An FPGA-specific Approach to Floating-Point Accumulation and Sum-of-Products

ICFPT08, 7-10 December 2008

Florent de Dinechin[*], **Bogdan Pasca**[*], **Octavian Cret**[°], **Radu Tudoran**[°]
[*]projet Arénaire, ENS-Lyon/INRIA/CNRS/Université de Lyon, France
[°]Technical University of Cluj-Napoca, Romania

Outline

Context of this work

Proposed accumulator

Improved sum-of-products

Conclusions

Context of this work

Summing a large number of **floating-point** *terms fast and accurately*

Crucial for:

- **Scientific computations:**
  - dot-product, matrix-vector product, matrix-matrix product
  - numerical integration
- **Financial simulations:**
  - Monte-Carlo simulations
- ...

Not familiar with floating-point arithmetic?

Floating-Point(FP) numbers

Let x be a **normalized** binary FP number:

$$x = (-1)^S \times 1.f \times 2^e$$

where:

S - the **sign** of x

f - the **fraction** of x.

e - the **exponent** of x

Not familiar with floating-point arithmetic?

Floating-Point(FP) numbers

Let x be a **normalized** binary FP number:

$$x = (-1)^S \times 1.f \times 2^e$$

where:

S - the **sign** of x

f - the **fraction** of x.

e - the **exponent** of x

- e gives the dynamic range
  - IEEE-754 FP **double precision**, e_{min}=-1022 and $e_{max} = 1023$
- number of bits of f gives the **precision** p
  - IEEE-754 FP **double precision**, p=52

Not familiar with floating-point arithmetic?

Floating-Point(FP) numbers

Let x be a **normalized** binary FP number:

$$x = (-1)^S \times 1.f \times 2^e$$

where:

S - the **sign** of x

f - the **fraction** of x.

e - the **exponent** of x

- e gives the dynamic range
  - IEEE-754 FP **double precision**, e_{min}=-1022 and $e_{max} = 1023$
- number of bits of f gives the **precision** p
  - IEEE-754 FP **double precision**, p=52

Not familiar with floating-point arithmetic?

Floating-Point(FP) numbers

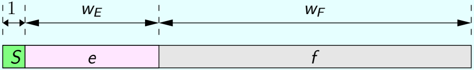
Let x be a **normalized** binary FP number:

$$x = (-1)^S \times 1.f \times 2^e$$

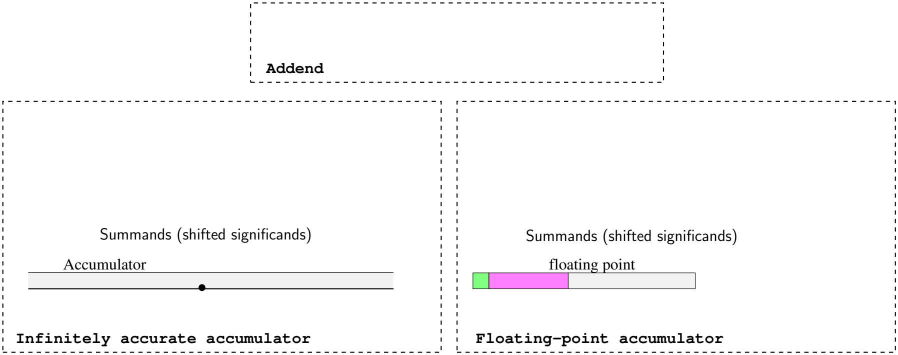
where:

- S - the **sign** of x
- f - the **fraction** of x.
- e - the **exponent** of x

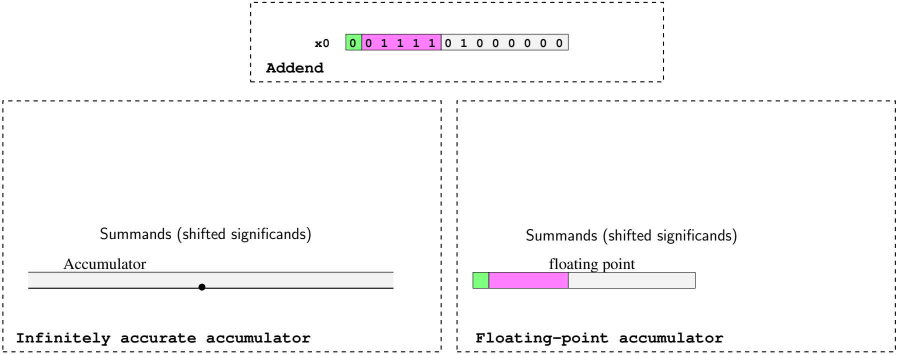
Graphical representation:



