GRFPU - High Performance IEEE-754 Floating-Point Unit

Edvin Catovic
Gaisler Research
edvin@gaisler.com

GRFPU Overview

- IEEE754 compliant supporting single and double FP numbers
- Primarily developed for use with LEON
- Significant system performance improvement over existing solutions
- Extensively validated
- Fault Tolerant
- Written in high-level synthesizable VHDL-code

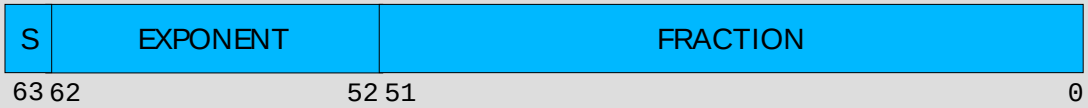
IEEE-754 Standard for Binary Floating-Point Arithmetic

- Formats

Single precision FP

$$fp_{single} = (-1)^s 1.f * 2^{exp-127}$$

Double precision FP



$$fp_{double} = (-1)^s 1.f * 2^{exp-1023}$$

- Operations
  - Arithmetic: Addition, subtraction, multiplication, division and square-root
  - Comparison
  - Format conversions: FP to integer, integer to FP
- Rounding
  - 4 rounding modes: round-to-nearest, round-to-zero, round-to- $+\infty$, round-to- $-\infty$
- Exceptions
  - Invalid operation, division by zero, overflow, underflow, inexact

FPU Design Challenges

- FP Algorithms
  - Complexity
  - Correctness and accuracy
  - IEEE-754 compliance
- System architecture
  - HW or SW support ⇒ Affects overall system performance
- Hardware design
  - Complex high precision operations ⇒ Large data paths
  - Tradeoffs to achieve high performance/area
- Test and validation

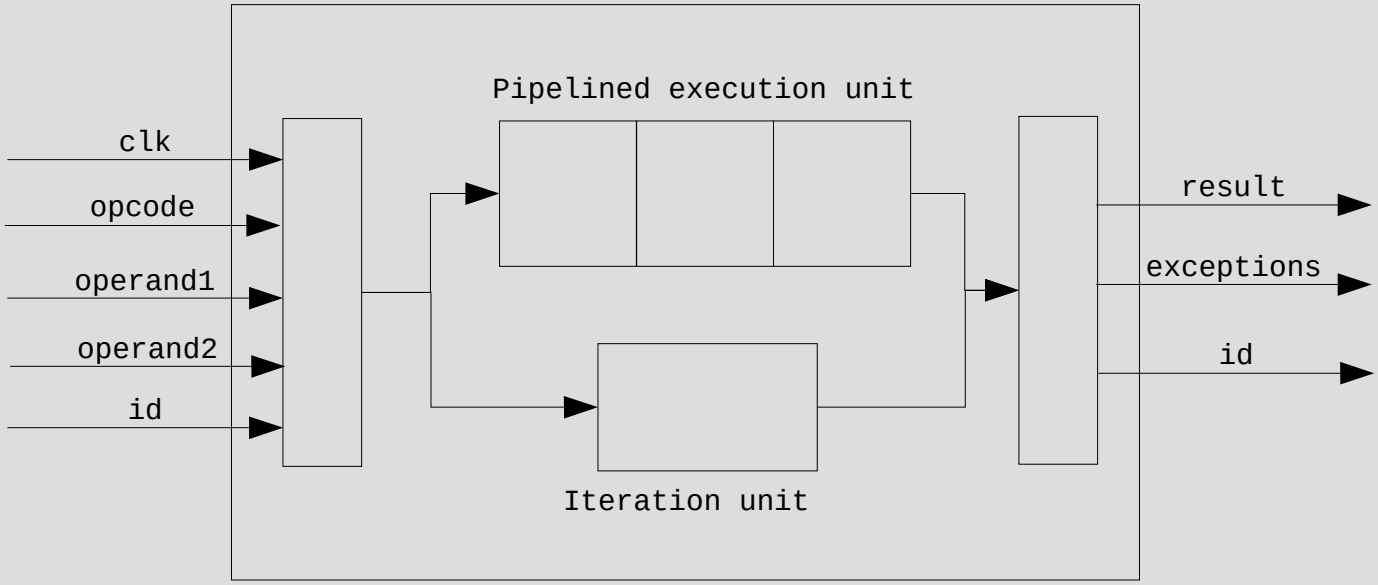
FPU Algorithms

- System level performance impact
  - Latency and throughput
  - HW Support
- Division and square-root
  - Lack of support in HW ⇒ overall CPI increase ⇒ system performance degradation
  - High latency (> 30 clock cycles) ⇒ significant CPI increase
- Div/sqrt by digit-recurrence
  - Dedicated HW, high latency
- Div/sqrt by functional iteration
  - Low latency div and sqrt operations
  - Multiplication is basic step ⇒ Multiplier can be shared between mul, div and sqrt
  - Small area overhead ⇒ high performance/area

