Verifikation der VAMP-Floating Point Unit

Christoph Berg

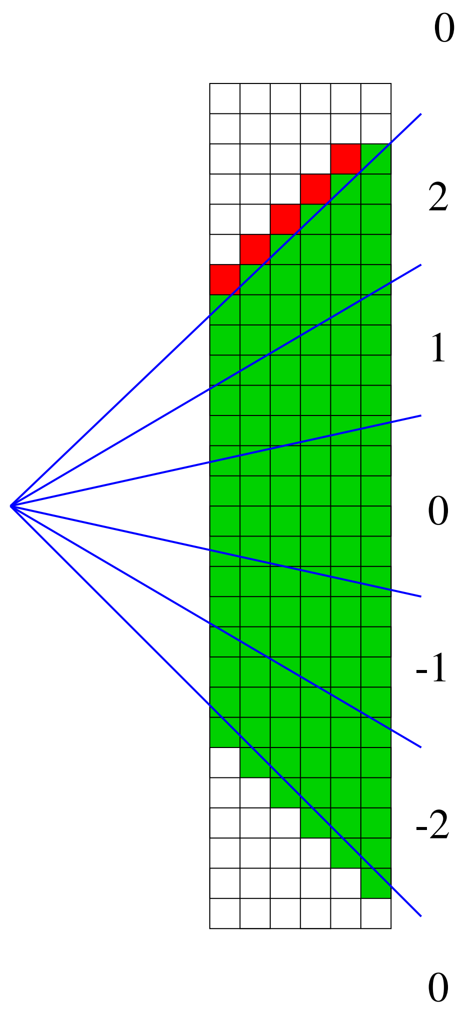
cb@cs.uni-sb.de

2. Juli 2001

Inhalt:

- Formale Verifikation, Beispiel
- VAMP
- IEEE-Arithmetik
- Addierer: Beweis
- Implementierung

Korrektheit von Hardware



- SRT-Division im Pentium I
- Tabelle wurde verifiziert
- Rote Einträge wurden „vergessen“ $\leadsto$ falsche Einträge oberhalb der Gerade
- Wichtig: Korrektheit **und** Abdeckung
- $\leadsto$ **ein** Tool

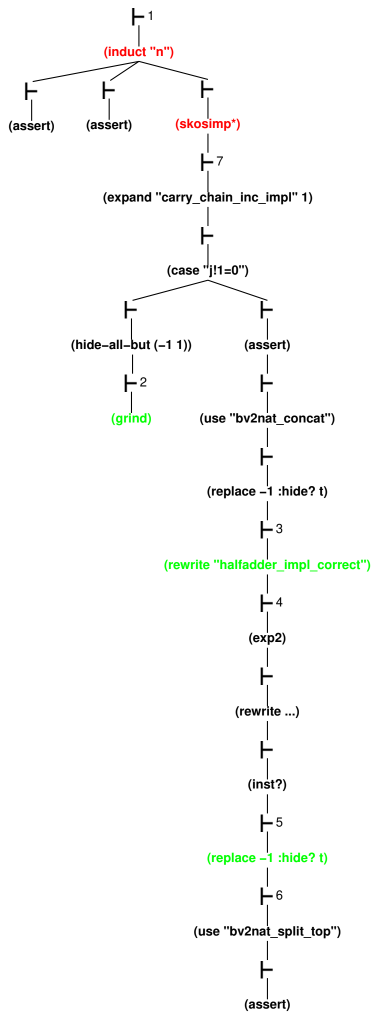
Formale Verifikation

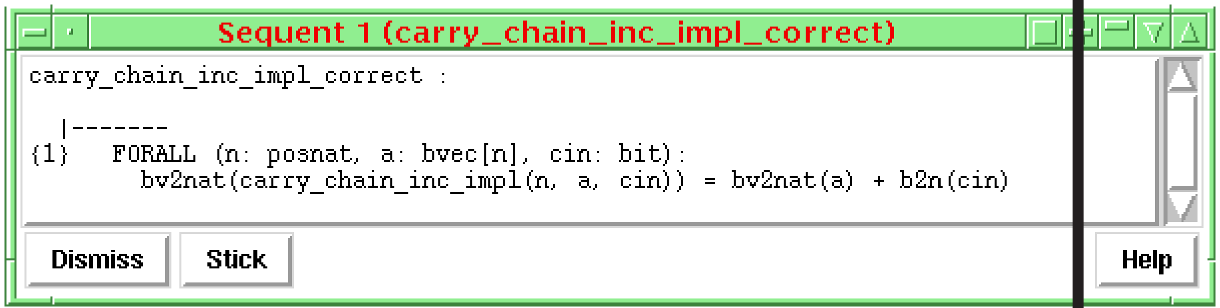
- Theorem Proving: PVS, ACL2, Isabelle, Nqthm, …
  - \+ Mächtige, allgemeine Beweistechniken
  - · automatisch oder interaktiv
  - – kompliziert
- Equivalence Checking: div. BDD-Ansätze, ACV
  - \+ vollautomatisch, einfach
  - – korrekte Referenz-Designs?
  - – Induktion schwierig
- Model Checking: SMV
  - \+ vollautomatisch, komplexe Aussagen über zeitliches Verhalten
  - – nur für (Kontroll-)Automaten
- $\leadsto$ Kombination aus den Verfahren wünschenswert