Accumulation



Accumulation



Accumulation

x0 0 0 1 1 1 1 0 1 0 0 0 0 0 0

Addend

x0 1 0 1 0 0 0 0 0 0

Summands (shifted significands)

Accumulator

1.0 1 0 0 0 0 0 0

Infinitely accurate accumulator

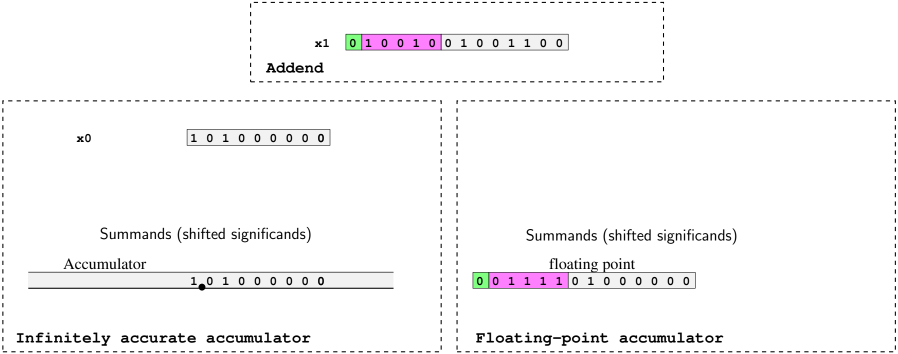
Summands (shifted significands)

floating point

0 0 1 1 1 1 0 1 0 0 0 0 0 0

Floating-point accumulator

Accumulation



Accumulation

x1 `0 1 0 0 1 0 0 1 0 0 1 1 0 0`

Addend

x0 `1 0 1 0 0 0 0 0 0`
x1 `1 0 1 0 0 1 1 0 0`

Summands (shifted significands)

Accumulator
`1 0 1 1.1 0 1 0 0 0 0 0`

Infinitely accurate accumulator

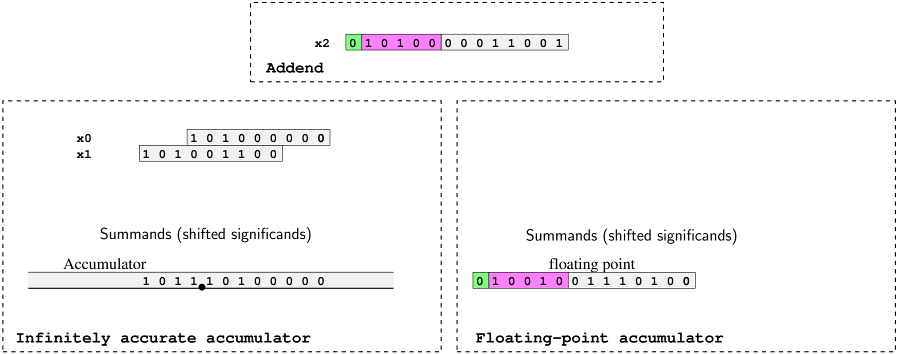
x1 `1 0 1 0 0 1 1 0 0`
acc `1 0 1 0 0 0 0 0 0`

Summands (shifted significands)

floating point
`0 1 0 0 1 0 0 1 1 1 0 1 0 0`

Floating-point accumulator

Accumulation



Accumulation

x2 0 1 0 1 0 0 0 0 0 1 1 0 0 1

Addend

```
x0            1 0 1 0 0 0 0 0 0
x1      1 0 1 0 0 1 1 0 0
x2  1 0 0 0 1 1 0 0 1
```

Summands (shifted significands)

Accumulator

1 0 1 1 1 0.1 1 0 0 0 0 0 0

Infinitely accurate accumulator

```
x2   1 0 0 0 1 1 0 0 1
acc      1 0 1 1 1 0 1 0 0
```

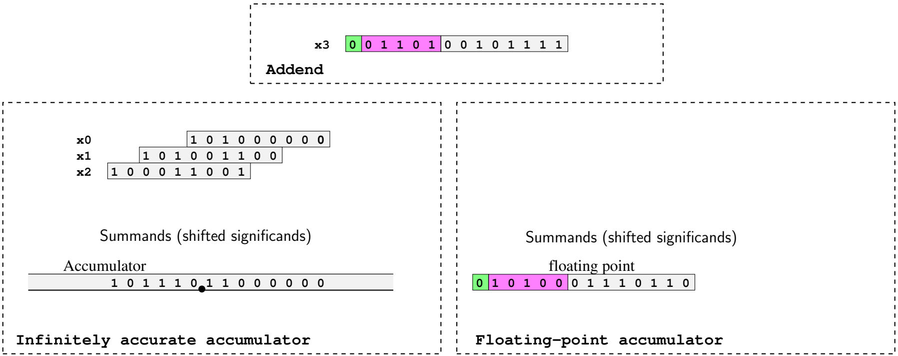
Summands (shifted significands)

