Reliable and accurate computing. Computer arithmetic approach for digital filters.

Anastasia Volkova

4th Workshop on the Design and Analysis of Robust Systems
July 13, 2019

Impact of numerical errors

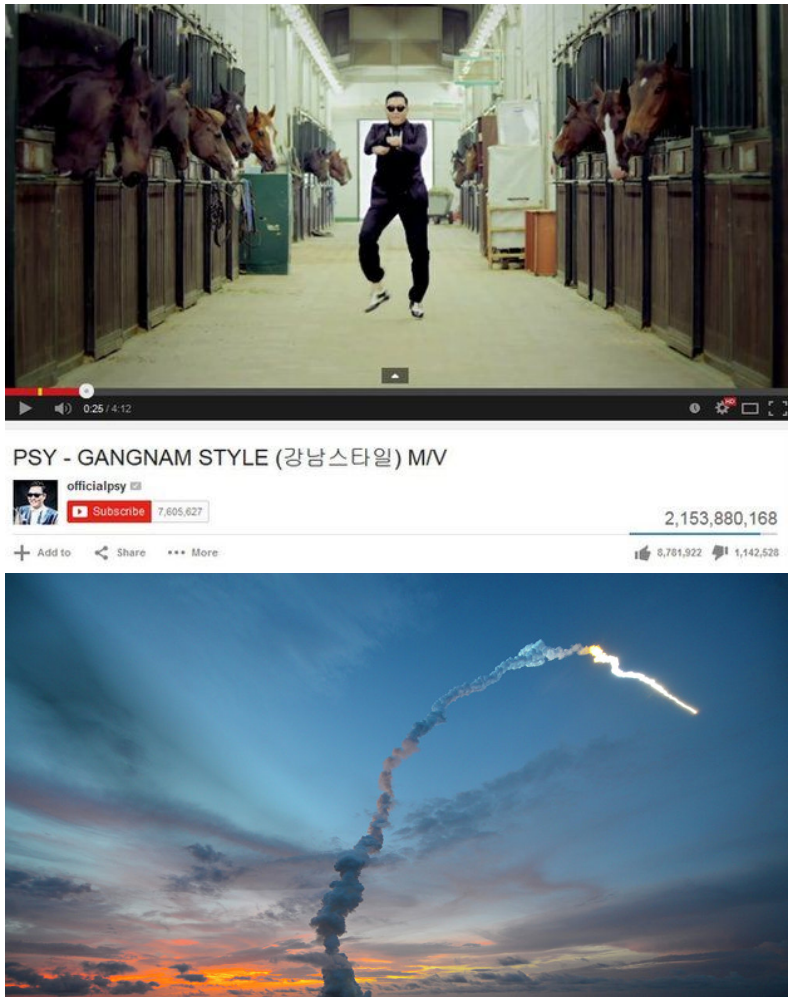
Funny fact:

- Broken YouTube

Impact of numerical errors

Funny fact:

- Broken YouTube



Not funny:

- Explosion of Ariane 5
- Boeing 787 problems
- Patriot missiles

Digitization and the choices we make

Digitization and the choices we make

- Formats and Arithmetic
  - floating-point
  - fixed-point

Digitization and the choices we make

- Formats and Arithmetic
  - floating-point
  - fixed-point
- Hardware and Software
  - CPU/GPU/FPGA/ASIC
  - language, compiler

Digitization and the choices we make

- Formats and Arithmetic
  - floating-point
  - fixed-point
- Hardware and Software
  - CPU/GPU/FPGA/ASIC
  - language, compiler
- Generalist and App. Specific
  - math. libraries[†]
  - digital filters, control, neural networks,...

[†]Wed, 17:50 "*Sound Approximation of Programs with Elementary Functions*"

Digitization and the choices we make

- Formats and Arithmetic
  - floating-point
  - fixed-point
- Hardware and Software
  - CPU/GPU/FPGA/ASIC
  - language, compiler
- Generalist and App. Specific
  - math. libraries†
  - digital filters, control, neural networks,. . .