PVS: Carry-Chain-Incrementer

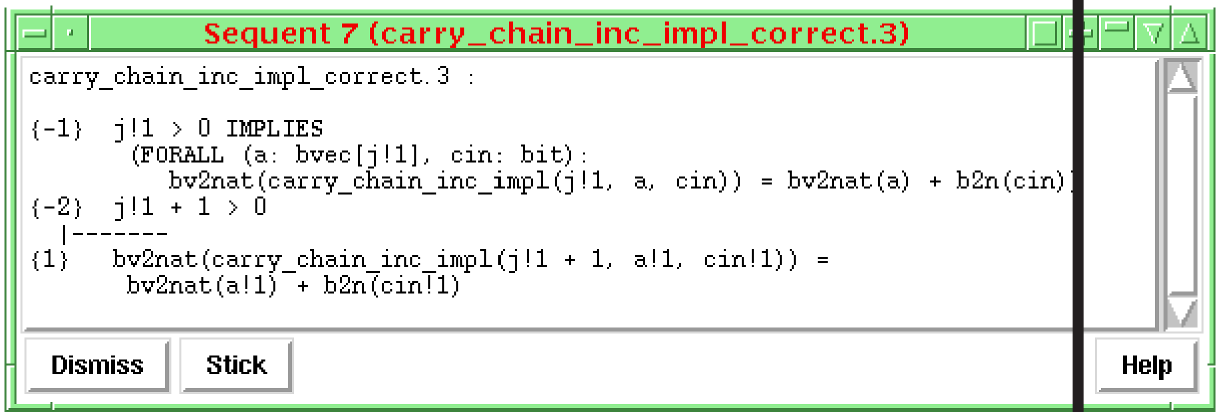
```
carry_chain_inc: Theory
Begin
  lib: Library = "../lib";
  Importing halfadder, lib@bv_lemmas

  carry_chain_inc_impl(n:posnat, a:bvec[n], cin:bit): Recursive bvec[n+1] =
    If n = 1 Then
      halfadder_impl(a(0), cin)
    Else
      Let chain = carry_chain_inc_impl(n-1, a^(n-2, 0), cin) in
        halfadder_impl(a(n-1), chain(n-1)) o chain^(n-2,0)
    Endif
    Measure n;

  carry_chain_inc_impl_correct: Lemma
    Forall (n: posnat, a: bvec[n], cin: bit):
      bv2nat(carry_chain_inc_impl(n, a, cin)) = bv2nat(a) + cin;

End carry_chain_inc
```