floating point

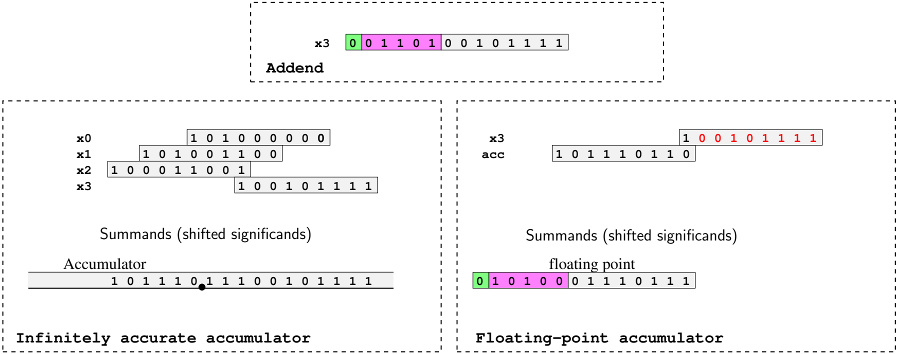
0 1 0 1 0 0 0 1 1 1 0 1 1 0

Floating-point accumulator

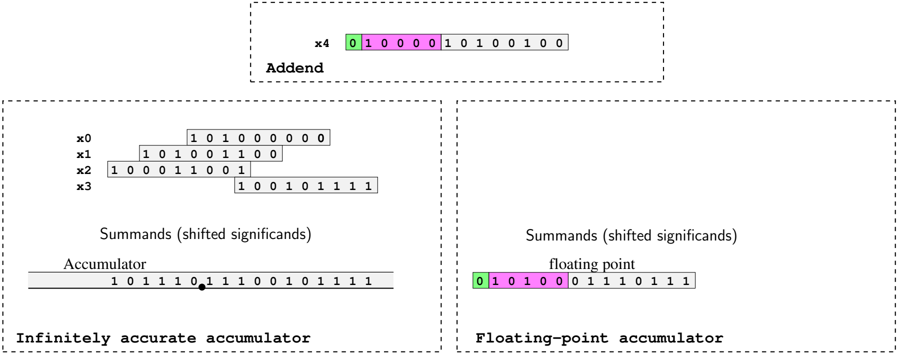
Accumulation



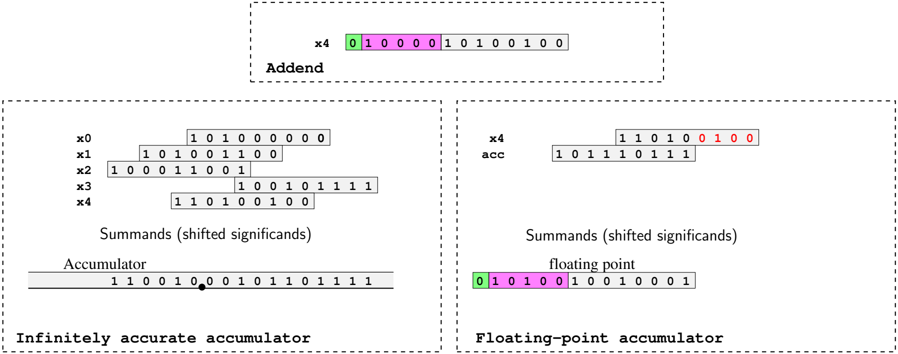
Accumulation



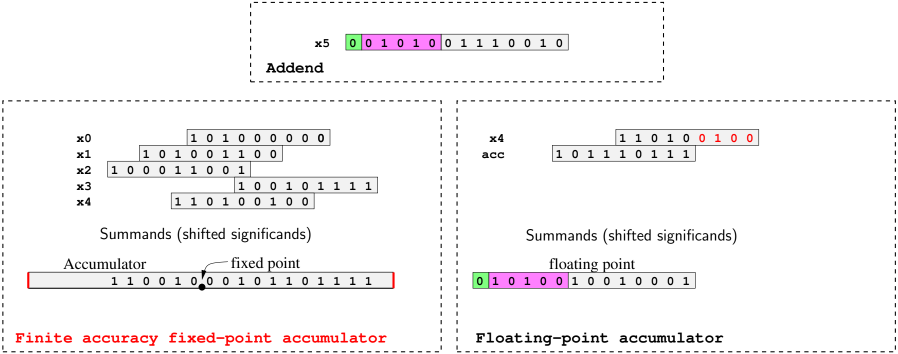
Accumulation



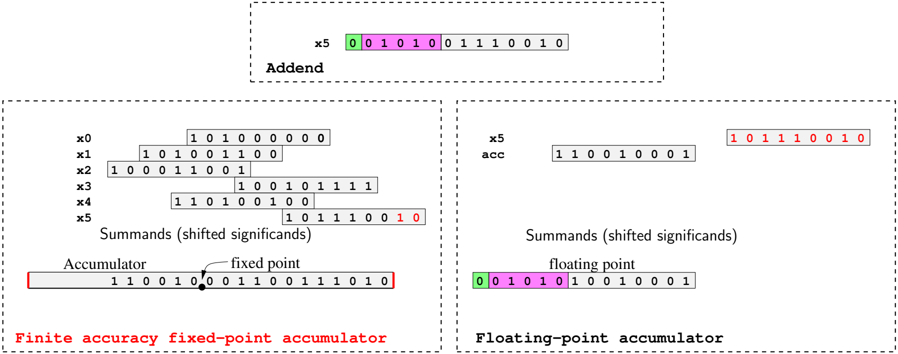
Accumulation



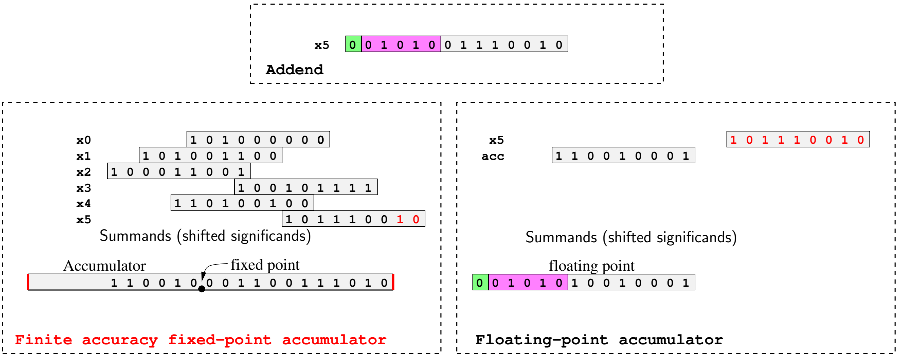
Accumulation



Accumulation



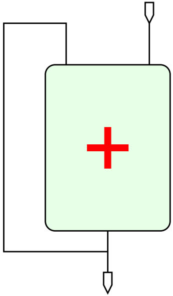
Accumulation



Accuracy:

Exact Result	=	50.2017822265625
FP Acc	=	50.125
Fixed-Point Acc	=	50.20166015625

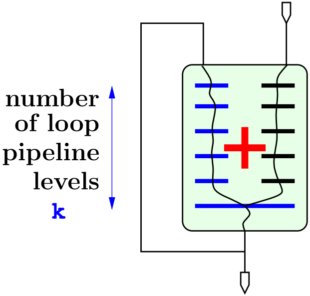
Closer look



Accumulator based on combinatorial floating-point adder

