Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation

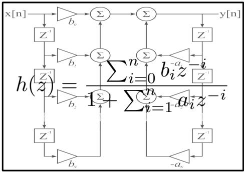
Benoit Lopez - Thibault Hilaire - Laurent-Stéphane Didier

DASIP 2012

October 24^{th} 2012

Outline

On the first hand... A filter



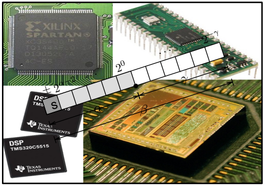
- Signal Processing
- LTI filters: FIR or IIR
- Its transfer function:

$$h(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$

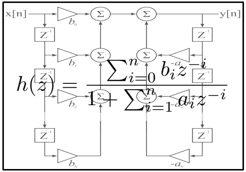
- Algorithmic relationship used to compute output(s) from input(s), for example:

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$$

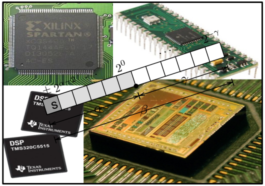
On the other hand... A target



- Hardware target (FPGA, ASIC) or software target (DSP, μC)
- Using fixed-point arithmetic for different reasons:
  - no FPU
  - cost
  - size
  - power consumption
  - etc.

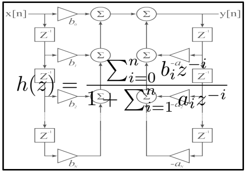


$$h(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$

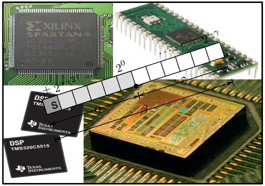


Need

Methodology and tools for the implementation of embedded filter algorithms in fixed-point arithmetic.



$$h(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$

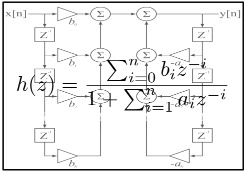


Need

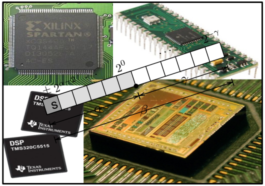
Methodology and tools for the implementation of embedded filter algorithms in fixed-point arithmetic.

A first (and basic) methodology

1. Given a filter, choose an algorithm
2. Round the coefficients in fixed-point arithmetic
3. Implement algorithm



$$h(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$



Need

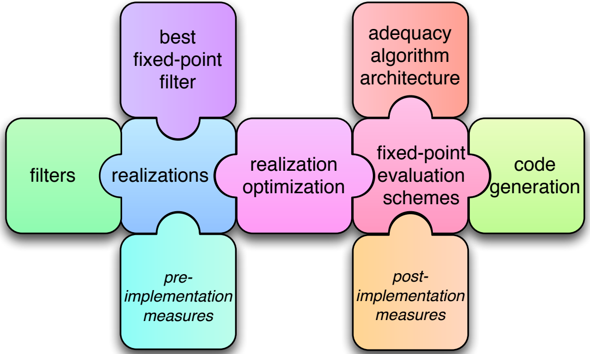
Methodology and tools for the implementation of embedded filter algorithms in fixed-point arithmetic.

A first (and basic) methodology

1. Given a filter, choose an algorithm
   - There are many possible realizations
2. Round the coefficients in fixed-point arithmetic
   - What format? Depends on the choice of algorithm
3. Implement algorithm
   - Is there only one possible implementation?

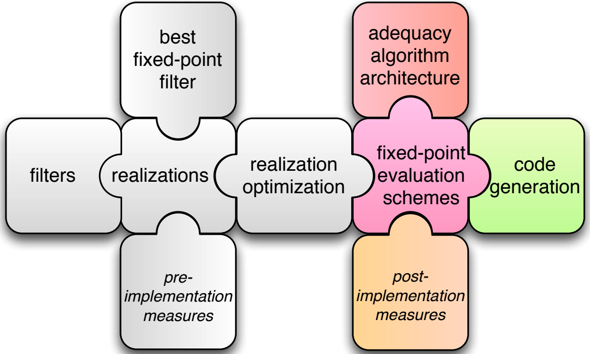
From filter to code

This work is a part of a global approach, transforming filters into fixed-point codes.