BEWEIS

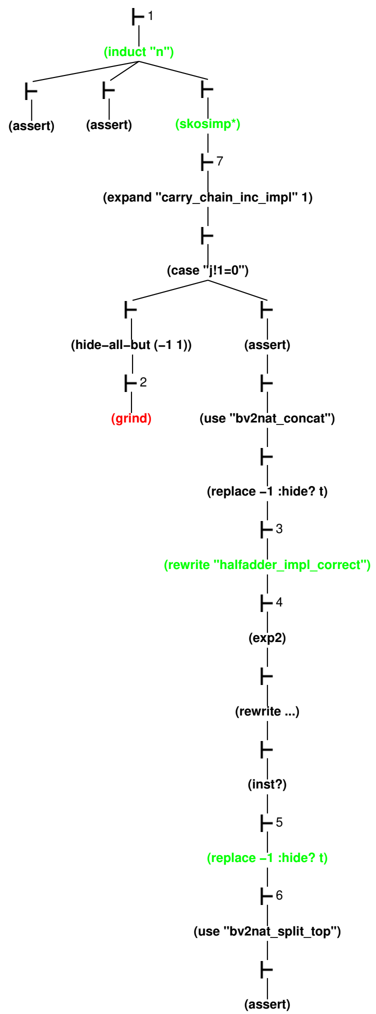


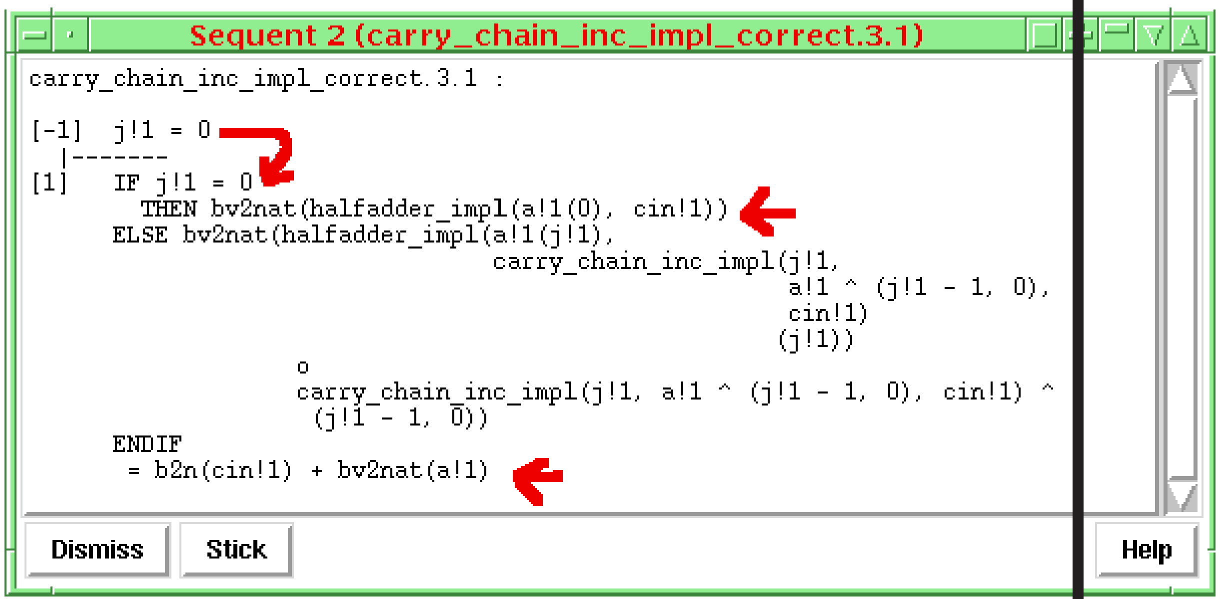


- Beweis per Induktion
- (induct ”n”)



