Formal Verification of the VAMP Floating Point Unit

Christoph Berg Christian Jacobi

{cb,cj}@cs.uni-sb.de

Saarland University

Institute of Computer Architecture, Prof. W. J. Paul

Saarbrücken, Germany

Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

Project Description

Project at Saarland University, Institute of Prof. W. J. Paul

- Verification of a complete CPU featuring
  - MIPS instruction set architecture [Kröning]
  - Tomasulo scheduler
  - Precise, nested interrupts
  - MMU: cache, virtual memory, TLB [Beyer]
  - IEEE compliant Floating Point Unit [Berg,Jacobi]

- Verification on the gate-level
- Designs based on [MP00]
- Verification using theorem proving: PVS [CRSS94]
- Automatic translation to Verilog HDL [Beyer,Leinenbach]
- Implementation on Xilinx FPGA [Leinenbach,Meyer]

FPU Features

- Single and double precision
- Add/sub, mul/div, conversion/comparison,
  each a separate unit
- All 4 IEEE rounding modes supported
- Normal/denormal operands and results
- Exceptions
- Out-of-order control for operation within Tomasulo CPU

Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

Related Work

- Designs and paper proofs taken from [MP00]
- Theory of Rounding [EP97,MP00]

- Formalization of IEEE standard in theorem provers [Min95,Har99,Jac01]

- Verified FPUs
  - AMD [MLK98, Rus 97,98,00]
  - Intel [OZGS99,AJK+00]

- Verified components [AS95,CCH+96,CB98]
- Verified algorithms [ML96, Har97]

Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

IEEE Formalization

- IEEE-Standard 754 informal

- Factorings (s, e, f) [Matula]
  - Sign-Bit $s \in \{0, 1\}$
  - Exponent $e \in \mathbb{Z}$
  - Significant $f \in \mathbb{R}, f \geq 0$

- Value $[\![s, e, f]\!] := (-1)^s \cdot 2^e \cdot f$

$\rightsquigarrow$ reason about numbers in theory of rounding

$\rightsquigarrow$ reason about bitvectors only in hardware

IEEE Formalization (cont.)

A factoring (s, e, f) is called

- *normal*, if $e \geq e_{min}$ and $1 \leq f < 2$
- *denormal*, if $e = e_{min}$ and $0 \leq f < 1$
- *IEEE factoring*, if normal or denormal
- p*-representable*, if $2^{p-1} \cdot f \in \mathbb{N}$

THEOREM:

Every real $x \neq 0$ has a unique IEEE factoring $\eta(x) = (s, e, f)$ with $[\![\eta(x)]\!] = x$.

Rounding

"... In this mode, the representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero shall be delivered ..." [IEEE]

- Formalization of the rounding function in two steps: [Min95]

1. Round to integer:

$$rd_{int}(x) := \begin{cases} \lfloor x \rfloor & \text{if } x - \lfloor x \rfloor < \lceil x \rceil - x \\ \lceil x \rceil & \text{if } x - \lfloor x \rfloor > \lceil x \rceil - x \\ x & \text{if } \lfloor x \rfloor = \lceil x \rceil \\ 2 \lfloor \lceil x \rceil / 2 \rfloor & \text{else} \end{cases}$$

2. Scale appropriately: let $x \in \mathbb{R}$, and $(s, e, f) = \eta(x)$:

$$rd(x) := 2^{e-p} \cdot rd_{int}\big(x \cdot 2^{-(e-p)}\big).$$

- Conformance to the standard is a theorem.
- Similar definitions for other rounding modes.

Decomposition Theorem

Decompose rounding into three steps:

1. η-computation (pre-normalization)

$$(s, e, f) := \eta(x)$$

2. significand rounding

$$sigrd(f) := 2^{-(p-1)} \cdot rd_{int}(f \cdot 2^{p-1})$$

3. post-normalization

$$postnorm(s, e, f) := \begin{cases} (s, e+1, 1) & \text{if } sigrd(f) = 2 \\ (s, e, sigrd(f)) & \text{otherwise} \end{cases}$$

THEOREM: 1. $rd(x) = [\![postnorm(\eta(x))]\!]$

2. $postnorm(\eta(x))$ is an IEEE factoring

$\rightsquigarrow$ similar decomposition of hardware

Exceptions

"Overflow ... shall be signaled whenever the destination format's largest finite number is exceeded in magnitude by what would have been the rounded floating-point result were the exponent range unbounded." [IEEE]