From filter to code

This work is a part of a global approach, transforming filters into fixed-point codes.



Objective:

Given an algorithm and a target, find the *optimal* implementation.

- model the fixed-point algorithms
- model the hardware resources (computational units, etc.)

Objective:

Given an algorithm and a target, find the *optimal* implementation.

- model the fixed-point algorithms
- model the hardware resources (computational units, etc.)
- evaluate the degradation
- find one/some *optimal* implemented algorithm(s)

Outline

Evaluation Scheme

The only operations needed in filter algorithm computation are sum-of-products:

$$S = \sum_{i=1}^{n} c_i \cdot x_i$$

where c_i are known constants and x_i variables (inputs, state or intermediate variables).

Evaluation Scheme

The only operations needed in filter algorithm computation are sum-of-products:

$$S = \sum_{i=1}^{n} c_i \cdot x_i$$

where c_i are known constants and x_i variables (inputs, state or intermediate variables).

Question:

How to implement S in fixed-point arithmetic?

SoP Example

Let H be the transfer function of a butterworth filter of $4^{th}-$order:

$$H(z) = \frac{0.00254078 + 0.01016312z^{-1} + 0.01524469z^{-2} + 0.01016312z^{-3} + 0.00254078z^{-4}}{1 - 2.64402372z^{-1} + 2.77901148z^{-2} - 1.34558515z^{-3} + 0.25124989z^{-4}}$$

SoP Example

Let H be the transfer function of a butterworth filter of $4^{th}-$order:

$$H(z) = \frac{0.00254078 + 0.01016312z^{-1} + 0.01524469z^{-2} + 0.01016312z^{-3} + 0.00254078z^{-4}}{1 - 2.64402372z^{-1} + 2.77901148z^{-2} - 1.34558515z^{-3} + 0.25124989z^{-4}}$$

Associated algorithm (Direct Form 1):

$$\begin{aligned} y(k) \quad = \quad & 0.00254078\ u(k) + 0.01016312z\ u(k-1) + 0.01524469z\ u(k-2) \\ & +0.01016312z\ u(k-3) + 0.00254078\ u(k-4) + 2.64402372\ y(k-1) \\ & -2.77901148\ y(k-2) + 1.34558515\ y(k-3) - 0.25124989\ y(k-4) \end{aligned}$$

SoP Example

Let H be the transfer function of a butterworth filter of $4^{th}-$order:

$$H(z) = \frac{0.00254078 + 0.01016312z^{-1} + 0.01524469z^{-2} + 0.01016312z^{-3} + 0.00254078z^{-4}}{1 - 2.64402372z^{-1} + 2.77901148z^{-2} - 1.34558515z^{-3} + 0.25124989z^{-4}}$$

Associated algorithm (integer part on 16 bits):

$$\begin{aligned} y(k) \quad = \quad & 21313.60743424 \times 2^{-23} \times u(k) + 21313.60743424 \times 2^{-21} \times u(k-1) \\ & +31970.43212288 \times 2^{-21} \times u(k-2) + 21313.60743424 \times 2^{-21} \times u(k-3) \\ & +21313.60743424 \times 2^{-23} \times u(k-4) + 21659.84231424 \times 2^{-13} \times y(k-1) \\ & -22765.66204416 \times 2^{-13} \times y(k-2) + 22046.0670976 \times 2^{-14} \times y(k-3) \\ & -16465.91279104 \times 2^{-16} \times y(k-4) \end{aligned}$$

SoP Example

Let H be the transfer function of a butterworth filter of 4^{th}−order:

$$H(z) = \frac{0.00254078 + 0.01016312z^{-1} + 0.01524469z^{-2} + 0.01016312z^{-3} + 0.00254078z^{-4}}{1 - 2.64402372z^{-1} + 2.77901148z^{-2} - 1.34558515z^{-3} + 0.25124989z^{-4}}$$