- very low frequency
- must pipeline for larger frequency

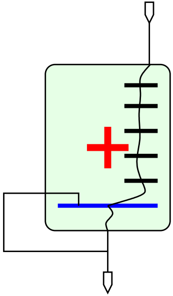
Closer look



Accumulator based on pipelined floating-point adder

- loop's critical path contains 2 shifters
- shifters are deeply pipelined
- produces k accumulation results
- these results have to be added somehow
  - adder tree
  - multiplexing mechanism on accumulation loop

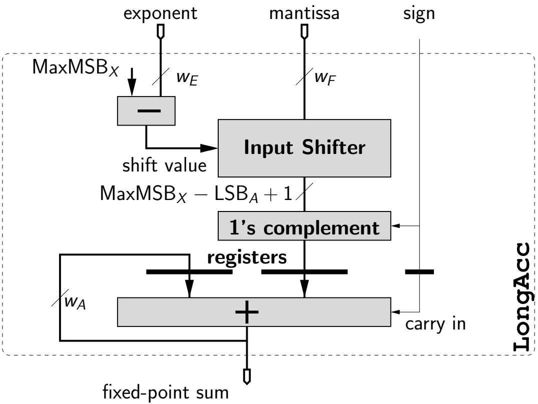
Closer look



Accumulator based on proposed long accumulator

- no shifts on the loop's critical path
- returns the result of the accumulation in fixed point
- the alignment shifter pipeline depth does not concern the result