GRFPU Architectural Features

- Implements all SPARC V8 FP operations
- Efficiently implementation of complex FDIV and FSQRT operations
- Low latency and high throughput
- Special handling of denormalized numbers
  - Operations on denormalized inputs deferred to software
  - Tiny results flushed to zero (allowed by IEEE-754)
  - Fast non-IEEE mode
- Fast FP multiplier
  - Capable of performing multiplication on two DP operands ⇒ low latency mul
  - Adapted for division and square-root ⇒ good performance/area tradeoff
  - Non-blocking division and square-root

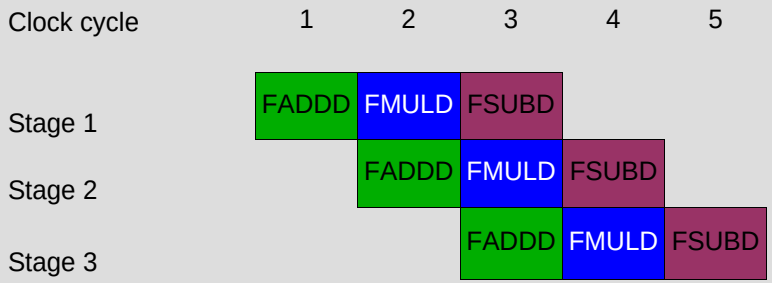
GRFPU – Logical View



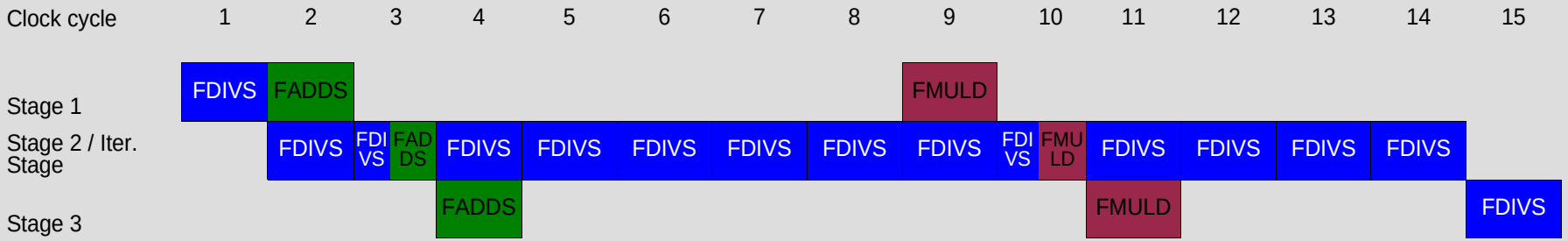
- All SPARC V8 FP operations
- FADD, FSUB, FMUL, FCMP and CONV are fully pipelined
- Separate non-blocking iteration unit (FDIV and FSQRT)

GRFPU Operation Timing Example

- Fully pipelined operations (FADD, FSUB, FMUL, FCMP, CONV)



- Fully pipelined operations interleaved with FDIV (or FSQRT)



  - FDIV and FSQRT are non-blocking operation
  - All others operations can be interleaved with FDIV or FSQRT
  - Operations can complete out-of-order