Associated algorithm (round to nearest integer):

$$\begin{aligned} y(k) \quad = \quad & 21314 \times 2^{-23} \times u(k) + 21314 \times 2^{-21} \times u(k-1) \\ & + 31970 \times 2^{-21} \times u(k-2) + 21314 \times 2^{-21} \times u(k-3) \\ & + 21314 \times 2^{-23} \times u(k-4) + 21660 \times 2^{-13} \times y(k-1) \\ & - 22766 \times 2^{-13} \times y(k-2) + 22046 \times 2^{-14} \times y(k-3) \\ & - 16466 \times 2^{-16} \times y(k-4) \end{aligned}$$

In software, addition is commutative but maybe not associative

On 4 bits

$$\begin{array}{ccc} 0.5 & =_2 & 0_\bullet 100 \\ 1.5 & =_2 & 01_\bullet 10 \\ 4 & =_2 & 0100_\bullet \end{array}$$

In software, addition is commutative but maybe not associative

On 4 bits

$$\begin{array}{ccc} 0.5 & =_2 & 0_\bullet 100 \\ 1.5 & =_2 & 01_\bullet 10 \\ 4 & =_2 & 0100_\bullet \end{array}$$

$4 + (0.5 + 1.5) =_2 010_\bullet 0 + 0100_\bullet =_2 0110_\bullet =_{10} 6$

$(4 + 0.5) + 1.5 =_2 0100_\bullet + 01_\bullet 10 =_2 0101_\bullet =_{10} 5$

In software, addition is commutative but maybe not associative

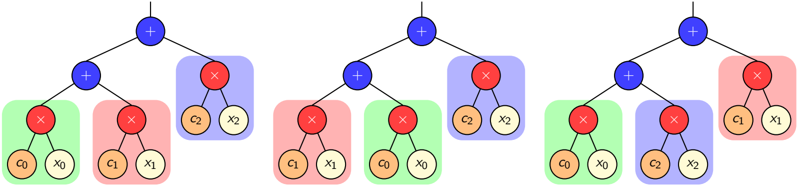
On 4 bits

$$\begin{aligned} 0.5 &=_2 0_\bullet 100 \\ 1.5 &=_2 01_\bullet 10 \\ 4 &=_2 0100_\bullet \end{aligned}$$

$4 + (0.5 + 1.5) =_2 010_\bullet 0 + 0100_\bullet =_2 0110_\bullet =_{10} 6$

$(4 + 0.5) + 1.5 =_2 0100_\bullet + 01_\bullet 10 =_2 0101_\bullet =_{10} 5$

Consider the order is important.



(a) (b) (c)

(a) = (b) but (a) $\neq$ (c)

oSoP

An evaluation scheme for a given SoP with a given order will be called ordered-Sum-of-Products (oSoP).

Number of oSoPs

For a given SoP of $N^{th}-$order, there are $\prod_{i=1}^{N-1}(2i-1)$ possible oSoPs to consider.

SoP Example

Let H be the transfer function of a butterworth filter of 4^{th}−order:

$$H(z) = \frac{0.00254078 + 0.01016312z^{-1} + 0.01524469z^{-2} + 0.01016312z^{-3} + 0.00254078z^{-4}}{1 - 2.64402372z^{-1} + 2.77901148z^{-2} - 1.34558515z^{-3} + 0.25124989z^{-4}}$$