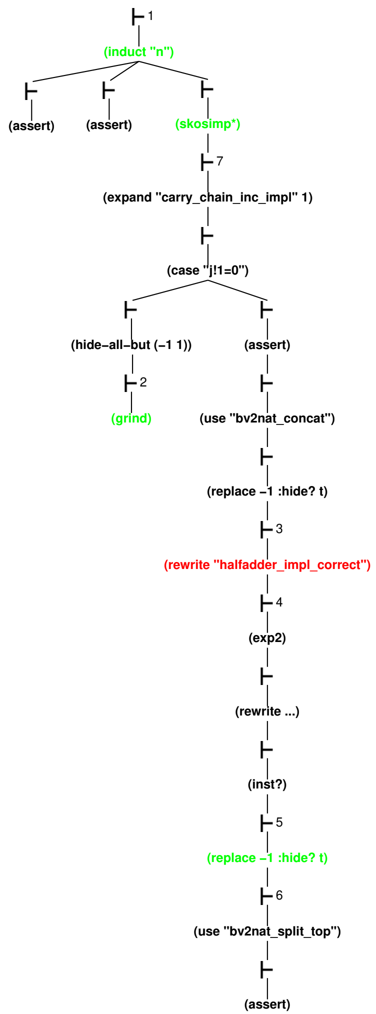
Beweis: Induktionsanfang

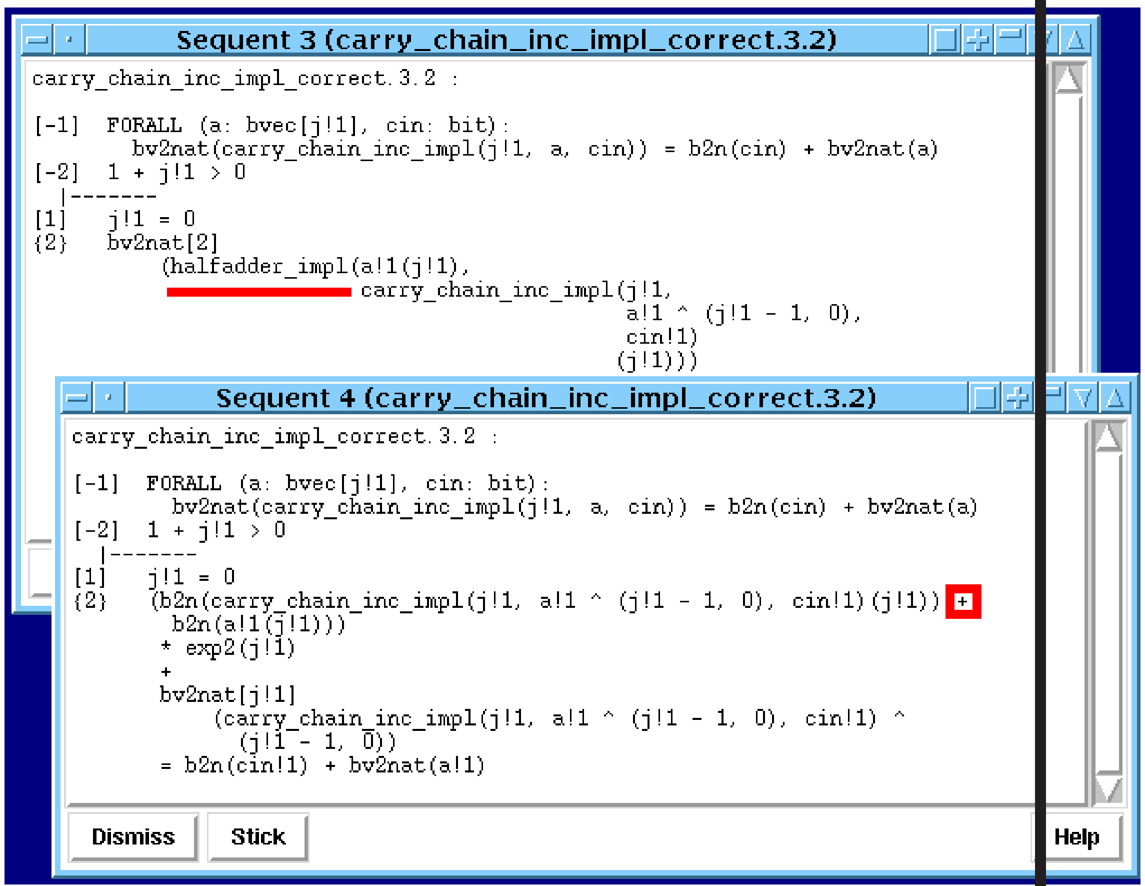




- wird automatisch gelöst
- (grind)

