Evaluating the hardware cost of the posit number system

FPL'19 – Barcelona

Yohann Uguen, Luc Forget, Florent de Dinechin
Univ Lyon, INSA Lyon, Inria, CITI

September 9, 2019

Motivation

Posit : new encoding scheme for real values

Motivation

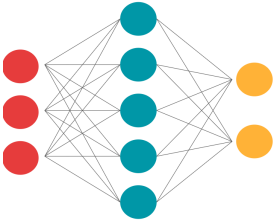
Posit : new encoding scheme for real values

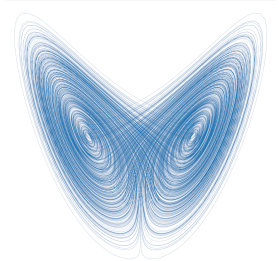
Posit claim : fewer bits, better results

Motivation

Posit : new encoding scheme for real values

Posit claim : fewer bits, better results

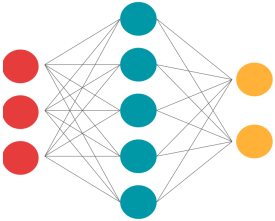


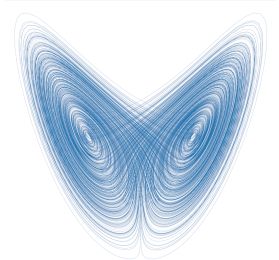


Motivation

Posit : new encoding scheme for real values

Posit claim : fewer bits, better results





How much does it cost ?

Floating point numbers

Floating point values consist in a value $1.F$ scaled by a power of two 2^E.

$$v = (-1)^s \times 1.F \times 2^E$$

Floating point numbers

Floating point values consist in a value $1.F$ scaled by a power of two 2^E.

$$v = (-1)^s \times 1.F \times 2^E$$

Encoding scheme	Max 2 power	Number of values $\in [1, 2[$
$W_E = 3$, $W_F = 4$: $s \mid e_2 \mid e_1 \mid e_0 \mid f_3 \mid f_2 \mid f_1 \mid f_0$	$2^7 = 128$	$2^4 = 16$
$W_E = 2$, $W_F = 5$: $s \mid e_1 \mid e_0 \mid f_4 \mid f_3 \mid f_2 \mid f_1 \mid f_0$	$2^3 = 8$	$2^5 = 32$

Floating point numbers dilemma

Trade-off between dynamic range and precision with the choice of W_E and W_F.

Floating point numbers dilemma

Trade-off between dynamic range and precision with the choice of W_E and W_F.

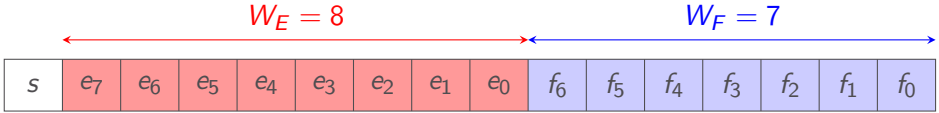
IEEE binary16 = FP<5, 10>

Floating point numbers dilemma

Trade-off between dynamic range and precision with the choice of W_E and W_F.

IEEE binary16 = FP<5, 10>

bfloat16 = FP<8,7>



Floating point numbers dilemma

Trade-off between dynamic range and precision with the choice of W_E and W_F.

IEEE binary16 = FP<5, 10>

$W_E = 5$ $W_F = 10$

s	e_4	e_3	e_2	e_1	e_0	f_9	f_8	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0

bfloat16 = FP<8,7>

$W_E = 8$ $W_F = 7$

s	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	f_6	f_5	f_4	f_3	f_2	f_1	f_0

DLFloat16 = FP<9, 6>

$W_E = 9$ $W_F = 6$

s	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	f_5	f_4	f_3	f_2	f_1	f_0

The posit encoding scheme – simple case

- Word size N
- Exponent: variable length sequence r of identical bits.
- Remaining bits: fraction bits

The posit encoding scheme – simple case

- Word size N
- Exponent: variable length sequence r of identical bits.
- Remaining bits: fraction bits

