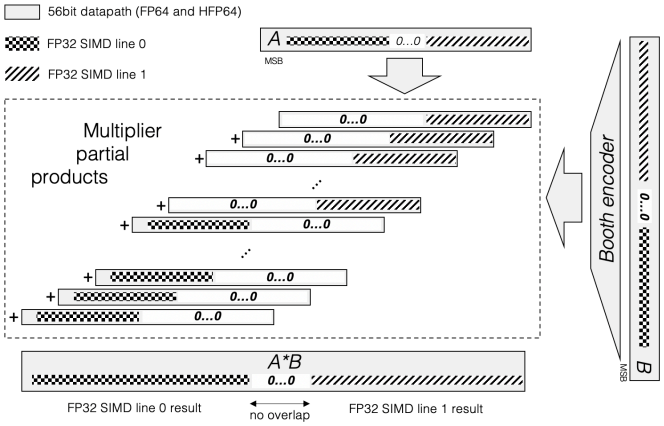

Fig. 4 Reusage of 64b Multiplier double throughput of 32b

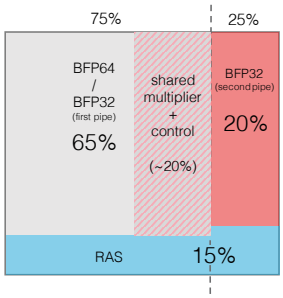

Fig. 5 Area spending for 32b SIMD and RAS of the BFU

- Vector FP Convert To / From Fixed / Logical

Fig. 5 shows the overall area spending of the BFU SIMD Implementation. The partitioning between function and RAS is 70% area for functionality and 30% for RAS protection logic. The base functionality of the 64b and 32b Binary Floating-Point function is 75% of the area. A 25% additional logic area is needed for the 32b SIMD Binary Floating-Point functionality (both including RAS). So, the overall area is 55% base functionality for 64b and 32b Binary Floating-Point, 15% for the additional 32b SIMD Binary Floating-Point capability and a 30% to protect both for RAS requirements.

IV. Reliability, Availability and Serviceability (RAS)

In the z15 BFU, the way of protecting the logic with residue has changed to a new scheme. The new scheme is easier maintainable and provides more fine granular checking. Each base block can be checked isolated and in case of changing one base block, only the local checking of that base block needs to be changed. The prior generations have been using a monolithic checking of the whole operation. The monolithic approach had many cases where the checking was turned off. In those cases, the checking was either too complicated or not feasible at all.

Fig. 6 shows the residue generation blocks for the main data flow and how they are placed between the base blocks. There are additional residue generation blocks for the Exponent & Control, which are not shown in the figure.

There is a residue generation block (gen_res) before and after each base block. Each base block does have a corresponding residue checking block, which protects the corresponding base block.

V. Verification Technique

The z15(TM) processor target market, especially banking and insurance sector requires a verified compliance to

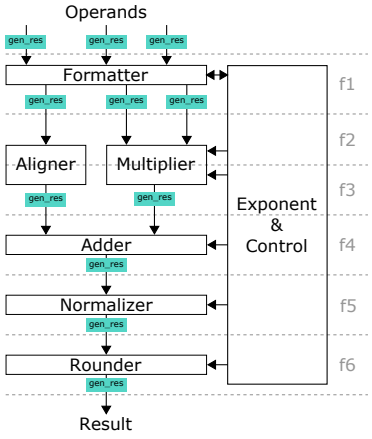

Fig. 6 Residue generation blocks as they are spread in the BFU

arithmetic standards for all computation. Exhaustive pre-silicon cycle-by-cycle simulation of the behaviour of the engine for all the representable arithmetic number would take many years. To overcome this limitation a number of techniques are applied to the verification of the VFU engine. The main obvious method used to tackle the complete verification is formal verification. This is also what is being used for the arithmetic engines with in-house tools [7]. It does still come with a number of challenges as the logic contains large (56x56) multipliers and in some area like number conversion, cycling multiple times through the pipeline. This significantly expand the exploration space for the formal verification. We use hierarchical methods to reduce the overall problem size and verify e.g. the multiplier standalone and then use a set of abstracted rules for the multiplier to verify the whole BFU [11]. Nevertheless the formal verification of the unit with its many hundreds of instructions can take months until the complete proof is achieved. This is not the desired turnaround time for designer modifying the logic and requiring fast validation response time. Partial formal test exploration is possible within hours but it does not cover well corner cases. To enable better and faster corner case testing, we rely on mathematically skewed random tests produced by an IBM tool FPGen [8][9] probing special arithmetic cases and operand alignments for floating points and rounding cases. These cases are based on the general IEEE arithmetic standard [13] as well as micro-architecture knowledge of the BFU itself to cover special cases handled by dedicated logic or transition between different paths/cases in the hardware, like overlapping/non-overlapping of the product and addend of an FMA operation.