Accumulator Architecture



- the sum is kept as a **large fixed-point number**
- one **alignment shift** (size depends on $MaxMSB_X$ and LSB_A)
- the loop's **critical path** contains a **fixed-point addition**
- fixed-point addition is fast on current FPGAs

Fast Accumulator Design

The accumulator should run at a target frequency

Fast Accumulator Design

The accumulator should run at a target frequency

- 64-bit addition works at 220MHz on Xilinx Virtex4 FPGA due to fast-carry chains
- still not enough ?
- use *partial carry-save representation*
  - cut large carry-propagation into chunks of k bits
  - critical path $=$ k-bit addition
  - small cost: $\lfloor width_{accumulator}/k \rfloor$ registers
- shifters can be arbitrarily pipelined for a given frequency

Fast Accumulator Design

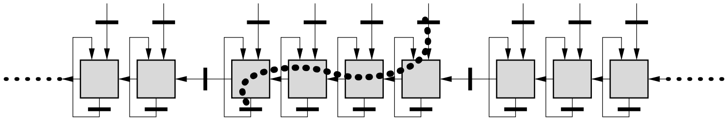
The accumulator should run at a target frequency

- 64-bit addition works at 220MHz on Xilinx Virtex4 FPGA due to fast-carry chains
- still not enough ?
- use *partial carry-save representation*
  - cut large carry-propagation into chunks of k bits
  - critical path $=$ k-bit addition
  - small cost: $\lfloor width_{accumulator}/k \rfloor$ registers
- shifters can be arbitrarily pipelined for a given frequency

Fast Accumulator Design

The accumulator should run at a target frequency

- 64-bit addition works at 220MHz on Xilinx Virtex4 FPGA due to fast-carry chains
- still not enough ?
- use *partial carry-save representation*
  - cut large carry-propagation into chunks of k bits
  - critical path $=$ k-bit addition
  - small cost: $\lfloor width_{accumulator}/k \rfloor$ registers

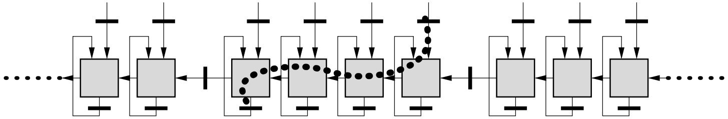


- shifters can be arbitrarily pipelined for a given frequency

Fast Accumulator Design

The accumulator should run at a target frequency

- 64-bit addition works at 220MHz on Xilinx Virtex4 FPGA due to fast-carry chains
- still not enough ?
- use *partial carry-save representation*
  - cut large carry-propagation into chunks of k bits
  - critical path $=$ k-bit addition
  - small cost: $\lfloor width_{accumulator}/k \rfloor$ registers



- shifters can be arbitrarily pipelined for a given frequency

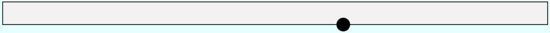
We advocate:

An **application tailored** fixed-point accumulator
for **floating-point inputs**

Ensuring that:

1. accumulator significand never needs to be shifted
2. it never overflows
3. provides a **result as accurate as the application requires**

Accumulator Parameters



MSB_A the weight of the MSB of the accumulator

- must to be larger than max. expected result

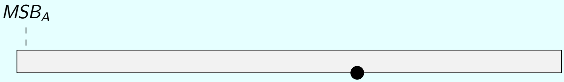
$MaxMSB_X$ the max. weight of the MSB of the summand

LSB_A weight of the LSB of the accumulator

- determines the final accumulation accuracy

The designer must provide values for these parameters.

Accumulator Parameters



MSB_A the weight of the MSB of the accumulator

- must to be larger than max. expected result

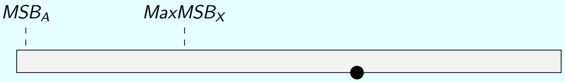
$MaxMSB_X$ the max. weight of the MSB of the summand

LSB_A weight of the LSB of the accumulator

- determines the final accumulation accuracy

The designer must provide values for these parameters.

Accumulator Parameters



MSB_A the weight of the MSB of the accumulator

- must to be larger than max. expected result

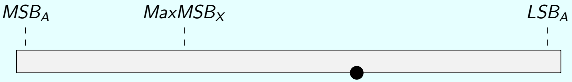
$MaxMSB_X$ the max. weight of the MSB of the summand

LSB_A weight of the LSB of the accumulator

- determines the final accumulation accuracy

The designer must provide values for these parameters.