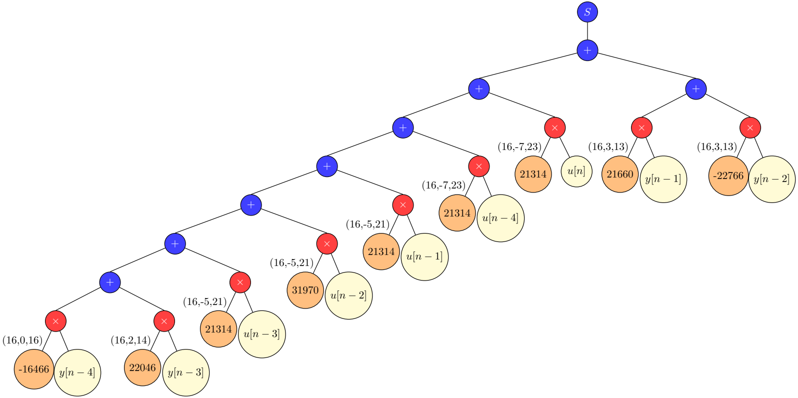
Associated algorithm:

$$\begin{aligned} y(k) \quad = \quad & 21314 \times 2^{-23} \times u(k) + 21314 \times 2^{-21} \times u(k-1) \\ & +31970 \times 2^{-21} \times u(k-2) + 21314 \times 2^{-21} \times u(k-3) \\ & +21314 \times 2^{-23} \times u(k-4) + 21660 \times 2^{-13} \times y(k-1) \\ & -22766 \times 2^{-13} \times y(k-2) + 22046 \times 2^{-14} \times y(k-3) \\ & -16466 \times 2^{-16} \times y(k-4) \end{aligned}$$

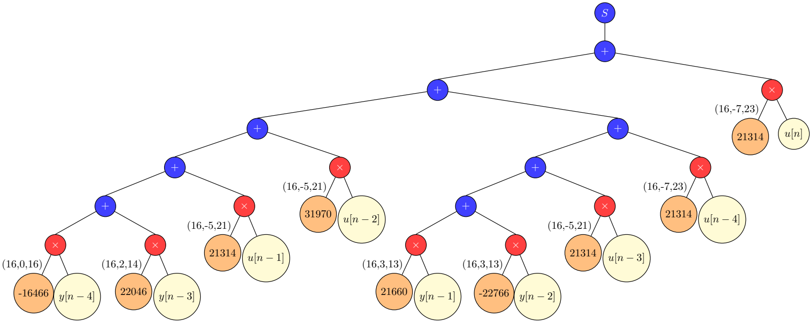
There are here 9 multiplications, so we have $\prod_{i=1}^{8}(2i-1) \simeq 2$ millions oSoPs to consider.

For example...

oSoP Example 1



oSoP Example 2



oSoP

An evaluation scheme for a given SoP with a given order will be called ordered-Sum-of-Products (oSoP).

Number of oSoPs

For a given SoP of $N^{th}-$order, there are $\prod_{i=1}^{N-1}(2i-1)$ possible oSoPs to consider.

Question: Which oSoP should we choose?

To do this, we developed a tool that:

- generates all possible oSoPs
- propagates fixed-point representation
- evaluates degradation errors

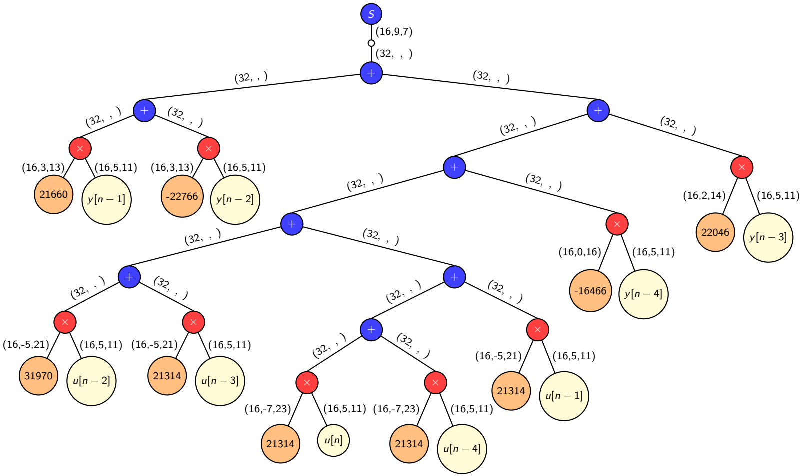
Fixed-Point propagation

FPR

Fixed-Point Representation (FPR) is defined as the tuple (wordlength, integer part, fractional part).

