GRFPU – High Performance IEEE-754 Floating-Point Unit

Edvin Catovic
Revised by: Jan Andersson
Aeroflex Gaisler, Kungsgatan 12, SE-411 19 Göteborg, Sweden
support@gaisler.com

1. Introduction

To improve floating-point performance of LEON-based systems, a new FPU called GRFPU has been developed at Aeroflex Gaisler, formerly Gaisler Research. With a peak performance of 250 MFLOPS on a typical 0.13 µm process, GRFPU offers a significant performance improvement over the existing solutions. Although primarily developed for the LEON processor, GRFPU can be used as a building block in a custom compute engine or a DSP. GRFPU is easily interfaced to a LEON processor through the GRFPU Control unit (GRFPC).

GRFPU and GRFPC are written in high- level synthesizable VHDL code and are suitable for both ASIC and FPGA development.

For space applications GRFPU and GRFPC are available in fault-tolerant versions with full SEU protection.

2. FPU Design Challenges

GRFPU complies to the IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754)[1]. The IEEE-754 standard defines floating-point number formats, operations, exceptions and their handling. A system conforming to the IEEE-754 standard can be realized in software, hardware or combination thereof. System level performance is greatly affected by the amount of floating-point (FP) operations implemented in hardware and the throughput and latency of the operations.

Optimizing a FPU for the most common operations such as addition, subtraction and multiplication might be tempting, but studies have shown that lack of hardware support for complex instructions, such as division and square-root, decreases overall system performance. Even if implemented in hardware, the latency of division and square-root operations affect overall system performance. A poor implementation of division and square- root can result in a 0.5 increase of overall CPI (clock cycles per instruction) for heavy floating-point applications[2].

3. GRFPU Architectural Features

GRFPU implements all floating-point arithmetic operations defined by the IEEE-754 standard in hardware. The advanced design combines high throughput and low latency. The most common operations such as addition, subtraction and multiplication are fully pipelined with throughput of one clock cycle and a latency of three clock cycles. Computationally more complex divide and square-root operations take between 1 to 24 clock cycles to complete and execute in parallel with other FP operations.

GRFPU division and square-root operations are based on the functional iteration class of algorithms and in particular series expansion algorithms[3]. The main advantages of series expansion is low-latency operation and possibility to share multiplier hardware between multiplication, division and square-root. A shared multiplier is key to achieving a high performance/area ratio. In order to implement series expansion efficiently, the GRFPU starts a computation at starting point stored in approximation tables.

A logical view of the GRFPU is shown in figure 1. The most common operations are fully pipelined and will take three clock-cycles. Division and square-root execute in a separate non-blocking iteration unit.

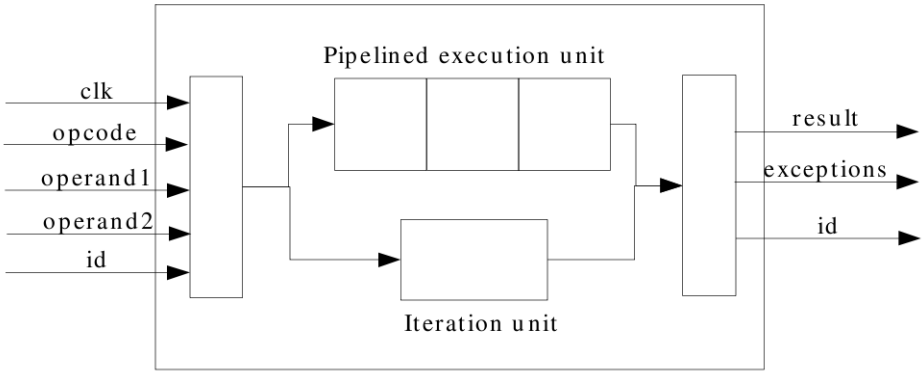

Figure 1: Logical view of GRFPU

GRFPU throughput and latencies are summarized in table 1.

Operation	**Throughput**	**Latency**
FADD, FSUB, FMUL, COMP, CONV	1	3
FDIVS/FDIVD (single/double precision)	15/16	15/16
FSQRTS/FSQRTD (single/ double precision)	23/24	23/24

Table 1: GRFPU throughput and latency

Figure 2 shows FP operation timing. Fully pipelined operations can be started on every clock cycle and complete in three clock cycles. Division and square root operate in parallel with other operations. The figure shows addition (FADDS) and multiplication (FMULD) operating in parallel with division. This interleaving of operations allows for out-of-order instruction execution.