Accumulator Parameters



MSB_A the weight of the MSB of the accumulator

- must to be larger than max. expected result

$MaxMSB_X$ the max. weight of the MSB of the summand

LSB_A weight of the LSB of the accumulator

- determines the final accumulation accuracy

The designer must provide values for these parameters.

Application Tailored

Application dictates parameter values

Application Tailored

Application dictates parameter values

Two possibilities:

- **software profiling** + safety margins
- **rough error analysis** + safety margins

Application Tailored

Application dictates parameter values

Two possibilities:

- **software profiling** + safety margins
- **rough error analysis** + safety margins

How to chose the parameters using the rough error analysis ?

MSB_A
- know an actual maximum + 10 bits safety margin
- consider the number of terms to sum

$MaxMSB_X$
- exploit input properties + safety margin
- worst case: $MaxMSB_X = MSB_A$

LSB_A **precision** vs. **performance**
- consider the desired final precision
- sum n terms, at most $\log_2 n$ bits are invalid

Application Tailored

Application dictates parameter values

Two possibilities:

- **software profiling** + safety margins
- **rough error analysis** + safety margins

How to chose the parameters using the rough error analysis ?

MSB_A
- know an actual maximum + 10 bits safety margin
- consider the number of terms to sum

$MaxMSB_X$
- exploit input properties + safety margin
- worst case: $MaxMSB_X = MSB_A$

LSB_A **precision** vs. **performance**
- consider the desired final precision
- sum n terms, at most $\log_2 n$ bits are invalid

Application Tailored

Application dictates parameter values

Two possibilities:

- **software profiling** + safety margins
- **rough error analysis** + safety margins

How to chose the parameters using the rough error analysis ?

MSB_A
- know an actual maximum + 10 bits safety margin
- consider the number of terms to sum

$MaxMSB_X$
- exploit input properties + safety margin
- worst case: $MaxMSB_X = MSB_A$

LSB_A **precision** vs. **performance**
- consider the desired final precision
- sum n terms, at most $\log_2 n$ bits are invalid

Application Tailored

Application dictates parameter values

Two possibilities:

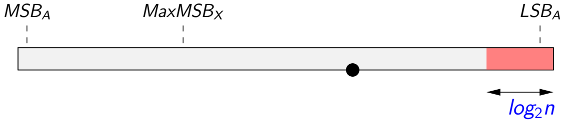
- **software profiling** + safety margins
- **rough error analysis** + safety margins

How to chose the parameters using the rough error analysis ?

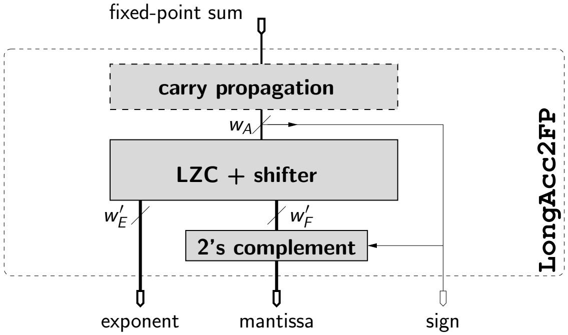
MSB_A
- know an actual maximum + 10 bits safety margin
- consider the number of terms to sum

$MaxMSB_X$
- exploit input properties + safety margin
- worst case: $MaxMSB_X = MSB_A$

LSB_A **precision** vs. **performance**
- consider the desired final precision
- sum n terms, at most $\log_2 n$ bits are invalid