What are the given (input) values?

- FPR of constants c_i's (given by wordlength and value of c_i)
- FPR of variables x_i's
- Wordlength of adders and multipliers
- Final FPR

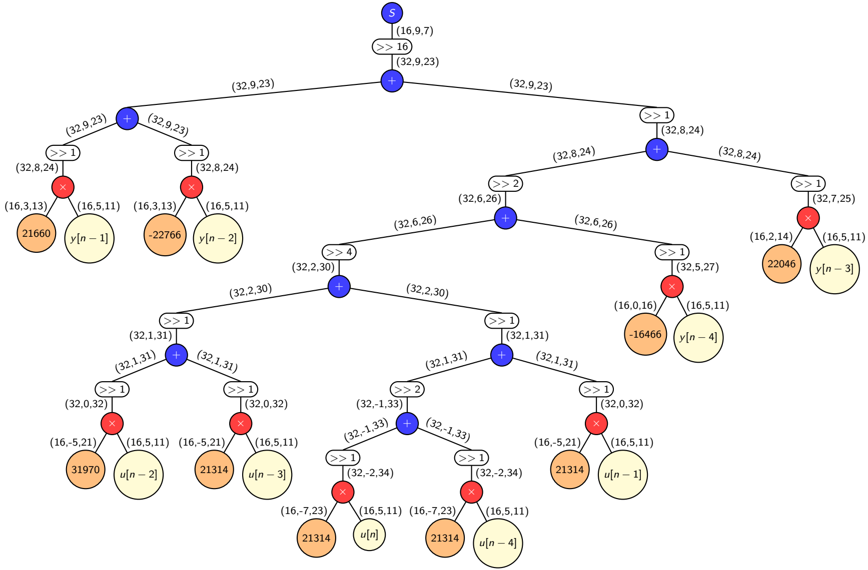

S
(16,9,7)
(32, ,)
21660
y[n − 1]
-22766
y[n − 2]
22046
y[n − 3]
-16466
y[n − 4]
31970
u[n − 2]
21314
u[n − 3]
21314
u[n]
21314
u[n − 4]
21314
u[n − 1]
(16,3,13)
(16,5,11)
(16,2,14)
(16,0,16)
(16,-5,21)
(16,-7,23)

Propagation Rules

From an oSoP parametrized with inputs FPR and wordlength, and using some propagation rules on adders and multipliers, we obtain a fully-parametrized oSoP.

Form the previous oSoP, we have the following fully-parametrized oSoP...

Propagation



Fixed-Point computational errors

Question:

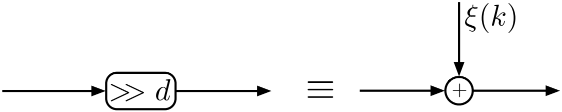
How to evaluate the numerical degradations?

Fixed-Point implementation implies numerical degradations, which depend on:

- the way the computations are organized
- the fixed-point representation of all the signals used in the computations
- and the fixed-point representation of each step of the operations

Noise

Usually in signal processing, we see degradation errors like additive white uniformly distributed noises.