Reliability

Automation

Optimization

†Wed, 17:50 "*Sound Approximation of Programs with Elementary Functions*"

Digitization and the choices we make

- Formats and Arithmetic
  - floating-point
  - fixed-point
- Hardware and Software
  - CPU/GPU/FPGA/ASIC
  - language, compiler
- Generalist and App. Specific
  - math. libraries†
  - **digital filters**, control, neural networks,. . .

Reliability

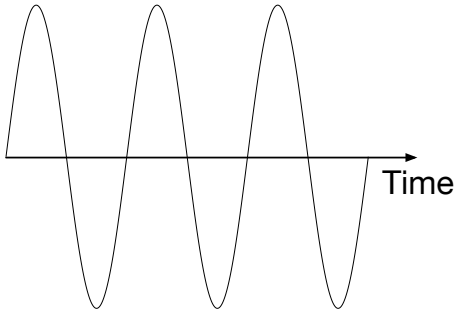
Automation

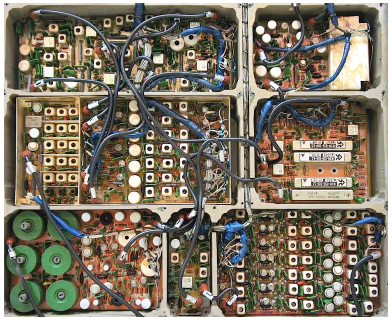
Optimization

†Wed, 17:50 "*Sound Approximation of Programs with Elementary Functions*"