Beyond the verification of mathematical operation, we also need to consider that more than 50 instructions can be in flight at any given point in time in the VFU engine. Exact sequencing of all control signals for the various pipes and stages including multi-reentry ops and instruction flushing due to speculative execution become very challenging. An adder supporting partial output masking and shifting, carry in/outs and rounding mode for multiple precision and split in multiple pipeline stages can easily have close to 40 different control signals. This requires to spend also significant verification effort in random simulation with sequences of instructions due to the out-of-order, pipeline nature of the design to reach the high frequency, low latency and high throughput. We are relying there on the Genesys Test Generator [10] coupled with a constraint solver reusing some constrains created for the FPGen tool, but also adding core micro-architecture constraints like instruction dependencies, scheduling or flushing [11]. Coverage events are coded, tracked and analysed across all verification disciplines to guaranty that defined rules and generated tests cover all cases defined during design planning and design implementation. Tracking and orchestration of the complete verification is done via the IBM Verification Cockpit Platform [12].

On top of this generic verification and as stated in paragraph 3, several new instructions were added or modified to support the doubled throughput. All of those instructions did have a working reference model for their scalar version to prove the correct functionality. In the beginning of the project, we verified those instructions by formal proving that the instance 0 of the BFU does behave the same for their low part of the vector as the instance 1 does behave for the high part. In addition, we prove that the high part of instance 0 does behave the same as instance 1 for the low part.

If we define the inputs vectors of the BFU $a = (a_0, a_1)$; $b = (b_0, b_1)$; $c = (c_0, c_1)$ and the result: $r = (r_0, r_1)$, and the input and result vector of BFU instance i to be a^i, b^i, c^i and r^i. Having two BFU instances, if $a_0^0 = a_1^1$, $b_0^0 = b_1^1$, $c_0^0 = c_1^1$, this means $r_0^0 = r_1^1$, and if $a_1^0 = a_0^1$, $b_1^0 = b_0^1$, $c_1^0 = c_0^1$, this means $r_1^0 = r_0^1$, for all vector instructions with equal inputs.

This cross verification makes sure that the calculation is the same for each vector element, while only proving each scalar calculation for each instruction once against the reference model. This way, the logic designers could early debug the newly written vector capabilities before all reference models for all vector instructions are available.

VI. Conclusion

This paper describes the SIMD Multi Format Floating Point Unit hardware in the Vector and Floating-Point Unit of the z15(TM) processor and the new approach for the formal verification. The hardware was designed to maximize performance for increasing amounts of floating-point operations in smaller hexadecimal and binary number formats while maintaining frequency, compared to IBM z14(TM). The newly written binary floating-point unit consists of a six stages deep execution pipeline and supports 5.2GHz. An advanced bypass network provides early result forwarding to prevent performance loss on dependent operations. The total area of the VFU hardware, including the vector and floating-point register files is 1.2mm^2 in a 14nm technology node, which is a significant shrink, compared to 3.9mm^2 of IBM z13(TM) [4]. This reduced area compares to achieving more than one third area saving beyond the technology shrink.

Acknowledgements

Many thanks to our excellent technical leads Silvia Melitta Müller, Eric Schwarz and Kevin Shum as well as to Osher Yifrach and Revital Arieli for the verification work and also to our physical design team which made this possible!

References

[1] "IBM Unveils z15 With Industry-First Data Privacy Capabilities" https://newsroom.ibm.com/2019-09-12-IBM-Unveils-z15-With-Industry-First-Data-Privacy-Capabilities (Press release). IBM, 2019.

[2] "IBM z15 (8561) Technical Guide" http://www.redbooks.ibm.com/abstracts/sg248850.html?Open. IBM, 2019.

[3] E. Schwarz, "The IBM z13 SIMD Accelerators for Integer, String, and Floating-Point", 22nd Symposium on Computer Arithmetic, 2014.

[4] C. Lichtenau, S. Carlough, S. Mueller, "Quad Precision Floating Point on IBM z13(TM)", 23nd Symposium on Computer Arithmetic, 2015

[5] "z/Architecture Principle of Operation" https://www.ibm.com/support/pages/node/6019426. IBM, 2019.

[6] "Fused Multiply-Adder with Booth-Encoding" https://patents.google.com/patent/US20140095568 Patent US20140095568A1. IBM, 2013.

[7] "RuleBase SixthSense Tool" https://www.research.ibm.com/haifa/projects/verification/Formal_Methods-Home/index.shtml

[8] "IBM FPgen Floating Point Test Generator (FPgen)" http://researcher.watson.ibm.com/researcher/view_group.php?id=9517

[9] "IBM Floating-Point Test Generator" https://www.research.ibm.com/haifa/projects/verification/fpgen/

[10] "IBM Genesys Professional Edition Test Generator" https://www.research.ibm.com/haifa/projects/verification/genesys_pro/index.html

[11] "IBM Constraint Solver" https://www.research.ibm.com/haifa/dept/vst/csp.shtml

[12] "Verification Cockpit Platform" https://www.research.ibm.com/haifa/dept/vst/hvt_psvtva_cockpit.shtml

[13] American National Standards Institute and Institute of Electrical and Electronic Engineers. "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, 1985