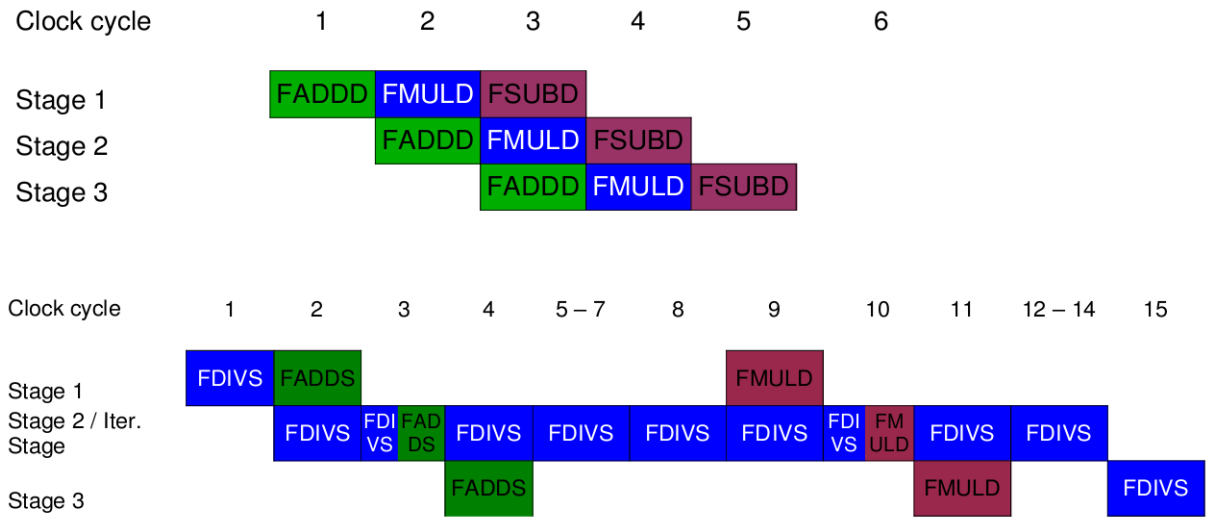

Figure 2: FP operation timing

Handling of denormalized FP numbers is deferred to software in order to minimize hardware complexity. The IEEE standard defines several ways to round a result smaller than the minimum normalized FP number. GRFPU rounds these results to zero.

4. GRFPU Implementation

Figure 3 shows a block diagram of the GRFPU. Four main blocks can be identified: unpack/decode logic, add/sub pipeline, FP multiplier and iteration block.

The most complex part of the GRFPU is the fast FP multiplier used for multiplication and iteration (division and square-root) operations. The multiplier is capable of multiplying two double-precision operands and is adapted for division and square-root algorithms. The precision of the multiplier is even greater than required by double precision multiplication to ensure convergence and accuracy of iteration algorithms.

In the first stage, partial products are created using Booth's algorithm[4]. The partial products are compressed using a Wallace-tree[4], producing the result of multiplication on carry- save form in the second stage. In the last stage, addition of the carry- save vector is performed combined with rounding and computation of iteration intermediate results.

Division and square-root latency is kept low by deriving the starting point of the iteration from approximation tables. The approximation tables are implemented as ROM and are part of the iteration block. The iteration block controls the buffer containing intermediate results of the iteration algorithm and handles sequencing of iteration operations through the multiplier.

The add/sub block computes the result of addition, subtraction, compare and convert operations.

5. Performance

GRFPU compares well against other FPUs on the market. Table 2 shows throughput and latency for comparable FPU cores. Compared to the MEIKO FPU, which is currently used in ERC32 and LEON systems, GRFPU gives a significant performance improvement. The higher performance is achieved by the low latency operations and pipelined execution.

FPU	FADDD	FMULD	FDIVD	Frequency	Area	Comments
GRFPU	1 (3)	1 (3)	16 (16)	250	100	0.13 µm, synthesis
ARM VFP9-S	1 (4)	2 (5)	28 (31)	140	100	0.18 µm, synthesis
ARM VFP11	1 (5)	2 (10)	29 (33)	350	100	0.13 µm, hard-block
AMD K7	1 (2)	1 (4)	17 (20)	500	?	0.13 µm, hard-block
MEIKO	8	10	50	140	25	0.18 µm, synthesis

Table 2: FPU comparison

GRFPU and GRFPC are written in high-level technology independent VHDL code and can be synthesized both for FPGA and ASIC technologies. Typically speed is 250 MHz on a 0.13 process with area of 100 kgates (GRFPU+GRFPC) and 65 MHz on a Virtex-II FPGA using approximately 8500 LUTs.