Formalization of OVF:

- $\text{OVF}(x) := |rd(x)| > X_{max} = 2^{e_{max}} \cdot (2 - 2^{1-p})$

Overflow can be decomposed:

- $\text{OVF\_before}(x) := |x| > 2^{e_{max}}$
- $\text{OVF\_after}(x) := (e = e_{max}$ and $\text{sigrd}(f) = 2)$

THEOREM: $\text{OVF}(x)$ iff $\big(\text{OVF\_before}(x)$ or $\text{OVF\_after}(x)\big)$

α-Equivalence

- Partition reals into equivalence classes by a relation $=_\alpha$ [EP97]
- α-Equivalence is mathematical counterpart to sticky-bit computation
- $x =_{e-p} x'$ implies $rd(x) = rd(x')$
- $x =_{e-p} x'$ implies $OVF(x) = OVF(x`)$
- Partition FPU into **Computational Units** and **Rounder**

Example: Division feeds quotient Q into rounder with

$$Q =_{e-p} a/b$$

Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

Verification of FPU Hardware

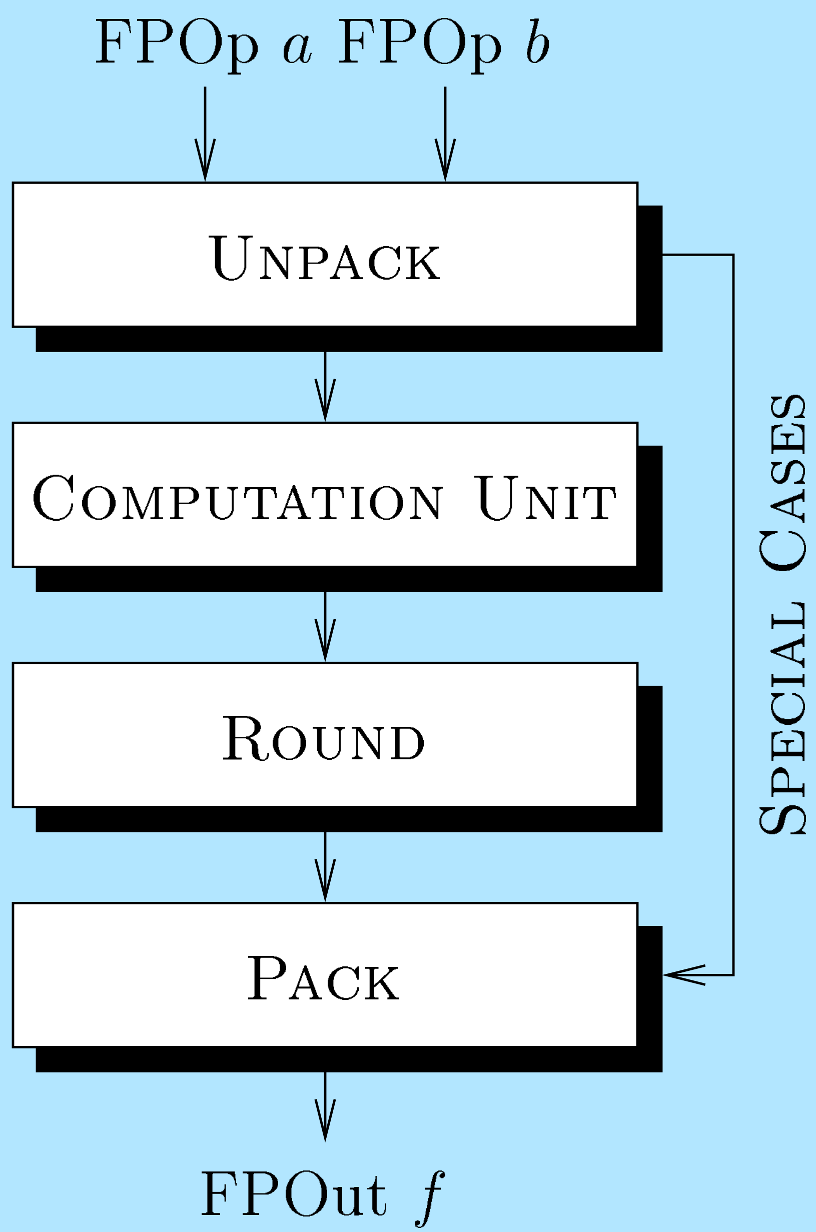
Unpack: Unpacks IEEE encoding, normalization for mul/div; handle special cases