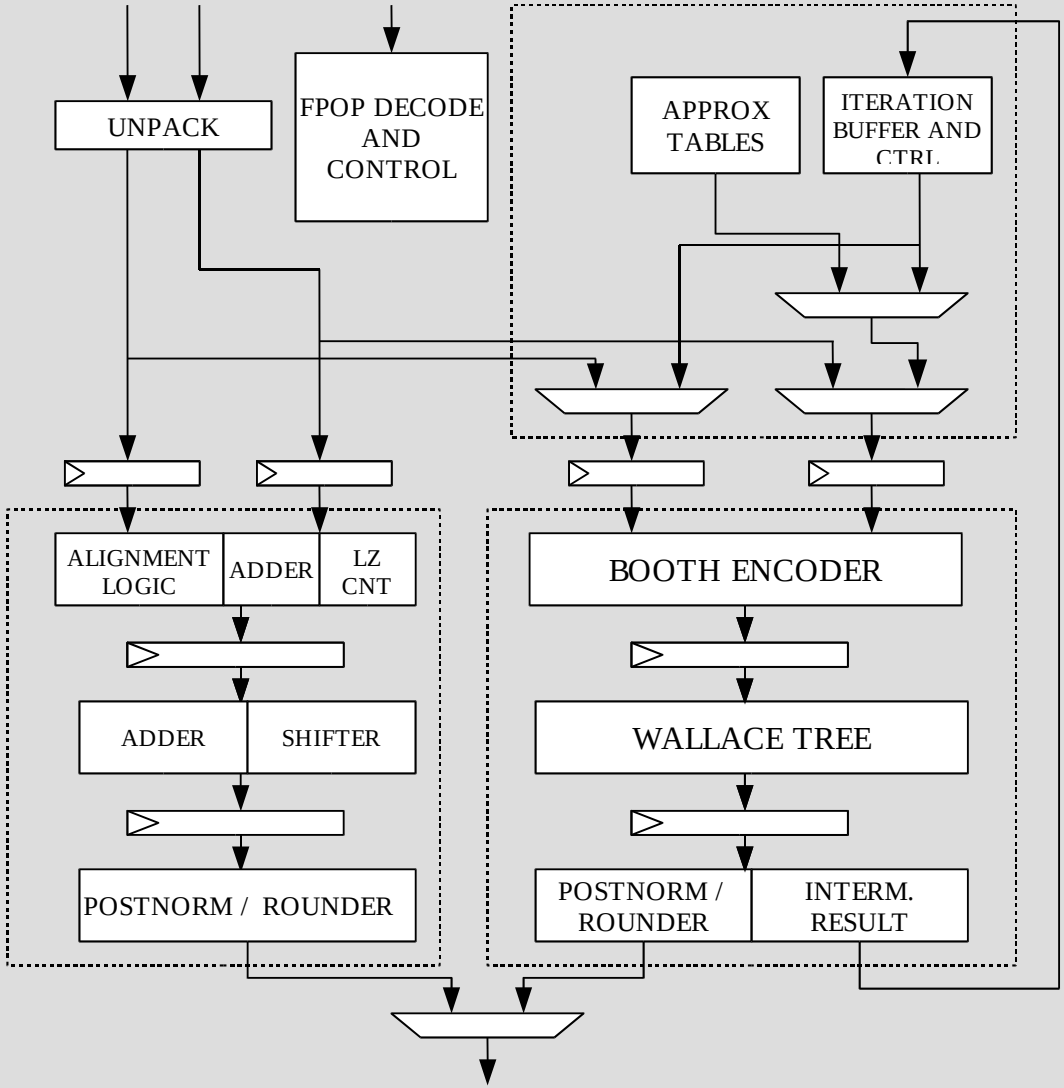
Performance

- Throughput and latency

OPERATION	THROUGHPUT	LATENCY
FADDD, FSUBD, FMULD, COMP, CONV	1	3
FDIVS	15	15
FDIVD	16	16
FSQRTS	23	23
FSQRTD	24	24

- Frequency
  - 250 MHz on 0.13 um standard-cell ASIC process
  - 65 MHz on Virtex-II FPGA
- Area
  - 100 kgates on ASIC
  - 8500 LUTs on Virtex-II FPGA

GRFPU Block Diagram



FPU Comparison

- Table shows throughput and (latency)