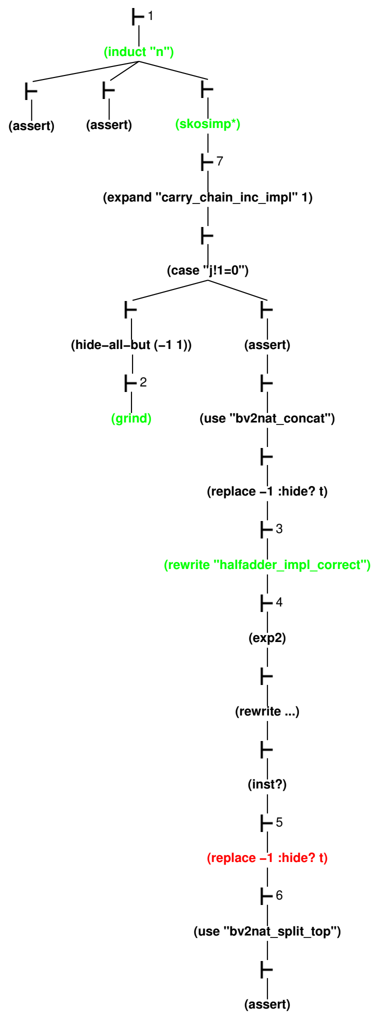
BEWEIS: HALBADDIERER

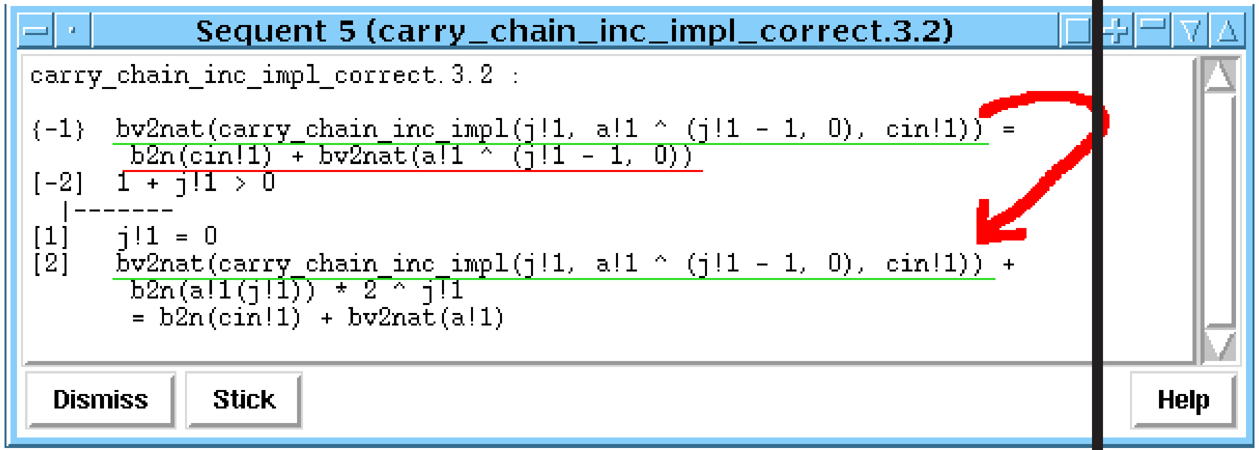




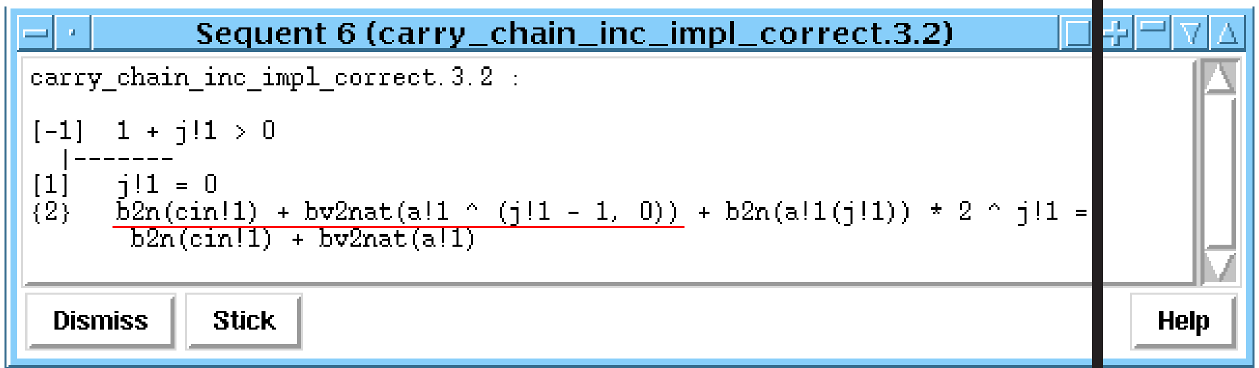
- halfadder_impl(a, b) = a + b

Beweis: Induktionsschritt



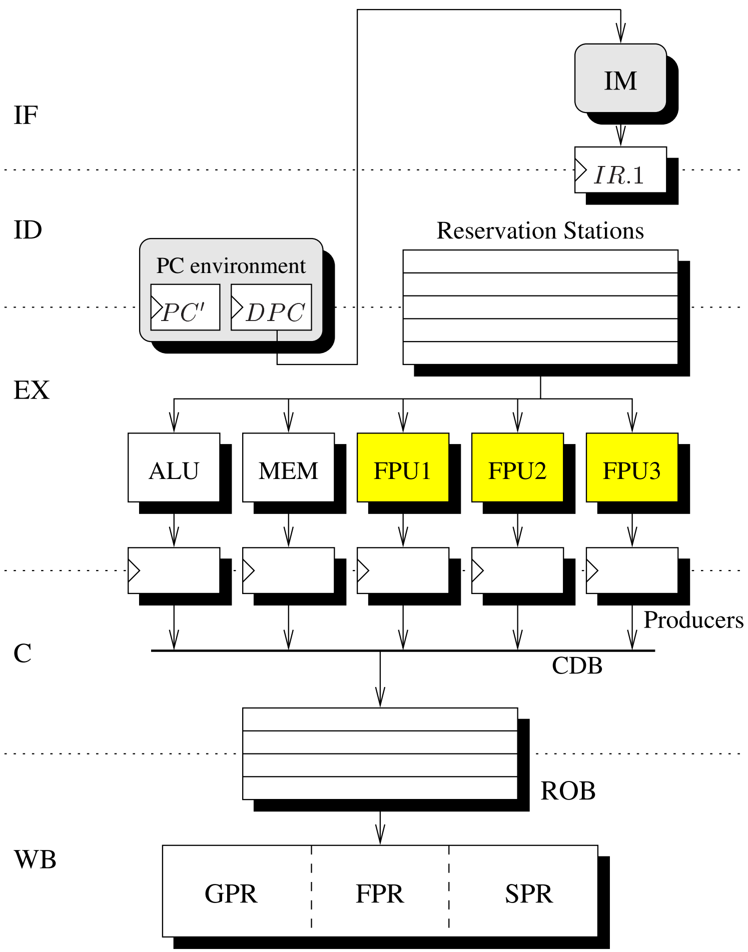


- Induktionsvoraussetzung
- (inst?), (replace -1 :hide? t)