Comp. Unit: Perform the actual operation, add/sub, mul/div, convert; compute α-equivalent result

Rounder: Round and compute exceptions

Pack: Pack to IEEE encoding

$\rightsquigarrow$ Each box is verified separately in PVS



Implementation of the Rounder

Inputs:

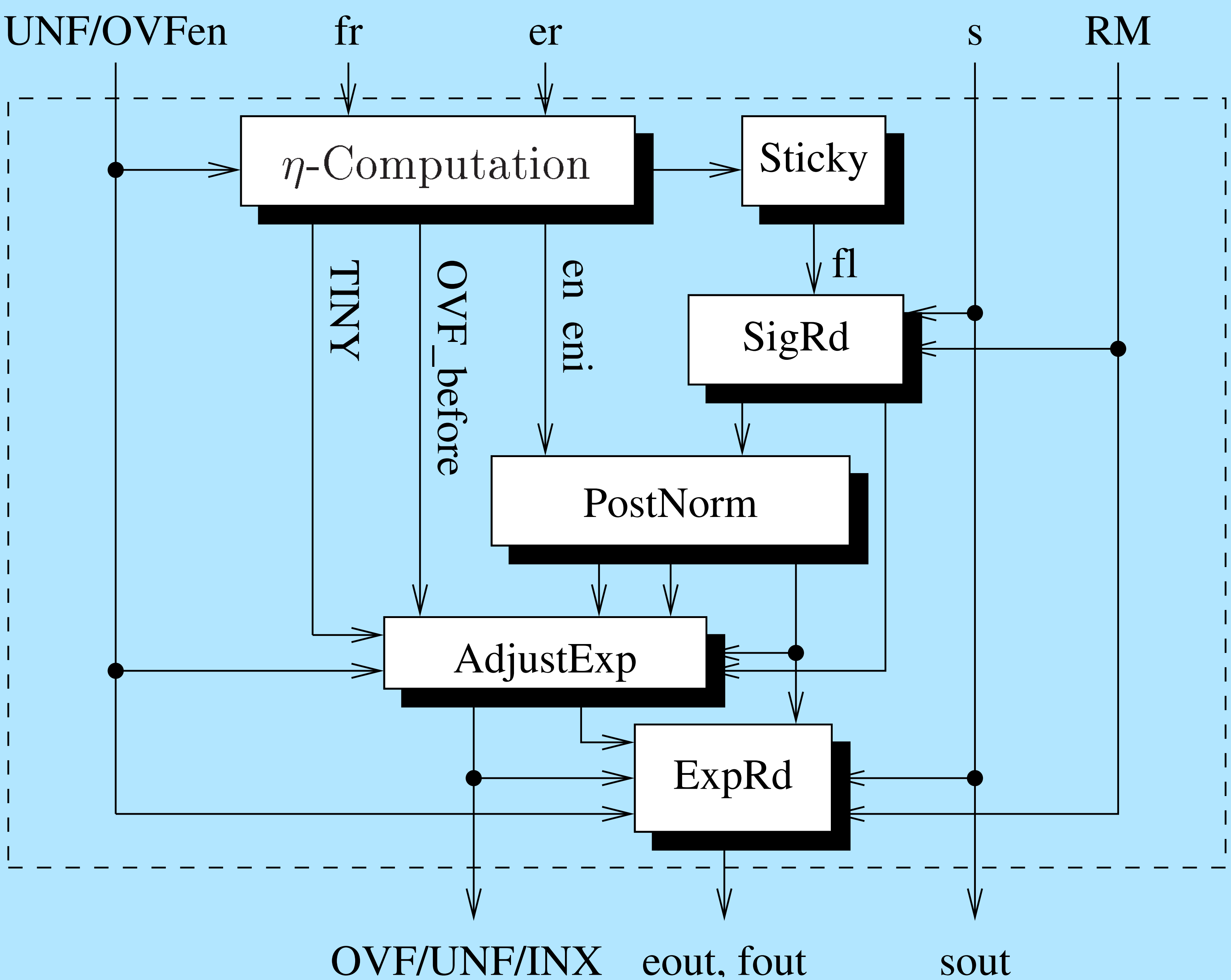
- intermediate result (s_r, e_r, f_r):
  - $s_r \in \mathbb{B}$ sign
  - $e_r \in \mathbb{B}^{13}$ exponent in 2's complement
  - $f_r \in \mathbb{B}^{57}$ binary fraction with 55 fraction bits
- $x =_{e-p} [\![s_r, e_r, f_r]\!]$
- input need not be normalized

- $f_r < 1$ implies $e_r \leq e_{max}$; this is different from [MP00]

Outputs:

- rounded result $rd(x)$ and exceptions.