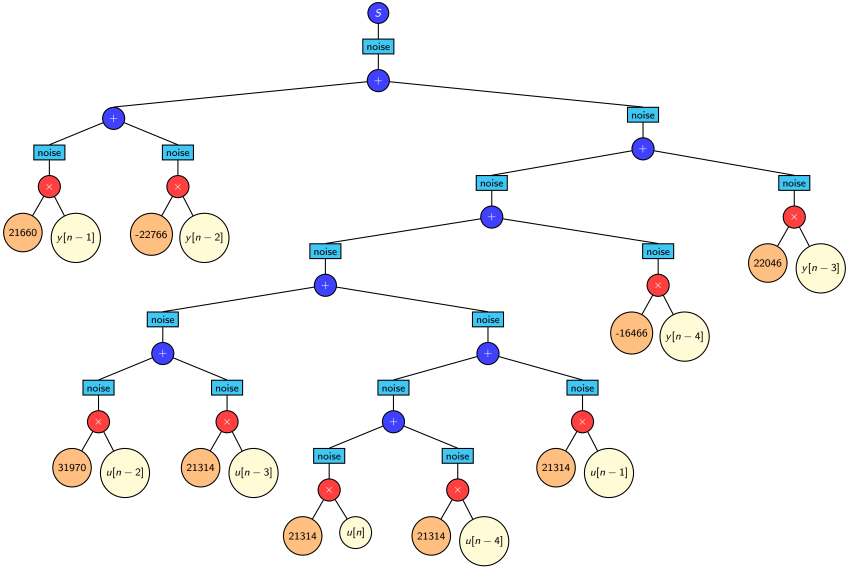


Right-shift of d bits:

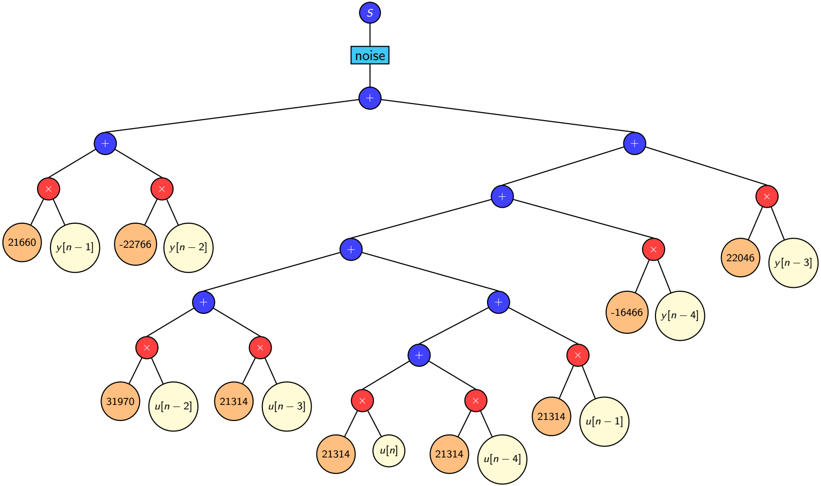
For a right-shift of d bits, first (μ) and second (σ) order moment are:

	Truncation	Best roundoff
μ	$2^{-\gamma-1}(1-2^{-d})$	$2^{-\gamma-d-1}$
σ^2	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$

Degradation errors



Degradation errors



Noise

- Different noises are added through the tree
- This cumulated noise can be viewed as output of a filter

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$$

$$y^{\dagger}(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y^{\dagger}(k-i) + \xi(k)$$

Noise

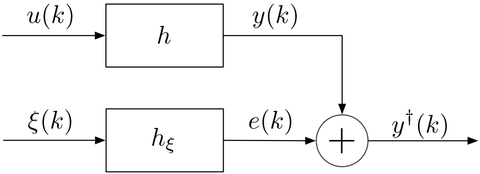
- Different noises are added through the tree
- This cumulated noise can be viewed as output of a filter

$$e(k) \triangleq y^{\dagger}(k) - y(k) = \xi(k) - \sum_{i=1}^{n} a_i e(k-i)$$

$$h_{\xi}(z) = \frac{1}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$

Noise

- Different noises are added through the tree
- This cumulated noise can be viewed as output of a filter



oSoP choice

Once we have evaluated degradations through our oSoPs, we want to choose the *best* one.

There are different criteria to choose oSoPs, like:

- noise
    - couple mean/variance

oSoP choice

Once we have evaluated degradations through our oSoPs, we want to choose the *best* one.

There are different criteria to choose oSoPs, like:

- noise
- latency (infinite parallelism)
  - height of the syntax tree
  - depending on the number of operators of the target

oSoP choice

Once we have evaluated degradations through our oSoPs, we want to choose the *best* one.