$r = 1$

0	1	0	1	0	0	0	1

$1.10001 \times 2^{1-1} = 1.53125$

Posit<8>

The posit encoding scheme – simple case

- Word size N
- Exponent: variable length sequence r of identical bits.
- Remaining bits: fraction bits

$r = 1$

0	1	0	1	0	0	0	1

$1.10001 \times 2^{1-1} = 1.53125$

Posit<8>

$r = 3$

0	1	1	1	0	0	0	1

$1.001 \times 2^{3-1} = 4.5$

The posit encoding scheme – simple case

- Word size N
- Exponent: variable length sequence r of identical bits.
- Remaining bits: fraction bits

$r = 1$

0	1	0	1	0	0	0	1

$1.10001 \times 2^{1-1} = 1.53125$

$r = 3$

0	1	1	1	0	0	0	1

$1.001 \times 2^{3-1} = 4.5$

$r = 5$

0	1	1	1	1	1	0	1

$1.1 \times 2^{5-1} = 24$

Posit<8>

The posit encoding scheme – simple case

- Word size N
- Exponent: variable length sequence r of identical bits.
- Remaining bits: fraction bits

$r = 1$

0	1	0	1	0	0	0	1

$1.10001 \times 2^{1-1} = 1.53125$

$r = 3$

0	1	1	1	0	0	0	1

$1.001 \times 2^{3-1} = 4.5$

Posit<8>

$r = 5$

0	1	1	1	1	1	0	1

$1.1 \times 2^{5-1} = 24$

$r = 7$

0	1	1	1	1	1	1	1

$1 \times 2^{7-1} = 64$

Posit simple case limitation

Bill Gates's fortune : $\approx 103.5 \times 10^9$\$

Posit simple case limitation

Bill Gates's fortune : $\approx 103.5 \times 10^9$\$

$$\text{Posit} < 32 > (103.5 \times 10^9) = 2^{30} \approx 1.1 \times 10^9$$

Posit simple case limitation

Bill Gates's fortune : $\approx 103.5 \times 10^9$\$

$$\text{Posit} < 32 > (103.5 \times 10^9) = 2^{30} \approx 1.1 \times 10^9$$

Increasing the range

- Shift the exponent of W_{ES} bits (scale by $2^{W_{ES}}$).
- Store exponent W_{ES} low bits before fraction bits.

Increasing the range

- Shift the exponent of W_{ES} bits (scale by $2^{W_{ES}}$).
- Store exponent W_{ES} low bits before fraction bits.

Posit<8,2>

$W_{ES} = 2$

0	1	0	1	1	0	0	1

$$1.001 \times 2^{0 \times 4 + 3} = 1.001 \times 2^3 = 9$$

Increasing the range

- Shift the exponent of W_{ES} bits (scale by $2^{W_{ES}}$).
- Store exponent W_{ES} low bits before fraction bits.

Posit<8,2>

$W_{ES} = 2$

0	1	0	1	1	0	0	1

$$1.001 \times 2^{0\times4+3} = 1.001 \times 2^3 = 9$$

0	1	1	1	0	0	0	1

$$1.1 \times 2^{2\times4+0} = 1.1 \times 2^8 = 385$$

Increasing the range

- Shift the exponent of W_{ES} bits (scale by $2^{W_{ES}}$).
- Store exponent W_{ES} low bits before fraction bits.

Posit<8,2>

$W_{ES} = 2$

0	1	0	1	1	0	0	1

$$1.001 \times 2^{0\times4+3} = 1.001 \times 2^3 = 9$$

0	1	1	1	0	0	0	1

$$1.1 \times 2^{2\times4+0} = 1.1 \times 2^8 = 385$$

0	1	1	1	1	1	1	1

$$1 \times 2^{6\times4+0} = 1 \times 2^{24} \approx 16 \times 10^6$$

Overview

Our goals:

- Evaluate the hardware cost of posits
- Compare this cost to standard FP hardware
- Provide an experimentation framework for posit hardware

gitlab.inria.fr/lforget/marto

Overview

Our goals:

- Evaluate the hardware cost of posits
- Compare this cost to standard FP hardware
- Provide an experimentation framework for posit hardware