Implementation of the Rounder



Decomposition Theorem: $rd(x) = [\![postnorm(\eta(x))]\!]$

Computation of OVF_before

OVF_before(x)

$$\iff |x| > 2^{e_{max}}$$

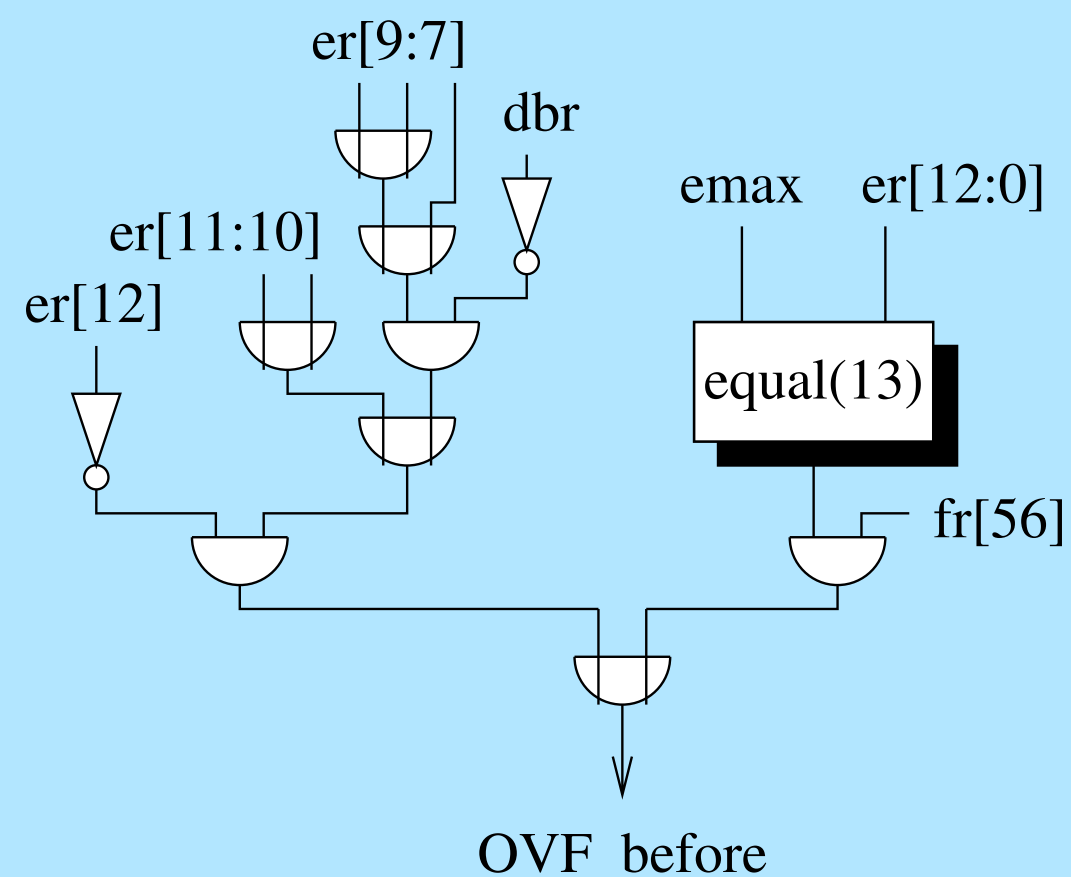
$$\iff 2^{e_r} \cdot f_r > 2^{e_{max}}$$