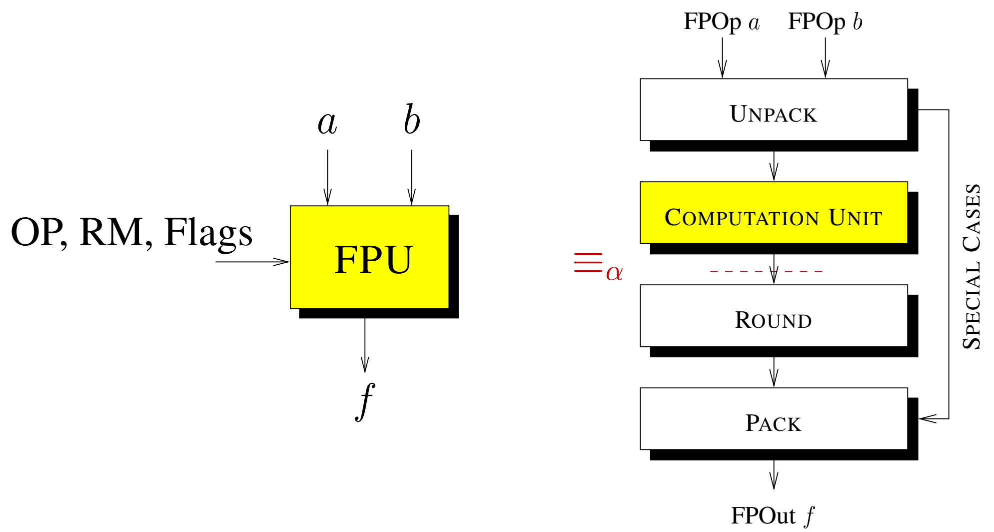


VAMP

- RISC-Prozessor:
  DLX/MIPS-Instruktionssatz,
  präsize Interrupts,
  Tomasulo-Scheduler, Cache
- FPU: $+-$, $\times\div$, Konversion,
  Vergleiche, Denormals
- CPU/FX: Daniel Kröning,
  FPU: Christian Jacobi/
  Christoph Berg,
  Memory: Sven Beyer,
  Software: Dirk Leinenbach,
  Carsten Meyer

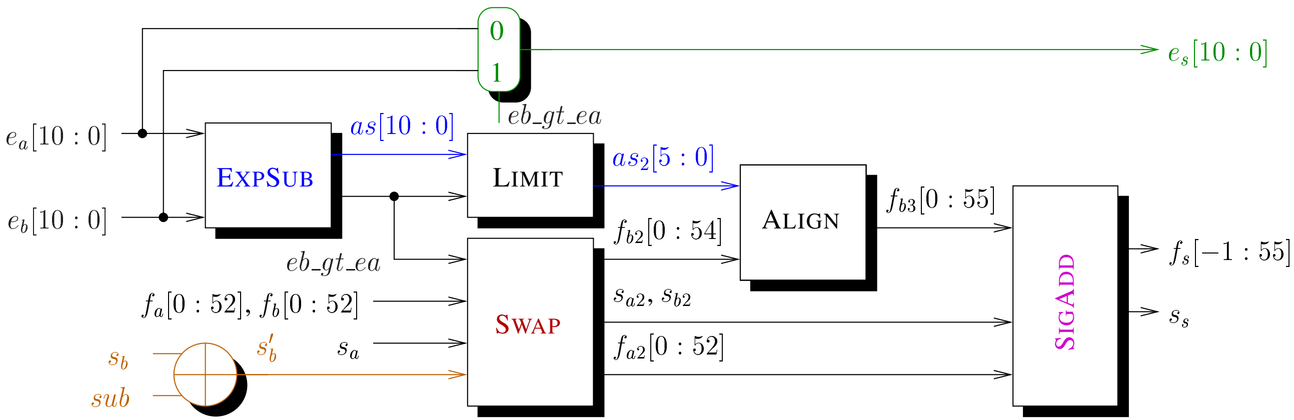


VAMP-FPU



- $+-$: (fast) kombinatorisch
- $\times\div$: Zyklus in der Newton-Raphson-Iteration
- Konversion: Spezialfall von Runden, Vergleiche: einfach
- Spezialfälle: extra Pfad
- Denormale Zahlen, Exceptions in Hardware

ADDIERER



- Subtraktion: $s'_b = s_b \oplus sub$
- $e_s = \max\{e_a, e_b\}$
- a und b tauschen, falls $e_b > e_a$
- $f_s \approx |f_a \pm 2^{-\delta} \cdot f_b|$, $\delta = |e_a - e_b|$

```
   100101001100
+ δ  110110111010
  1010001001111010
```

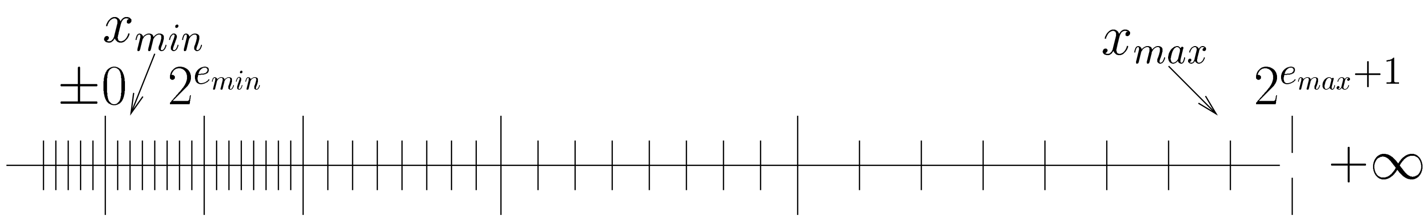
IEEE-ARITHMETIK (1)