6. Integration with LEON

The GRFPU can be attached to a LEON processor through the GRFPU Control unit (GRFPC). The control unit receives floating-point instructions from the LEON integer unit (IU) and schedules them for execution by the GRFPU. The floating–point operations are executed in parallel with other integer instructions. The control unit handles the FPU register file, FPU status register and handles floating–point exceptions providing full compliance with the SPARC V8 instruction scheduling and trap model[5].

The GRFPC provides several features improving the overall system performance. FP operations do not block the IU pipeline and FP operations complete even if the IU pipeline is blocked (e.g. in case of a cache miss). The LEON integer pipeline is only stalled in case of operand or resource conflicts. GRFPC exploits GRFPU's capability of out-of-order execution of the FP operations, the instruction stream following floating-point division or square-root instructions is only stalled in case of operand or source conflicts.

A block diagram of GRFPC is shown in figure 4. The FP instructions are decoded in the first stage of the GRFPC pipeline. During the second stage FP register file access is performed. Data dependencies with the result of an operation in later stage is resolved by forwarding logic.

During the next stage a FP operation is started on the GRFPU while the instruction is inserted in a low-latency instruction buffer.

The write-back stage handles the results of the FP operations. Floating-point exceptions are detected and the FP status register and the FP deferred queue are updated. If an instruction completes and does not cause an exception its result is written to the FP register file. High-

latency instructions, whose result might not be available until several clock cycles after the instruction reaches the write-back stage, are inserted in a high-latency instruction buffer.

7. System Level Performance

The GRFPU's high performance combined with parallel execution of floating-point and integer instructions gives a significant overall system performance increase compared to existing solutions. In a typical system with LEON2 and GRFPU/GRFPC running at 100 Mhz, a heavy FP application written in C and compiled with standard C-compiler gives 30-40 MFLOPS. If critical parts of FP software are hand-optimized and written in assembler a performance of 40-70 MFLOPS can be achieved.

A typical GNC application runs 60 % faster on LEON2 + GRFPU/GRFPC compared to running on LEON2 + MEIKO at the same clock frequency.

8. Fault tolerance (applicable to fault-tolerant version)

Both GRFPU and GRFPC include SEU protection by design. The FPU register file is protected using parity coding and sparing, while all other registers are, optionally, protected with TMR.

9. Validation and testing

Validation of FPUs has shown to be a very hard task, and several FPUs on the market have had bugs which were not found before large-scale deployment. Validation and testing of the GRFPU was therefore prioritized and performed in several stages during the development work.

During the early phase of the development the accuracy and correctness of the floating-point algorithms were proved mathematically. Before implementing the GRFPU in hardware, a model of the GRFPU was written in C and attached to TSIM/LEON simulator using the loadable module interface. This provided a possibility to model the GRFPU and perform extensive simulations of the LEON processor with the GRFPU before implementation of the actual hardware started. The high performance of the TSIM simulator (+20 MIPS) allowed large and exhaustive test programs to be executed. The use of a simulator also provided an efficient debugging environment.

Research in the area of FPU validation has been intensive during past years and has resulted in a number of floating-point test software packages. Several software packages were used during the validation of the GRFPU. UCBTEST[6] uses number theory to generate hard cases of floating-point operations. TestFloat[7] uses a large set of test vectors and random data to check the FPU implementation against its own software implementation of floating-point arithmetic. IeeeCC754[8] checks if the implementation is compliant to the IEEE-754 standard.

10. Conclusion

GRFPU is IEEE-754 compliant high- performance FPU that compares well against other FPUs on the market. Advanced design combines high throughput and low latency for floating- point operations. GRFPU has been extensively validated during several stages of

development work. GRFPU can easily be integrated with LEON processor using the GRFPU Control unit GRFPC. GRFPC offers parallel and out - of- order FP instruction execution delivering high system level performance for heavy FP applications. Great performance increase is achieved with GRFPU/GRFPC compared to MEIKO solution. Portability and FT capabilities make GRFPU/GRFPC suitable for long-term space use. GRFPU and GRFPC are designed in high-level synthesizable VHDL-code making it independable of target technology and suitable for system- on- chip (SOC) prototyping on FPGAs.