Our tool Marto (Modern arithmetic tools):

- Open source HLS library for custom sized posit arithmetic
- Handling of Addition, Product, and *quire accumulation*

gitlab.inria.fr/lforget/marto

Marto usage example

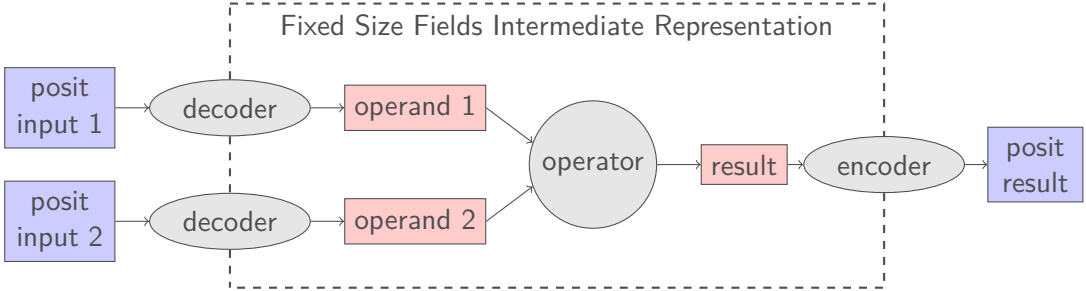
IEEE binary32 adder

```
#include "ieeefloats/ieee_dim.hpp"

// IEEENumber<WE, WF>

IEEENumber<8, 23> op1;
IEEENumber<8, 23> op2;
IEEENumber<8, 23> op3;

// Compute the IEEE sum
auto sum = op1 + op2 + op3;

// ...
```

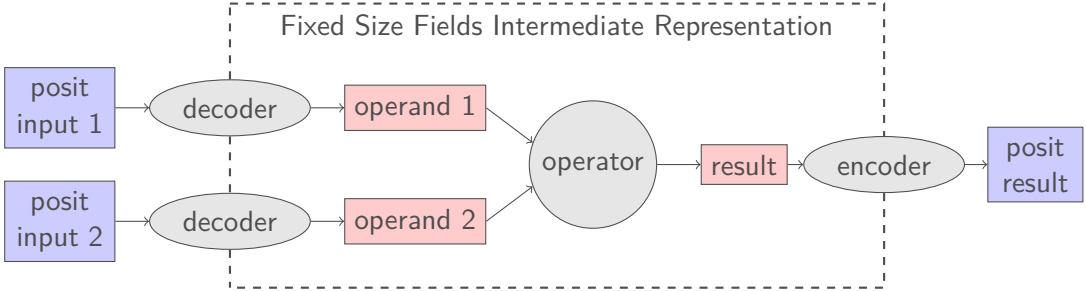
Posit 32,2 adder

```
#include "posit/posit_dim.hpp"

// PositNumber<N, WES>

PositNumber<32, 2> op1;
PositNumber<32, 2> op2;
PositNumber<32, 2> op3;

// compute the Posit(32,2) sum
auto sum = op1 + op2 + op3;

// ...
```

Variable-size fields are not hardware friendly



Variable-size fields are not hardware friendly



Which intermediate representation ?

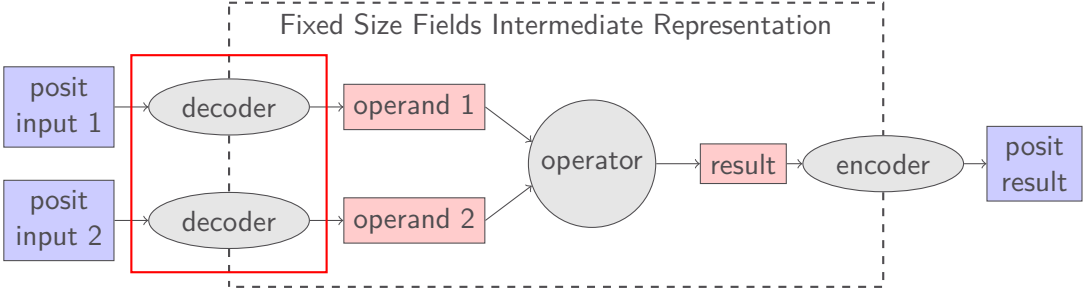
Posit Intermediate Format

Posit Intermediate Format (PIF) : the smallest floating point format to store any value of a Posit format.