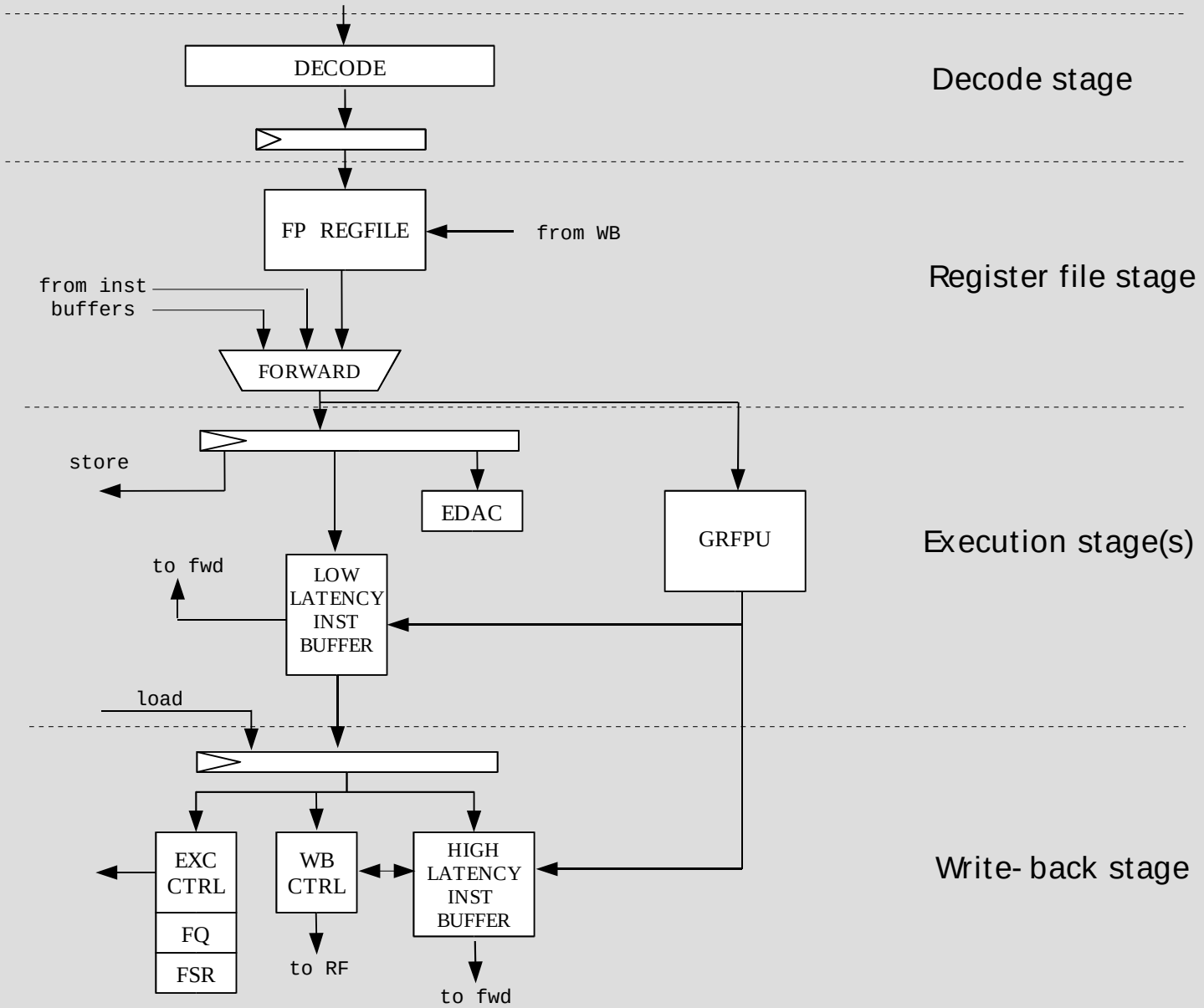
FPU	FADDD	FMULD	FDIVD	FREQ	AREA	COMMENTS
GRFPU	1 (3)	1 (3)	16 (16)	250	100	0.13 um, synthesis
ARM VFP9-S	1 (4)	2 (5)	28 (31)	140	100	0.18 um, synthesis
ARM VFP11	1 (5)	2 (10)	29 (33)	350	100	0.13 um, hard-block
AMD K7	1 (2)	1 (4)	17 (20)	500	?	0.13 um, hard-block
MEIKO	8	10	50	140	25	0.18 um synthesis

- GRFPU compares well against other FPUs on the market

GRFPU Controller - GRFPC

- GRFPC provides an interface between LEON and GRFPU
- Schedules SPARC FPOPs for execution on GRFPU
- Handles FP register file (32 x 32-bit FP registers)
- Exception handling (FP Status register, FP deferred queue)
- Parallel execution of FP and integer operations
  - FP operations do not block IU pipeline and vice versa
  - FP load and store handled by IU
- Out-of-order execution of FP instructions
- Full compliance with SPARC V8 instruction scheduling and trap model