References

1. "Standard for Binary Floating - Point Arithmetic", ANSI/IEEE Standard No. 754-1985, The Institute of Electrical and Electronics Engineers, Inc., New York, August 1985.
2. "Implementing Division and Other Floating- Point Operations: A System Perspective", Stuart F. Oberman and Michael J. Flynn, in Proceedings of SCAN- 95, International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics, Wuppertal, Germany, September 1995.
3. "An Analysis of Division Algorithms and Implementations", Stuart F. Oberman and Michael J. Flynn, July 1995.
4. "Computer Arithmetic Algorithms", Israel Koren, A. K. Peters Ltd., 2nd edition, 2002.
5. "The SPARC Architecture Manual Version 8", SPARC International, Prentice Hall, 1992
6. UCBTEST suite, available at http://www.netlib.org/fp/ucbtest.tgz
7. TestFloat, available at http://www.jhauser.us/arithmetic/TestFloat.html
8. IeeeCC754, available at http://www.win.ua.ac.be/~cant/index.html

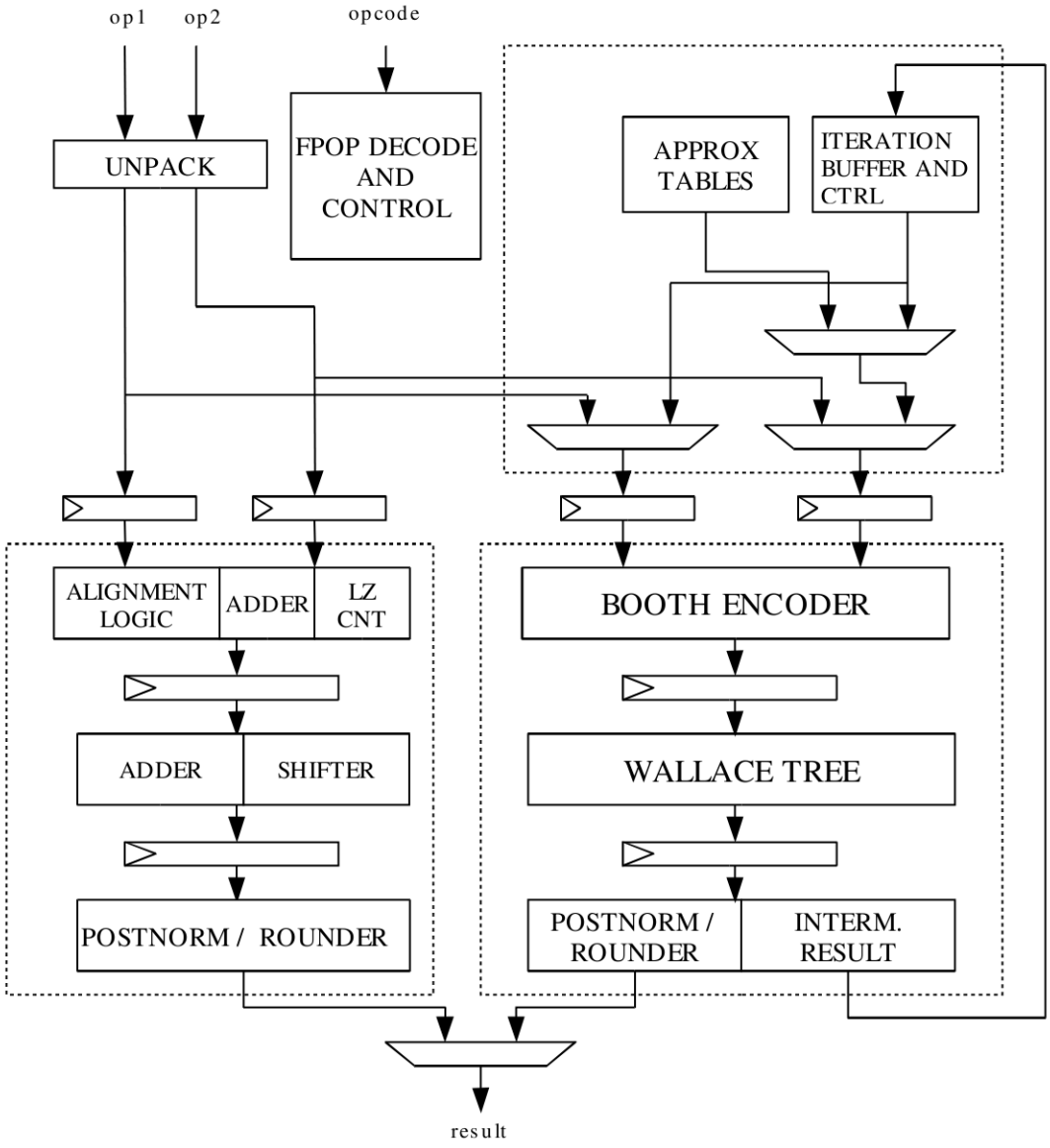

Figure 3: GRFPU Block Diagram

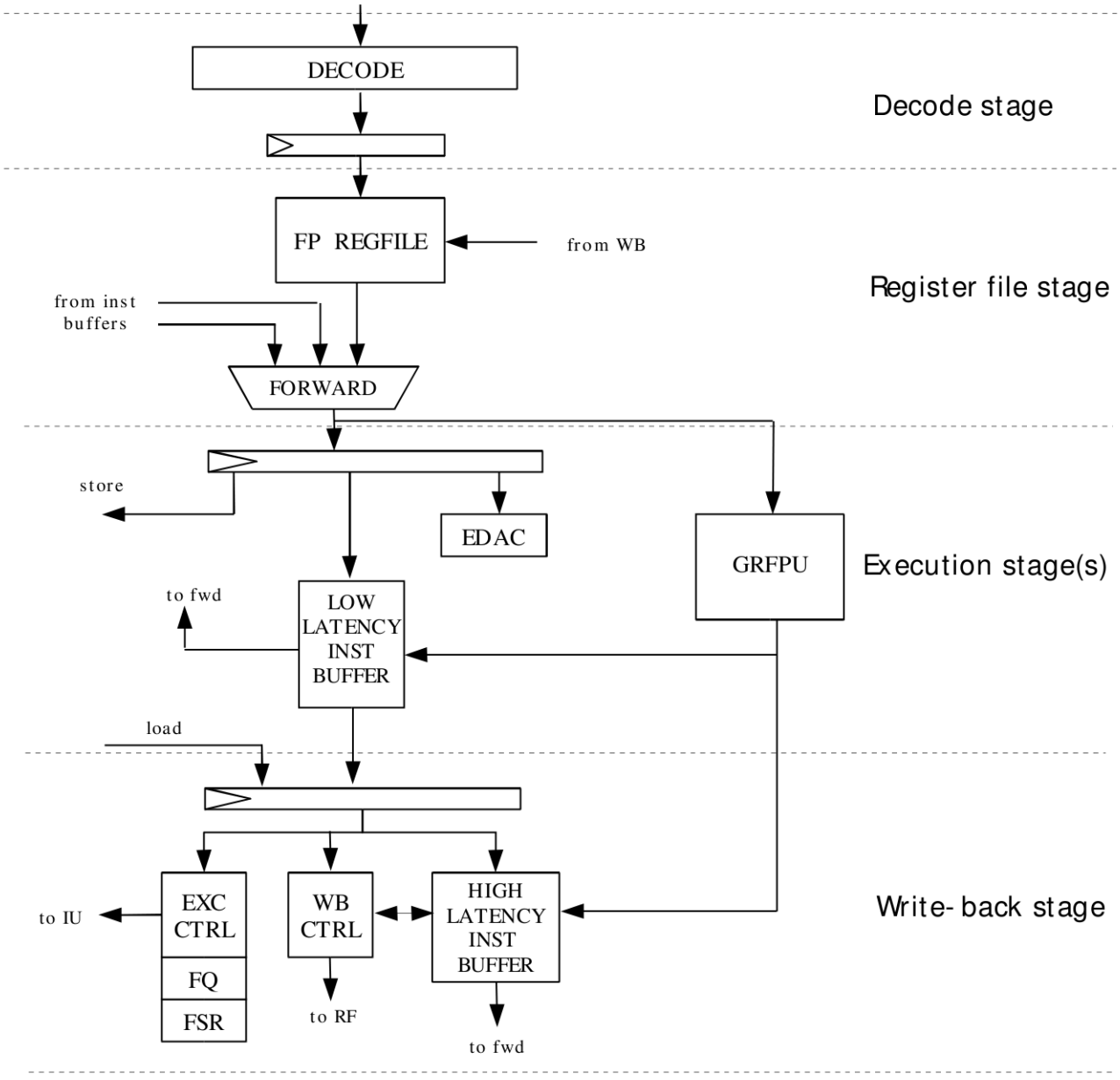

Figure 4: GRFPC Block Diagram