$$\iff e_r + \log f_r > e_{max}$$

$$\iff e_r > e_{max} \text{ or}$$

$$e_r = e_{max} \text{ and } f_r \geq 2,$$

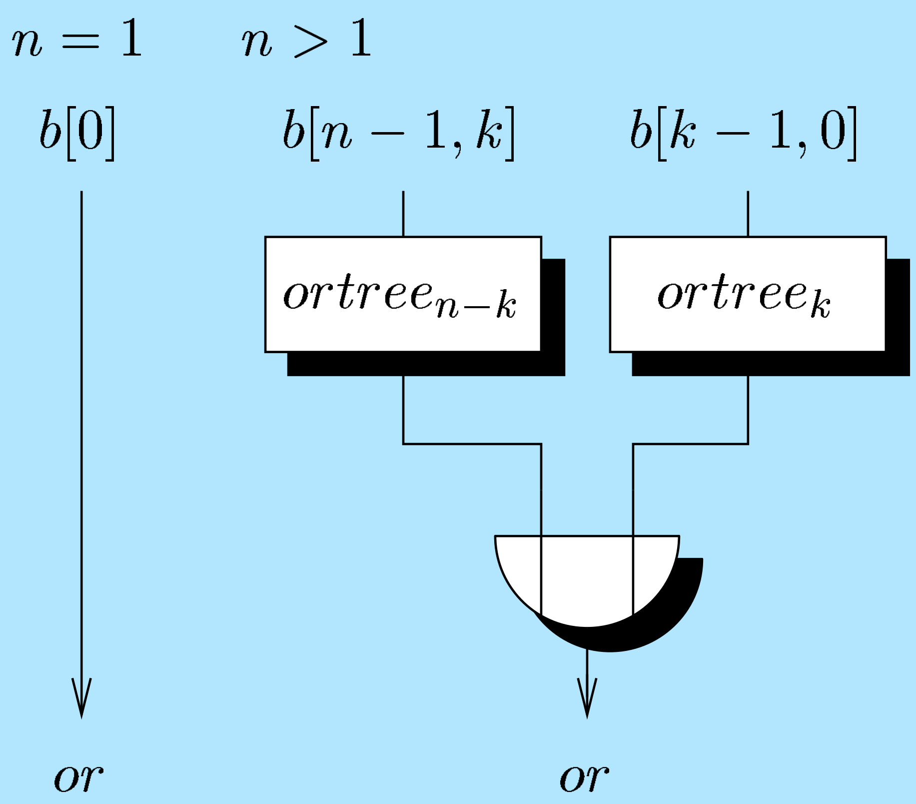
since $f_r < 1$ implies $e_r \leq e_{max}$



Evaluate high-order bits of e_r to check if $e_r > e_{max}$,

and compare e_r with e_{max} and check top-bit of f_r.

Combinatorial Hardware in PVS




```
ortree(n: nat, b: bvec[n]): Recursive bit =
  If n=1 Then
    b^0
  Else
  Let k=floor(n/2) In
    ortree(n-k, b^(n-1,k)) OR
      ortree(k, b^(k-1,0))
  Endif
```

Large library of verified basic circuits (or-tree, adder, ...) [BJK01]

Tool automatically translates PVS to Verilog.

Verification Effort (Combinatorial)

	PVS		MP00		
	Lemmas	Steps	Lemmas	Pages	Steps/Page
Basic Circuits	107	4050	4	23	176
Theory of Rounding	235	4388	9	33	132
Unpacker	35	671	0	5	134
Add/Sub	97	2079	1	14	148
Mul/Div	102	2666	5	18	148
Rounder	98	4027	5	22	183
Compare/Convert	21	901	1	20	45
Misc.	123	894			
$\sum$	823	19676	25	135	145

- large proof gaps in [MP00]
- many seemingly trivial things not proved in [MP00], e.g., bus widths

  $\rightsquigarrow$ this was source of several bugs
- lots of mathematics in [MP00] in the continuous text
- FPU verification: 2.5 person-years (incl. time to learn PVS)