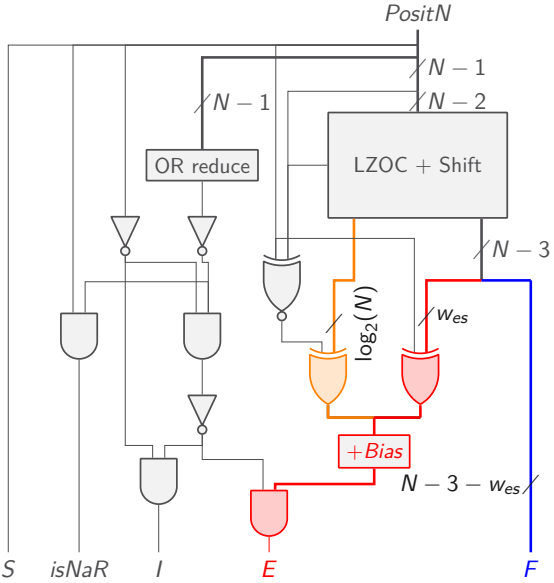
- Significand stored in 2's complement
- Extra bits for exact rounding (Round, Sticky)
- Extra bits for logic simplification (IsNaR, I)

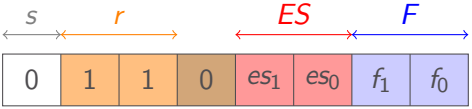
Format	W_E	W_F
Posit(8,0)	4	5
Posit(16, 1)	6	12
Posit(32, 2)	8	27
Posit(64, 3)	10	58