There are different criteria to choose oSoPs, like:

- noise
- latency (infinite parallelism)
- adequacy with hardware target
  - number of operators
  - wordlength of operators
  - *etc.*

oSoP choice

Once we have evaluated degradations through our oSoPs, we want to choose the *best* one.

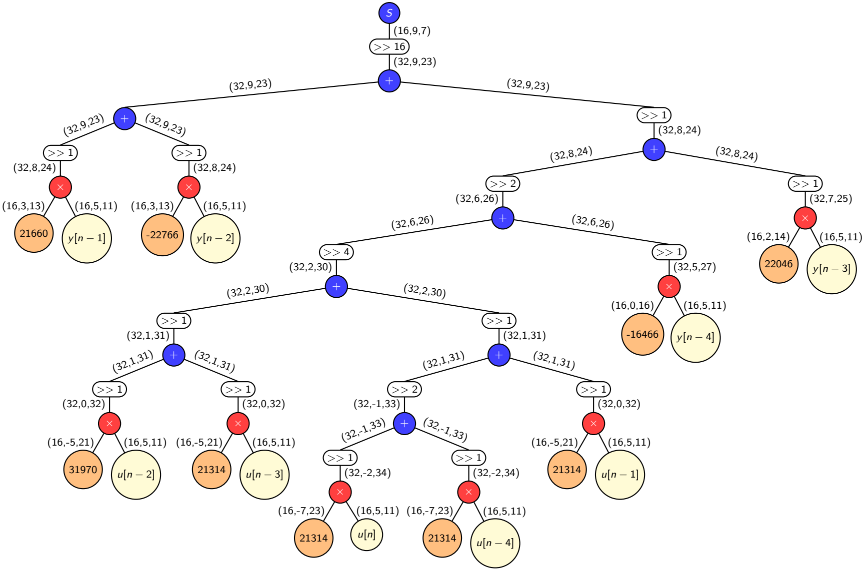
There are different criteria to choose oSoPs, like:

- noise
- latency (infinite parallelism)
- adequacy with hardware target
- *etc.*

Some options

Roundoff around multiplication

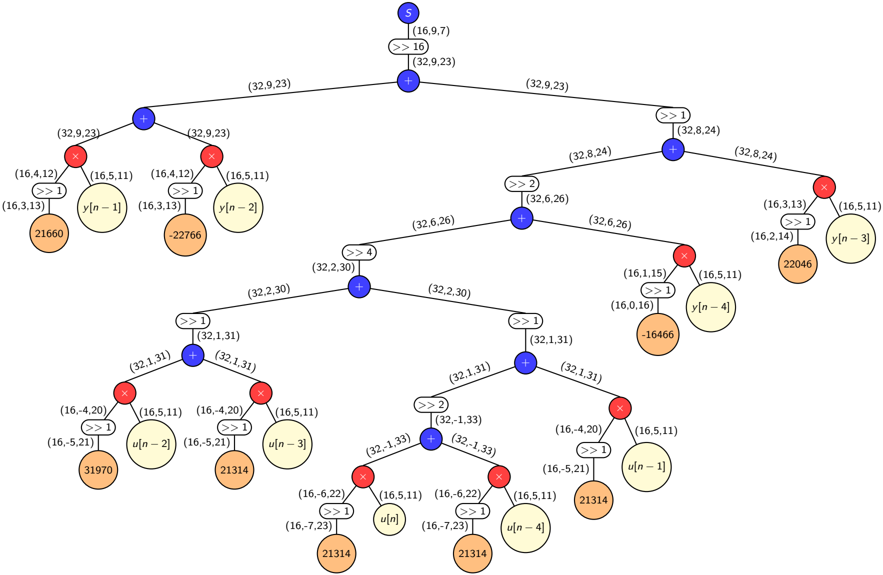
- Roundoff After Multiplication (RAM)