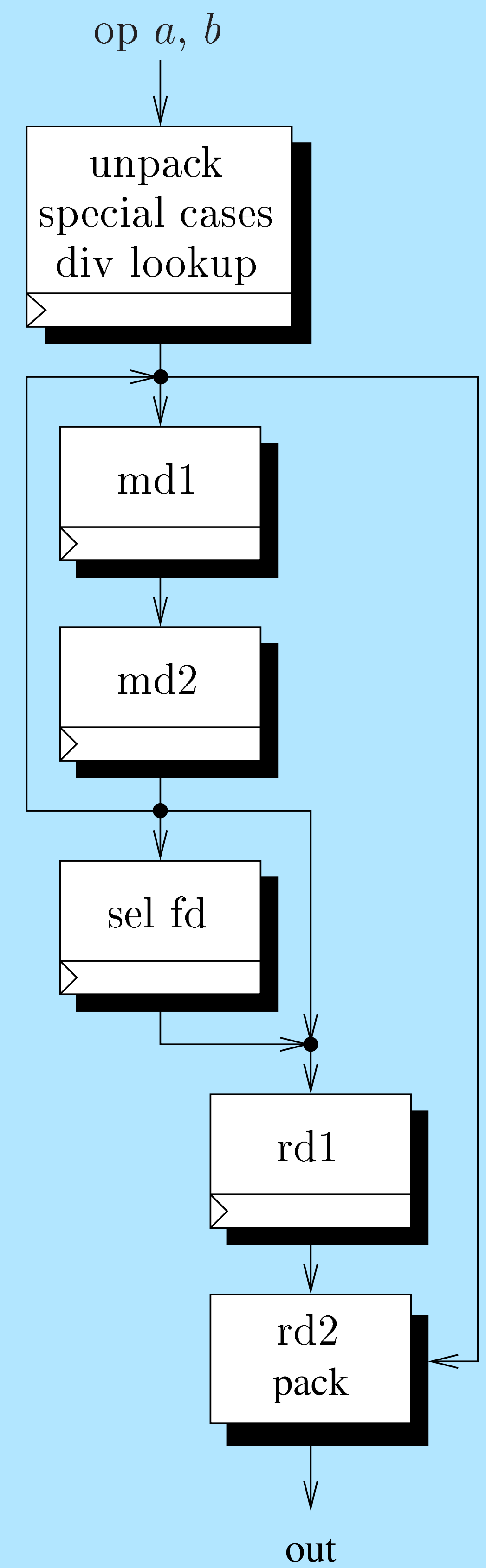
Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

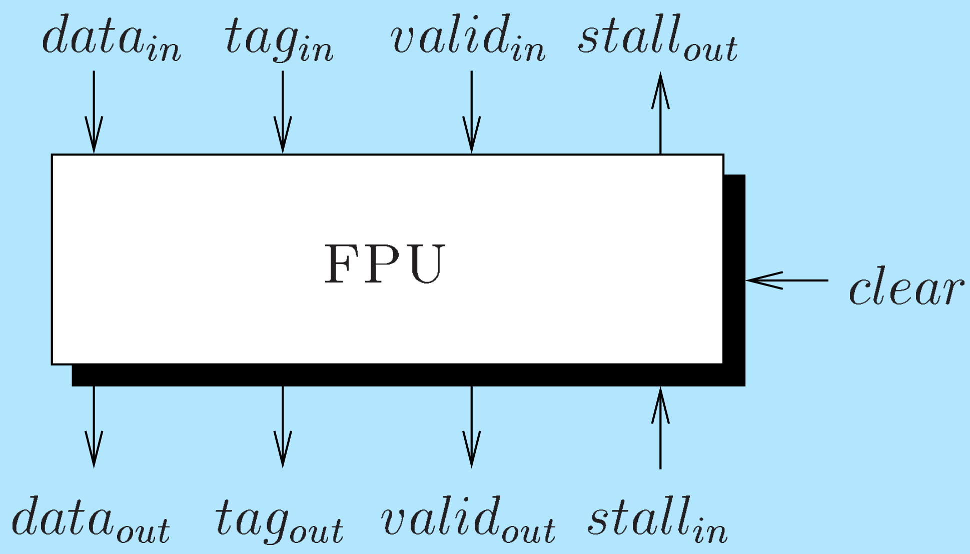
Verification of the Pipeline

- FPU incorporated into Tomasulo CPU
- Design&Verification of pipeline new
- Operations "flow" through pipeline, identified by tags
- **Liveness**: each operation eventually completes, and return value is correct
- **Consistency**: each completed operation was started
- Verification using theorem proving/invariant checking *extremely* costly

⇒ Combine model checking/theorem proving



Tomasulo Pipelines



Formal definition of liveness:

$$\begin{aligned} \forall t : valid_{in}(t) \wedge \neg stall_{out}(t) &\text{ implies} \\ \exists t' \geq t : valid_{out}(t') \wedge \neg stall_{in}(t') \wedge& \\ tag_{out}(t') = tag_{in}(t) \wedge& \\ data_{out}(t') = FPU(data_{in}(t)).& \end{aligned}$$

Similar definition for consistency.