Posit decoder

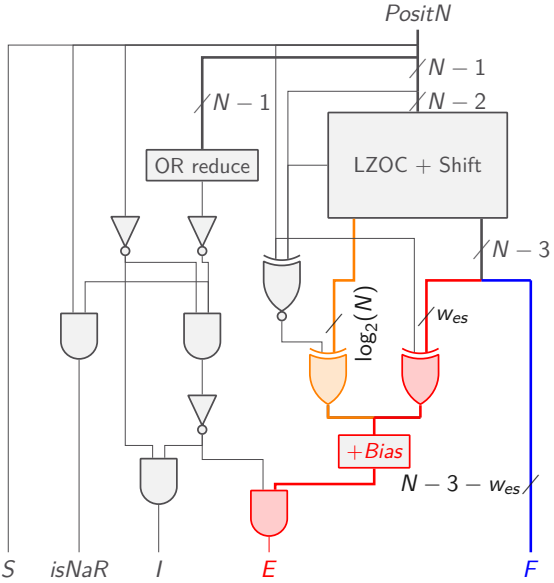


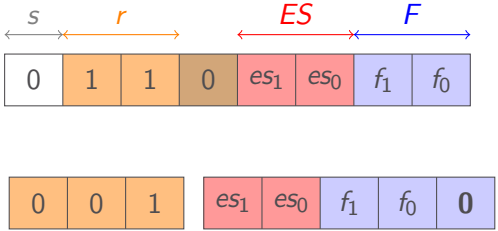
Posit decoder



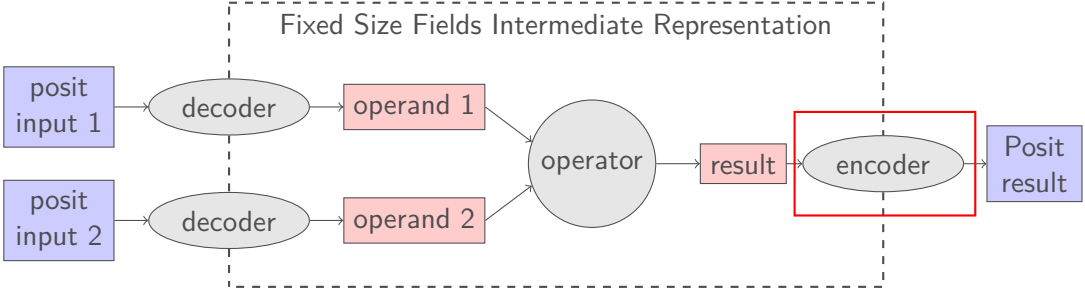


Posit decoder

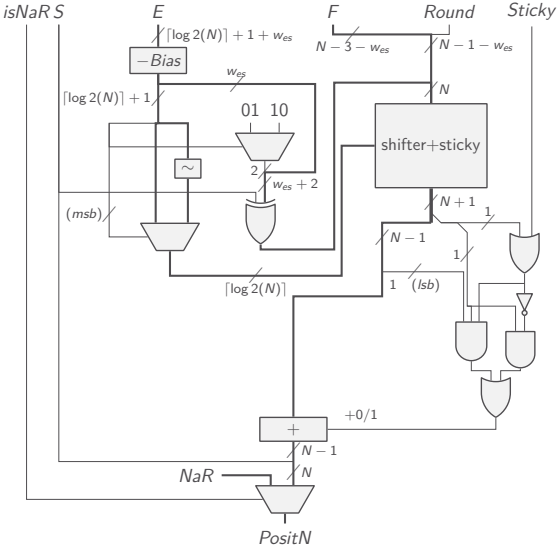




Posit encoder



Posit encoder



Posit addition comparison with state of the art

N	Design	LUTs	Delay (ns)
16	Chaurasiya *et al.*	**320**	23
	Jaiswal *et al.*	460	**21**
	Marto (this work)	**320**	**21**
32	Chaurasiya *et al.*	981	40
	Jaiswal *et al.*	1115	29
	Marto (this work)	**745**	**24**

Synthesis targets Zynq FPGA

- Chaurasiya *et al.* : *Parametrized Posit Arithmetic Hardware Generator* 2018
- Jaiswal *et al.* : *PACoGen: A Hardware Posit Arithmetic Core Generator* 2019

Posit product comparison with state of the art

N	Design	LUTs	delay (ns)	DSPs
16	Chaurasiya *et al.*	**218**	24	1
	Jaiswal *et al.*	271	19	1
	Marto (this work)	253	**18**	1
32	Chaurasiya *et al.*	572	33	4
	Jaiswal *et al.*	648	**27**	4
	Marto (this work)	**469**	**27**	4

Synthesis targets Zynq FPGA

- Chaurasiya *et al.* : *Parametrized Posit Arithmetic Hardware Generator* 2018
- Jaiswal *et al.* : *PACoGen: A Hardware Posit Arithmetic Core Generator* 2019

Comparison with floating point adder

N	format	LUTs	Regs.	cycles@333 MHz
16	Marto posit	447	371	17
	IEEE-754	216	205	12
32	Marto posit	999	975	23
	IEEE-754	425	375	14
	Xilinx float	341	467	9
64	Marto posit	1759	2785	36
	IEEE-754	918	792	17
	Xilinx double	641	1098	11

Synthesis targets Kintex 7

Posit product : ∼**2x slower**, requires ∼**2x more LUTs**

Comparison with floating point multiplier

N	Format	LUTs	Regs.	DSPs	cycles@333 MHz
16	Posit	269	292	1	16
	Soft FP<5, 10>	38	127	1	8
32	Posit	544	710	4	21
	Soft FP<8, 23>	67	228	2	9
	Xilinx Float	80	193	3	7
64	Posit	1501	2410	16	42
	Soft FP<11,52>	259	651	9	10
	Xilinx double	196	636	11	17

Targets Kintex 7

Posit product : ∼**2x slower**, requires ∼**8x more LUTs**

Comparison with floating point multiplier

N	Format	LUTs	Regs.	DSPs	cycles@333 MHz
16	Posit	269	292	1	16
	Soft FP<5, 10>	38	127	1	8
32	Posit	544	710	4	21
	Soft FP<8, 23>	67	228	2	9
	Xilinx Float	80	193	3	7
64	Posit	1501	2410	16	42
	Soft FP<11,52>	259	651	9	10
	Xilinx double	196	636	11	17