- IEEE-Standard 754 (1985)

1	N	P-1
s	e	$.f$

- $e \in \{e_{min} - 1, e_{min}, \ldots, e_{max}\}$

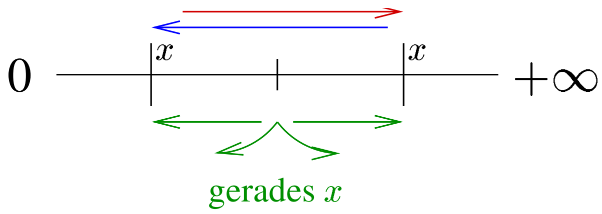
- Dargestellte Zahl: $\begin{cases} (-1)^s \cdot 2^e \cdot (1.f) & \text{für } e \geq e_{min} \\ (-1)^s \cdot 2^{e_{min}} \cdot (0.f) & \text{für } e = e_{min} - 1 \end{cases}$



- Single precision: $(N, P) = (8, 24)$, double: $(N, P) = (11, 53)$.

IEEE-ARITHMETIK (2)

- Rundungsmodi $rd_{\mathcal{M}}$: up, down, zero, nearest

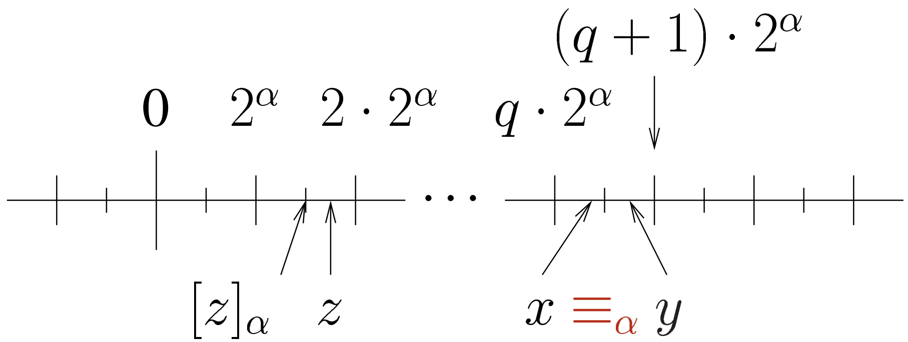


- werden zurückgeführt auf $\lceil\cdot\rceil$, $\lfloor\cdot\rfloor$ und Skalieren

- Korrektheit der FPU: Für alle $\circ, x, y, \mathcal{M}$:

$$x \circ_{FPU} y \;=\; rd_{\mathcal{M}}(x \circ y)$$

Abstraktion: α-Äquivalenz



- x und y sind α-äquivalent:

  $x \equiv_\alpha y \quad :\Longleftrightarrow \quad x = y \vee \big(\exists q \in \mathbb{Z}\colon q \cdot 2^\alpha < x, y < (q+1)\cdot 2^\alpha\big)$

- α-Repräsentant: $[x]_\alpha$

- Satz: $[f]_\alpha = f[k,\alpha] + 2^{\alpha-1} \cdot sticky_\alpha(f)$

- Satz: $rd_{\mathcal{M}}(x) = rd_{\mathcal{M}}([x]_{e-P})$

- Addierer: $(s_s, e_s, f_s) \equiv_{e-P} (s_a, e_a, f_a) \pm (s_b, e_b, f_b)$

Addierer: Algorithmus

Sei $e_a \geq e_b$, $f_{b2} := [2^{-\delta} \cdot f_b]_{-(P+1)}$. Es gilt:

$$2^{e_a} \cdot \big((-1)^{s_a} \cdot f_a + (-1)^{s_b} \cdot 2^{-\delta} \cdot f_b\big) \equiv_{e-P} 2^{e_a} \cdot \big((-1)^{s_a} \cdot f_a + (-1)^{s_b} \cdot f_{b2}\big).$$