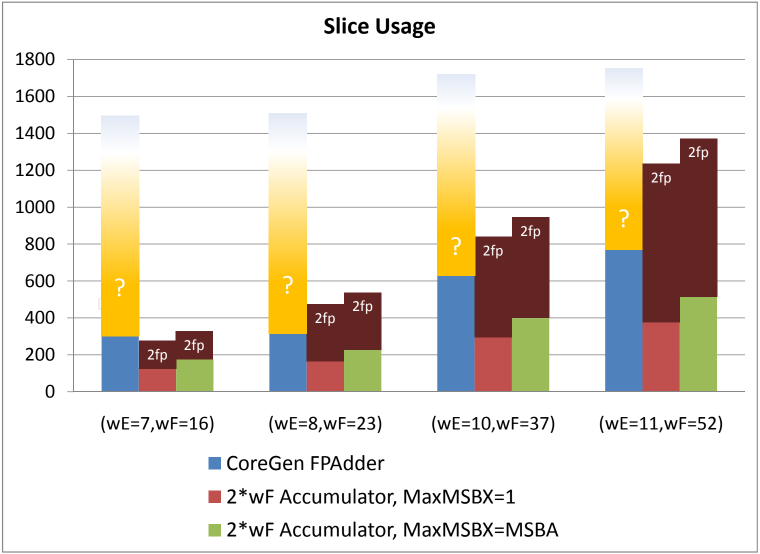


Post-normalization unit, or not

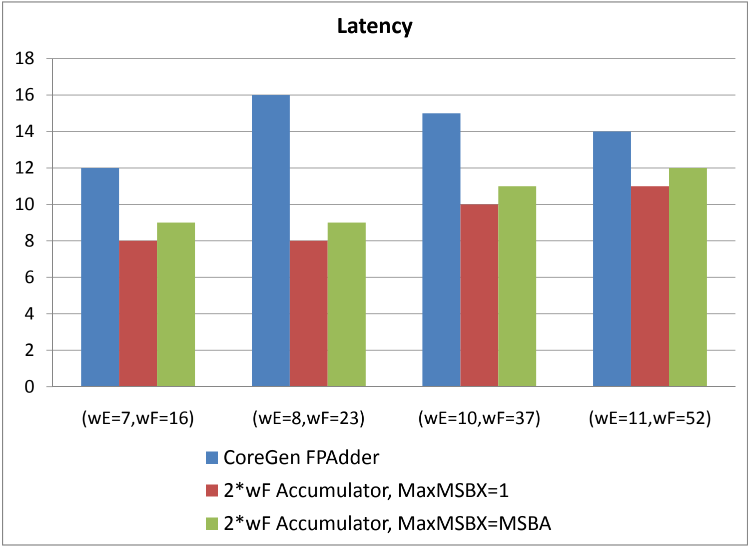


- converts fixed-point accumulator format to floating-point
- pipelined unit may be shared by several accumulators
- less useful:
  - many applications do not need the running sum
  - better to do conversion in software, use FPGA to accelerate the computation

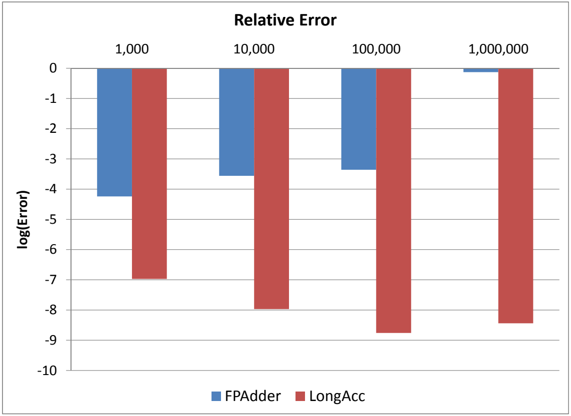
Performance results



Performance results



Relative error results



Accumulation of FP($w_E = 7, w_F = 16$) in unif. [0,1]

- LongAcc ($MSB_A = 20,\ LSB_A = -11$)