Targets Kintex 7

Posit product : ∼**2x slower**, requires ∼**8x more LUTs**

Conclusion

- Posit operators: slow and big compared to floating point operators
- What to do if you need more precision:
  - Custom word size allowed? Consider using custom sized floats
  - Or use the next standard float, it is still less expensive
  - Unless memory bandwith is the limit then maybe posit might help
- In both cases, our tool allows you to easily exploit state-of-the-art architectures

Future work

- Implement IEEE-754 multiplier
- Add multi HLS tool support using hint HLS integer library
- Revisit posit application level studies with custom floats

Future work

- Implement IEEE-754 multiplier
- Add multi HLS tool support using hint HLS integer library
- Revisit posit application level studies with custom floats

Thank you for your attention

gitlab.inria.fr/lforget/marto

The posit encoding scheme

A posit [1] encoding is parametrised by the word size N and the exponent shift size W_{es}. For a positive value, the code is made of the following fields :

- The first sign bit s is set to zero,
- The range encoded by the length r of a sequence of identical bits b,
- The exponent shift es on W_{es} bits,
- The remaining $N - (k + 2 + W_{es})$ bits are the significand bits f.

The encoded value is

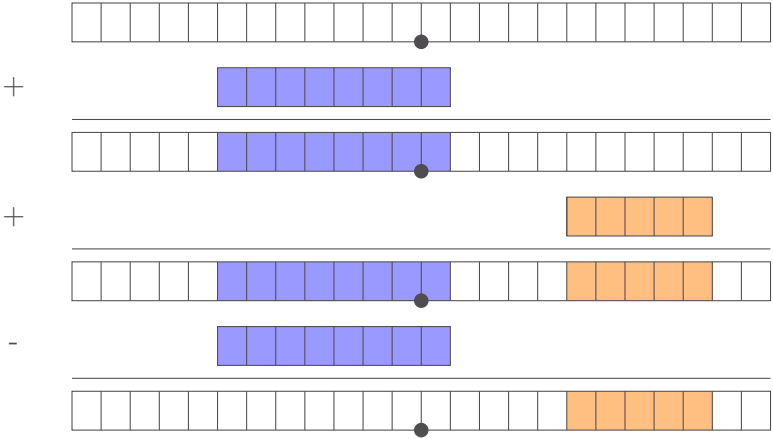
$$v = 1.f \cdot 2^{k2^{W_{es}} + es}$$

$$k = \begin{cases} -r & \text{if } b = 0 \\ r - 1 & \text{if } b = 1 \end{cases}$$

Negative values are encoded as 2's complement of their opposite

The posit quire

The quire is a very wide fixed precision value which is able to hold exactly any product of two posit values.



Posit arithmetic distinctive characteristics

- Only two "special codes" : 0 and NaR
- Only one zero
- Saturated arithmetic
- Mandatory Kulisch like exact accumulator
- Need a first running length computation to interpret correctly the encoded value

Posit intermediate representation format

Posit Intermediate Format (PIF) is a floating point representation with fixed size fields that is a super set of a given posit format.
For a given posit(N, W_{es}) format, the fields are :

- A "is NaR" flag,
- the sign bit s,
- the weight one bit I,
- the fraction part f of width $W_F = N - (W_{es} + 1)$,
- the biased exponent E of width $W_E = 1 + W_{es} + \lceil \log_2(N-2) \rceil$
- a round bit r and a sticky bit g to avoid double rounding

Encoded value (when not NaR):

$$v = (-2 \cdot s + I.fr) \times 2^{E - E_{min}}$$

Experiments

Exact accumulator

		LUTs	Regs.	DSPs	cycles	delay (ns)
Quire 16	U	1409	1763	1	1028	3.215
	S32	1239	1431	1	1031	2.643
	S64	1185	1555	1	1030	2.756
Quire 32 (512 bits)	U	5068	6256	4	1040	8.850
	S32	4394	4779	4	1055	2.854
	S64	3783	4564	4	1047	2.961
Kulisch 32 (559 bits)	S32	4446	5290	2	1050	2.875
	S64	4365	5276	2	1041	2.854