GRFPC Block Diagram



Instruction Trace Example

TIME	ADDRESS	INSTRUCTION	RESULT
262843492	40003700	fsubs %f6, %f3, %f6	[10000060]
262843493	40003704	ld [%o0 + 0xc], %f5	[3cc90aaf]
262843494	40003708	fmuls %f2, %f4, %f2	[00000054]
262843495	4000370c	fmuls %f5, %f6, %f3	[00001fe1]
262843499	40003710	fsubs %f2, %f3, %f2	[00000054]
262843503	40003714	st %f2, [%o7 + %o3]	[40014dd8]
262843504	40003718	ld [%o1 + %i5], %f3	[c1200000]
262843505	4000371c	add %o7, 8, %o7	[00000028]
262843506	40003720	ld [%o1 + %i0], %f4	[c1200000]
262843507	40003724	add %i5, 8, %i5	[00000418]
262843510	40003728	fsubs %f4, %f3, %f4	[4000a800]
262843511	4000372c	ld [%o0 + 0x8], %f2	[3f7fec43]
262843515	40003730	fmuls %f2, %f6, %f2	[00000054]
262843516	40003734	fmuls %f5, %f4, %f5	[00001fe1]

Application Level Performance

- High performance provided by GRFPU

  Parallel integer and floating-point instruction execution

  ⇒ Overall system level performance increase

- Example: LEON2 + GRFPU/GRFPC running at 100 MHz
  - GRFPU @ 100 MHz: 100 MFLOPS peak FP performance
  - C-code: 30 - 40 MFLOPS @ 100 MHz
  - Hand coded assembly: 40 - 70 MFLOPS @ 100 MHz
- Large overall system level performance increase for heavy FP applications
  - A typical GNC application runs 60 %faster with GRFPU compared to MEIKO (at the same clock frequency)

Fault Tolerance

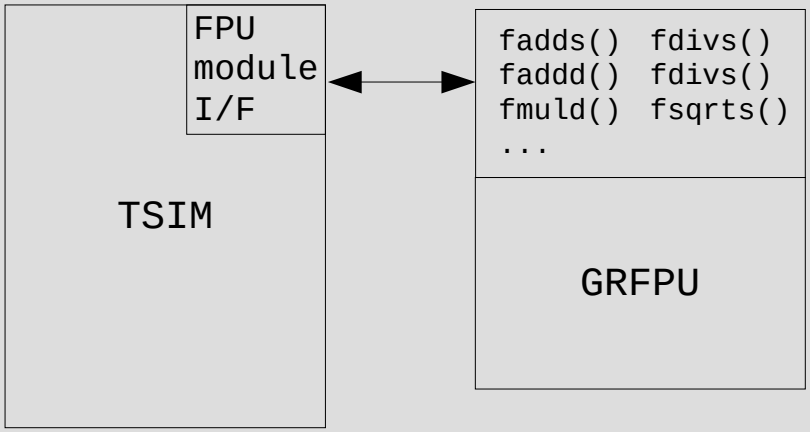
- GRFPU and GRFPC are SEU protected by design
- TMR registers
- FP register file is protected using (32, 7) BCH code (SEC/DED)
- Integrated with LEON instruction restart capability

Design methodology and validation

- FP algorithms are highly complex (specially divide and square-root)
- Validation showed to be a very hard task
  - Several cases of bugs in commercial processors were detected after large-scale deployment (Pentium divide-bug)
- GRFPU Design Work:
  - Phase 1: Development of FP algorithms. Correctness, accuracy and convergence of the FP algorithms were mathematically proved
  - Phase 2: Development of high level FPU model in C and attaching it to TSIM simulator. FP test programs and real-life software could be run on the model before development of HW started.
  - Phase 3: HW development
  - Phase 4: Test and validation
- Validation performed during several stages of the development work

Design methodology and validation (2)

- TSIM + GRFPU as loadable module



- TSIM simulates full functionality of LEON, memory and peripherals
- Provides an interface to attach user-defined FPU model
- Possible to test the FP algorithms before HW implementation started
- Offers high performance (+20 MIPS)
  - Large and exhaustive test programs were run on the C-model (UCBTEST, SoftFloat, IeeeCC754, GNC application)
- Used as golden model in later stages of the development work
- Efficient debugging environment

Design methodology and validation (3)

- Floating-Point Test programs
  - UCBTEST: Uses number theory to generate hard case test vectors.
  - TestFloat: Checks FPU implementation by comparing it against its own software implementation. Uses large set of test vectors + random data.
  - IeeeCC754: Checks IEEE754 compliance
- Run on both final implementation as well as a C-model of the GRFPU

Summary

- GRFPU/GRFPC offer significant performance improvement over existing solutions (LEON/MEIKO or ERC32/MEIKO)
- GRFPU compares well against other implementations
- Scheduled for 2 SOC designs
- Portability and FT capabilities makes it suitable for long-term space use