Accurate Sum-of-Products

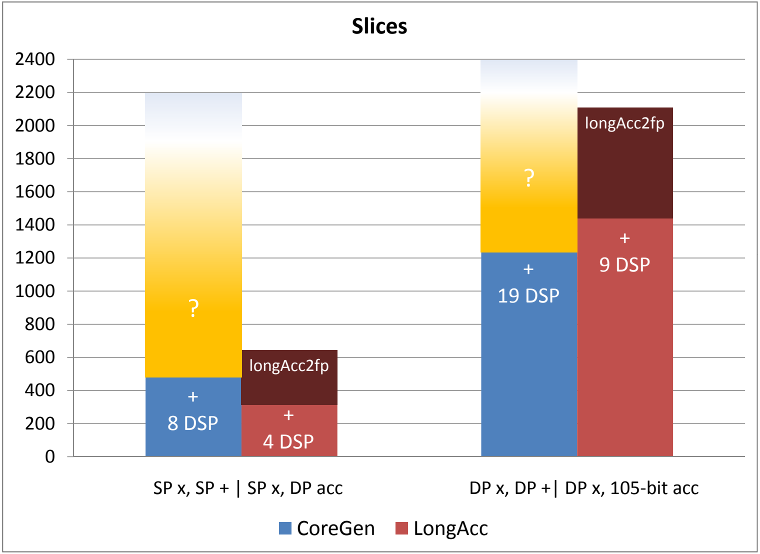
Ideea

Accumulate **exact** results of all multiplications

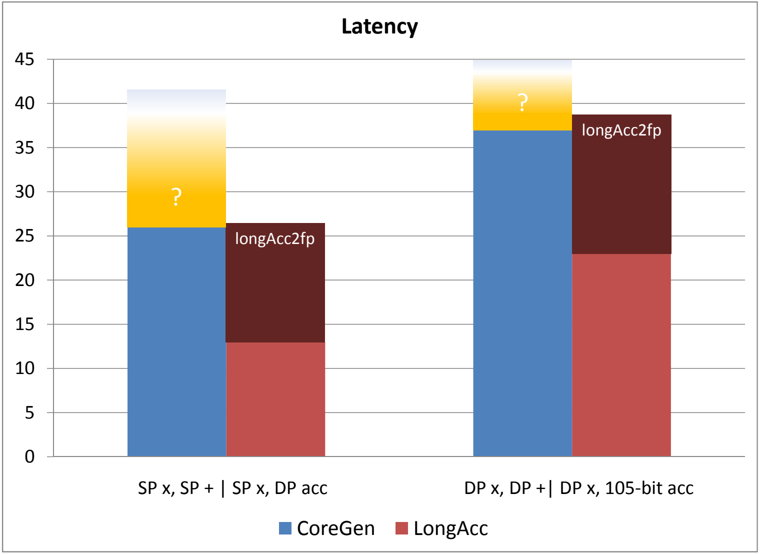
1. Use exact multipliers:
   - return all the bits of the exact product
   - contain no rounding logic
   - are cheaper to build
2. Feed the accumulator with exact multiplication results

Cost: Input shifter of accumulator is twice as large

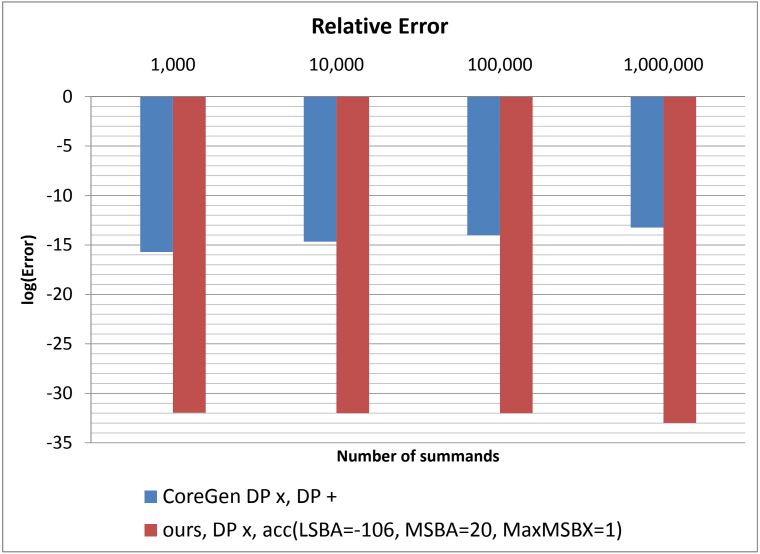
Operator Performance



Operator Performance



Operator Accuracy



Conclusion

- floating-point on FPGA should use the flexibility of the FPGA, not reimplement operators available in microprocessors
- faster and arbitrarily more accurate than a naive floating-point approach
- cost: designer-provided bounds on the accumulated values
- reward: improved performance + provably accurate accumulation
- available under GPL at: `http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/`

Thank you for your attention !

Thank you for your attention !
Questions ?

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=182341298371283 12.19238712398 7

    $\tilde{x}$=18234129837128312.19238712398 6

    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.19238712398 7

    $\tilde{x}$=0.19238712398 6

    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=18234129837128312.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=18234129837128312.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=18234129837128312.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=1823412983712831 2.192387123987

    $\tilde{x}$=1823412983712831 2.192387123986

    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987

    $\tilde{x}$=0.192387123986

    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=1823412983712831 2.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=1823412983712831 2.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$

Absolute and Relative Error

- measuring **only absolute error is not enough**
- the $error_{absolute} = |x - \tilde{x}|$
  - **example 1**
    x=1823412983712831 2.192387123987
    $\tilde{x}$=18234129837128312.192387123986
    $error_{absolute} = 1e - 12$
  - **example 2**
    x=0.192387123987
    $\tilde{x}$=0.192387123986
    $error_{absolute} = 1e - 12$
- better measure **relative error** (percentage error)
- $error_{rel} = \frac{\tilde{x} - x}{x}$
  - **example 1:** $error_{rel} = -5e - 29$
  - **example 2:** $error_{rel} = -5e - 12$