Temporal Logic

$\forall$ INP : $\forall t: \exists t' \geq t : p(t')$

Advantages:

1. expressiveness
2. data can be handled
3. easy to use in PVS, e.g., tag-abstraction
4. used in the VAMP core

Disadvantages:

1. hard to prove with invariants
2. model checking not possible

$AG\ AF\ p$

Advantages:

1. easy to prove using model checking
2. easy to read

Disadvantages:

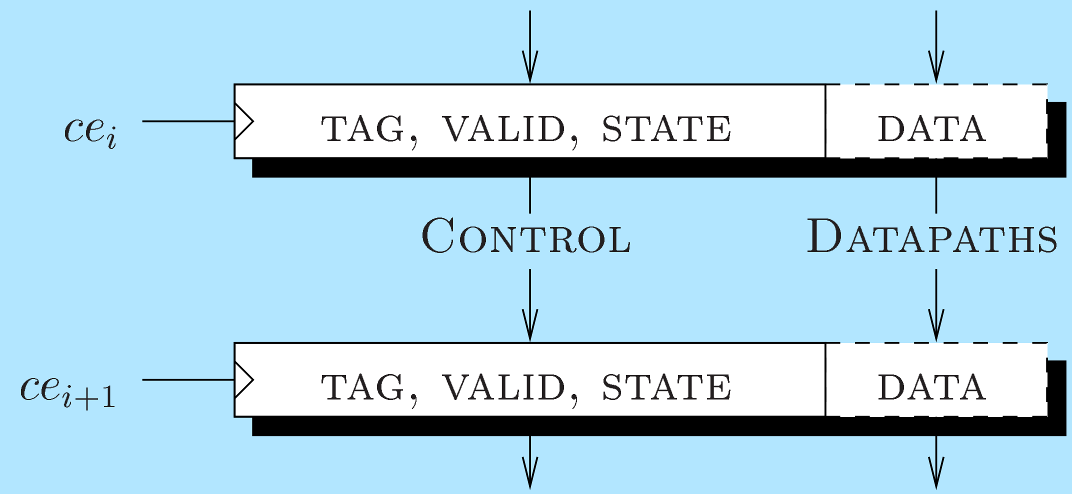
1. data can not be handled
2. abstractions complex to prove

$\rightsquigarrow$ use CTL/model checking to verify temporal properties (PVS built-in model-checker)

$\rightsquigarrow$ use theorem proving to prove that CTL matches its intended semantics

$\rightsquigarrow$ conclude liveness including functional correctness

Verification of Liveness



Idea: prove liveness of clock-enables with model-checking,
prove liveness of the pipeline (incl. data) with theorem proving

- clock-enables data-independent, only control-dependent
- extract pipeline control (including tags, iteration counter, etc.)
- verify liveness of clock-enables: $AG\ AF\ ce_i$
- conclude $\forall t\ \exists t' \geq t:\ ce_i(t')$
- "press" instructions through pipeline, thereby collecting computations on data

Outline

- VAMP project description & status
- Related work
- Formalization of the IEEE standard
- Verification of the FPU datapaths
- Pipelining
- Summary