Beweis: $\delta < 2$:

$$f'_b = [2^{-\delta} \cdot f_b]_{-(P+1)} = 2^{-\delta} \cdot f_b.$$

$\delta \geq 2$:

$$2^{-\delta} \cdot f_b \equiv_{-(P+1)} f'_b.$$

$$(-1)^{s_b} \cdot 2^{-\delta} \cdot f_b \equiv_{-(P+1)} (-1)^{s_b} \cdot f'_b.$$

$$(-1)^{s_b} \cdot 2^{e_a - \delta} \cdot f_b \equiv_{e_a-(P+1)} (-1)^{s_b} \cdot 2^{e_a} \cdot f'_b,$$

$$(-1)^{s_a} \cdot 2^{e_a} \cdot f_a + (-1)^{s_b} \cdot 2^{e_a - \delta} \cdot f_b \equiv_{e_a-(P+1)} (-1)^{s_a} \cdot 2^{e_a} \cdot f_a + (-1)^{s_b} \cdot 2^{e_a} \cdot f'_b.$$

$$(-1)^{s_a} \cdot 2^{e_a} \cdot f_a + (-1)^{s_b} \cdot 2^{e_a - \delta} \cdot f_b \equiv_{e-P} (-1)^{s_a} \cdot 2^{e_a} \cdot f_a + (-1)^{s_b} \cdot 2^{e_a} \cdot f'_b.$$

$\square$

Addierer: Beweis

- Hardware implementiert Algorithmus
- $f_{b2} = [2^{-\delta} \cdot f_b]_{-(P+1)}, \cdots$
- „Durchschieben“

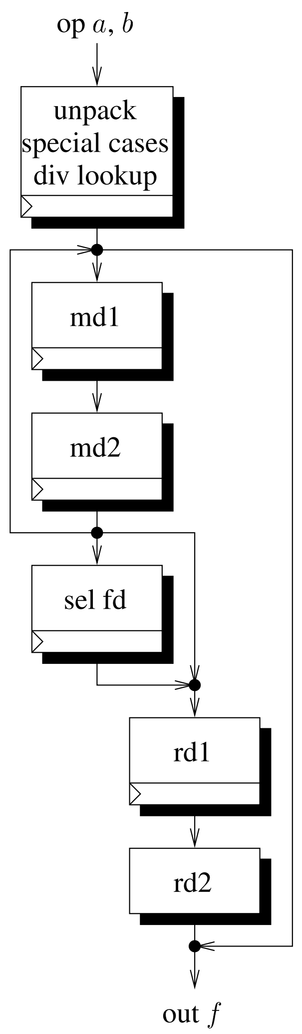
$$\begin{aligned}(s_a, e_a, f_a) \pm (s_b, e_b, f_b) &= (s_a, e_a, f_a) + (s_b', e_b, f_b)\\ &\equiv_{e-P} (s_{a2}, e_s, f_{a2}) + (s_{b2}, e_s, f_{b2})\\ &= (s_s, e_s, f_s)\end{aligned}$$

$\square$

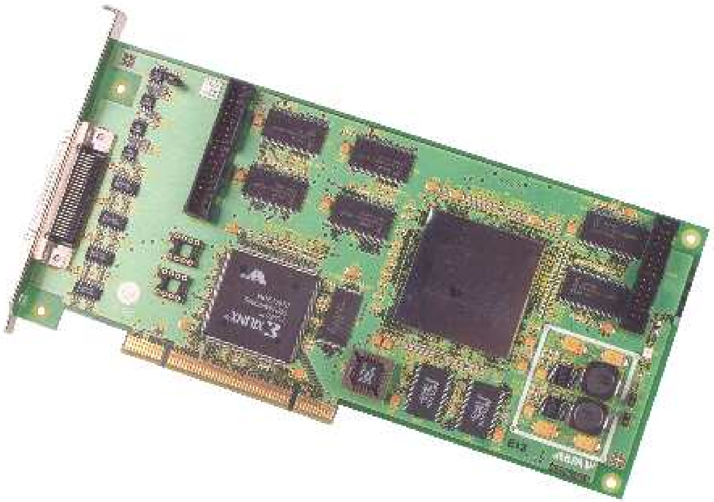
- $\leadsto$ Addierer erfüllt die Runder-Spezifikation

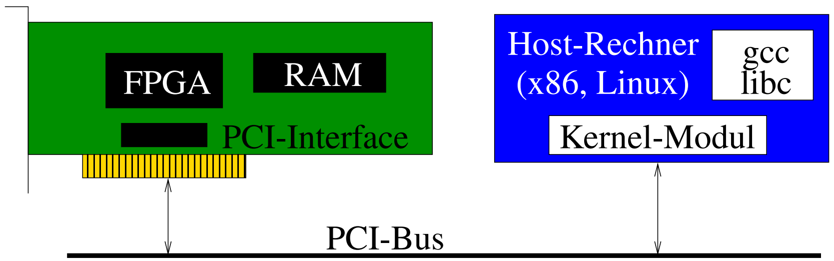
Multiplikation/Division

- Division:
  Newton-Raphson-Iteration
- Lebendigkeit, Korrektheit
- Verifikation in PVS schwierig
- Modelchecking: SMV
- PVS-Modelchecker



Implementierung





- Übersetzen der PVS-Funktionen nach Verilog HDL
- Synthese für ein Xilinx-FPGA
- Portieren von *gcc* und *libc*
- Benchmarks