S
(16,9,7)
>> 16
(32,9,23)
+
(32,9,23)
(32,9,23)
+
(32,9,23)
(32,9,23)
>> 1
(32,8,24)
×
(16,3,13)
(16,5,11)
21660
y[n − 1]
>> 1
(32,8,24)
×
(16,3,13)
(16,5,11)
-22766
y[n − 2]
>> 1
(32,8,24)
+
(32,8,24)
(32,8,24)
>> 2
(32,6,26)
+
(32,6,26)
(32,6,26)
>> 1
(32,7,25)
×
(16,2,14)
(16,5,11)
22046
y[n − 3]
>> 4
(32,2,30)
+
(32,2,30)
(32,2,30)
>> 1
(32,5,27)
×
(16,0,16)
(16,5,11)
-16466
y[n − 4]
>> 1
(32,1,31)
+
(32,1,31)
(32,1,31)
>> 1
(32,1,31)
+
(32,1,31)
(32,1,31)
>> 1
(32,0,32)
×
(16,-5,21)
(16,5,11)
31970
u[n − 2]
>> 1
(32,0,32)
×
(16,-5,21)
(16,5,11)
21314
u[n − 3]
>> 2
(32,-1,33)
+
(32,-1,33)
(32,-1,33)
>> 1
(32,-2,34)
×
(16,-7,23)
(16,5,11)
21314
u[n]
>> 1
(32,-2,34)
×
(16,-7,23)
(16,5,11)
21314
u[n − 4]
>> 1
(32,0,32)
×
(16,-5,21)
(16,5,11)
21314
u[n − 1]

Some options

Roundoff around multiplication

- Roundoff After Multiplication (RAM)
- Roundoff Before Multiplication (RBM)


S
(16,9,7)
>> 16
(32,9,23)
(32,9,23)
(32,9,23)
(32,9,23)
(32,9,23)
(16,4,12)
(16,5,11)
>> 1
(16,3,13)
y[n − 1]
21660
(16,4,12)
(16,5,11)
>> 1
(16,3,13)
y[n − 2]
-22766
>> 1
(32,8,24)
(32,8,24)
(32,8,24)
>> 2
(32,6,26)
(16,3,13)
(16,5,11)
>> 1
(16,2,14)
y[n − 3]
22046
(32,6,26)
(32,6,26)
>> 4
(32,2,30)
(16,1,15)
(16,5,11)
>> 1
(16,0,16)
y[n − 4]
-16466
(32,2,30)
(32,2,30)
>> 1
(32,1,31)
>> 1
(32,1,31)
(32,1,31)
(32,1,31)
(16,-4,20)
(16,5,11)
>> 1
(16,-5,21)
u[n − 2]
31970
(16,-4,20)
(16,5,11)
>> 1
(16,-5,21)
u[n − 3]
21314
(32,1,31)
(32,1,31)
>> 2
(32,-1,33)
(16,-4,20)
(16,5,11)
>> 1
(16,-5,21)
u[n − 1]
21314
(32,-1,33)
(32,-1,33)
(16,-6,22)
(16,5,11)
>> 1
(16,-7,23)
u[n]
21314
(16,-6,22)
(16,5,11)
>> 1
(16,-7,23)
u[n − 4]
21314

Some options

Roundoff around multiplication

- Roundoff After Multiplication (RAM)
- Roundoff Before Multiplication (RBM)

Without shift

Target may oblige us to have no shifts. So, all shifts needed are deferred to constants.

Outline

Conclusion

We try to answer the following question:

For a given sum-of-products, how to produce *optimal* implementation?

The main results are a tool and a methodology:

- Model the various evaluation schemes
- Propagate FPR through trees
- Evaluate degradations

Conclusion

We try to answer the following question:

For a given sum-of-products, how to produce *optimal* implementation?

The main results are a tool and a methodology:

- Model the various evaluation schemes
- Propagate FPR through trees
- Evaluate degradations

A lot of work still to be done:

- Propagate intervals rather than FPR
- Consider adequacy with hardware resources
- Consider a more realistic model for degradation errors
- Release the source code of our tool

Thank You

Any Questions?