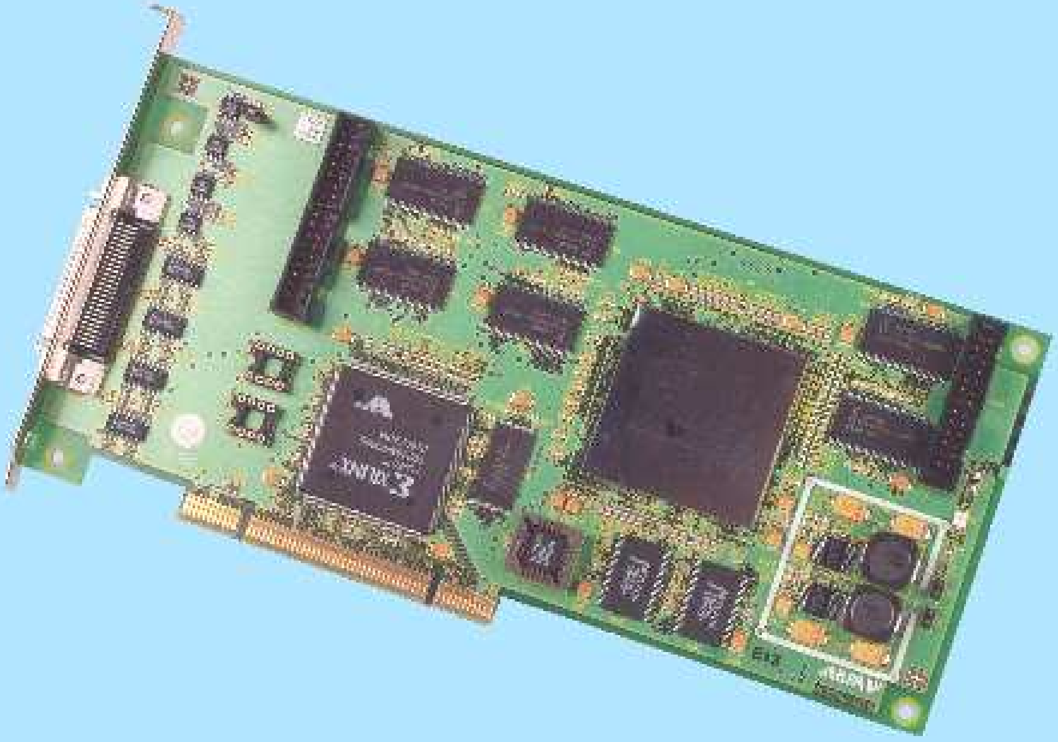
Summary

- Verification of a complex CPU & FPU
- Formalization of the IEEE Standard
- Verification of FPU datapaths & pipeline

$\rightsquigarrow$ combination of theorem-proving and model checking

- Translation to Verilog HDL
- Implementation on FPGA
- No hardware debugging necessary
- Mul/Div-FPU:

  8 MHz, 100K gates, 1630 latches



- Designs & proofs available at

  www-wjp.cs.uni-sb.de/projects/verification/

Literature

AJK$^+$00 Aagaard, Jones, Kaivola, et al.: Formal Verification of Iterative Algorithms in Microprocessors, DAC 2000

AS95 Aagaard, Seger: The Formal Verification of a Pipelined Double-Precision IEEE Floating-Point Multiplier, ICCAD 95

BJK01 Berg, Jacobi, Kröning: Formal Verification of a Basic Circuits Library. Applied Informatics, 2001

CB98 Chen, Bryant: Verification of Floating Point Adders, CAV 98

CCH$^+$96 Chen, Clarke, Ho, et al.: Verification of All Circuits in a Floating-Point Unit Using Word-Level Model Checking, FMCAD 96

CRSS94 Cyrluk, Rajan, Shankar, Srivas: Effective theorem proving for hardware verification. Theorem Proving in Circuit Design, 1994

EP97 Even, Paul: On the Design of IEEE Compliant Floating Point Units, Arith 97

Har97 Harrison: Floating-Point Verification in HOL Light: The Exponential Function, TR 428, University Cambridge Computer Lab, UK, 1997

Har99 Harrison: A Machine-Checked Theory of Floating Point Arithmetic, TPHOL 1999

Jac01 Jacobi: Formal Verification of a Theory of IEEE Rounding, TPHOL poster session 2001

Min95 Miner: Defining the IEEE-854 floating-point standard in PVS. NASA Technical Report, 1995

ML96 Miner, Leathrum: Verification of IEEE Compliant Subtractive Division Algorithms, FMCAD 96

MLK98 Moore, Lynch, Kaufmann: A Mechanically Checked Proof of the AMD5K86 Floating-Point Division Program, IEEE Trans Comp, Sept 98

MP00 Müller, Paul: Computer Architecture - Complexity and Correctness, Springer 2000

OZGS99 O'Leary, Zhao, Gerth, Seger: IA-64 Floating Point Operations and the IEEE Standard for Binary Floating-Point Arithmetic, Intel Technology Journal Q4, 1999

Rus97 Russinoff: Proof of IEEE Compliance of the AMD-K5 Floating Point Square Root Microcode, Formal Methods in System Design, 14:1, 1999

Rus98 Russinoff: A Mechanically Checked Proof of IEEE Compliance of the Floating Point Multiplication, Division, and Square Root Algorithms of the AMD-K7 TM Processor, LMS Journal of Computations and Mathematics, 1998

Rus00 Russinoff: Mechanical Verification of RTL: The Athlon Floating-Point Adder, FMCAD 00