Signals are everywhere

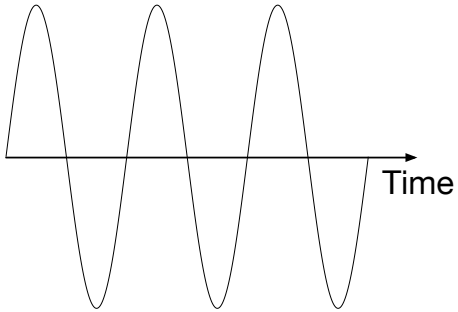
Analog signals: continuous-time

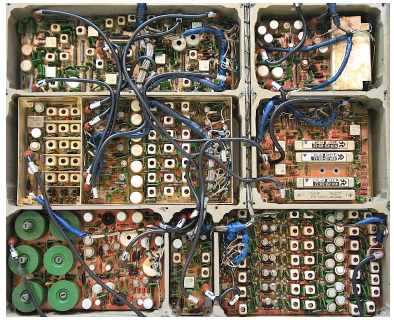




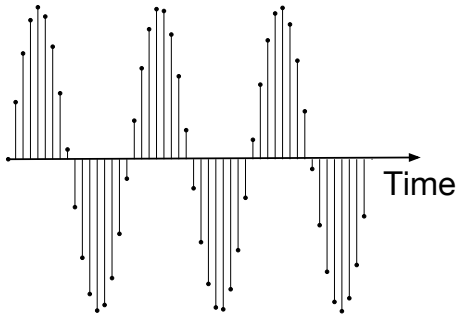
Signals are everywhere

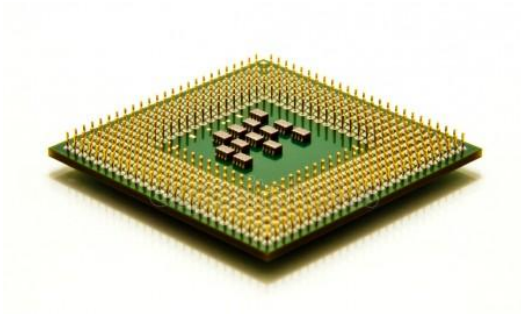
Analog signals: continuous-time



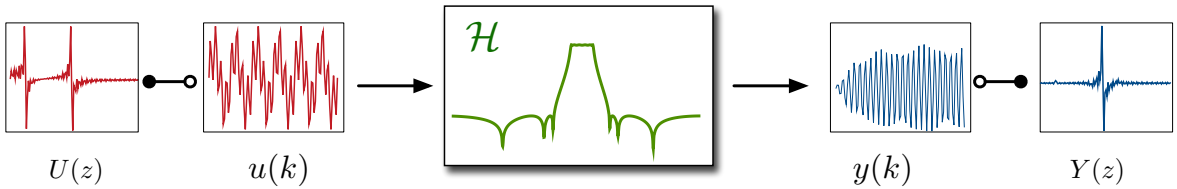


Digital signals: discrete-time

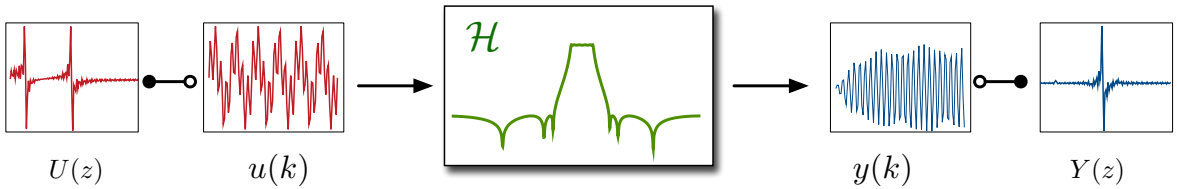




Linear digital filters



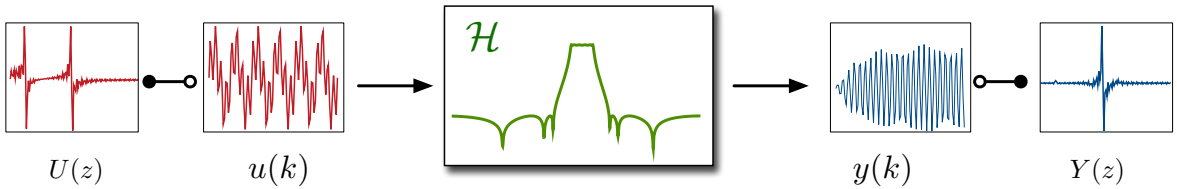
Linear digital filters



Frequency domain

$$H(z) = \frac{0.0495329964 - 0.148598989z^{-2} + \dots}{1 - 2.12060288z^{-1} + 2.7247492z^{-2} + \dots}$$

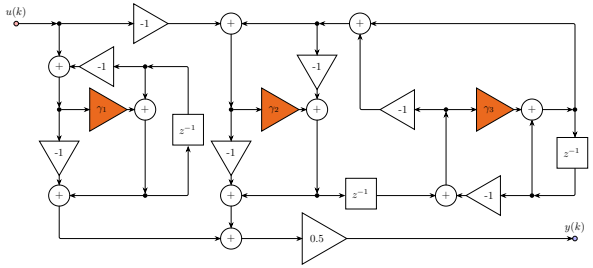
Linear digital filters



Frequency domain

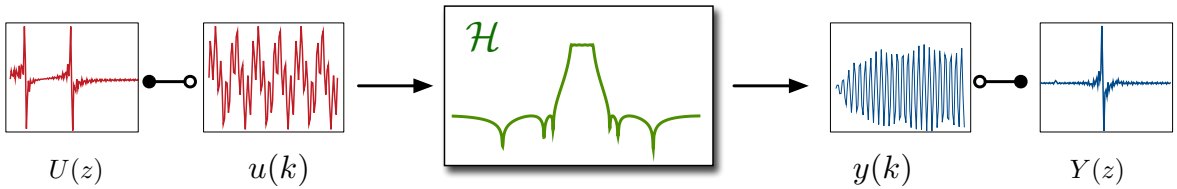
$$H(z) = \frac{0.0495329964 - 0.148598989z^{-2} + \dots}{1 - 2.12060288z^{-1} + 2.7247492z^{-2} + \dots}$$

Time domain



$$\gamma_1 = 0.347586468864209$$
$$\gamma_2 = 0.334748427563068$$
$$\gamma_3 = 0.084938546237853$$

Linear digital filters



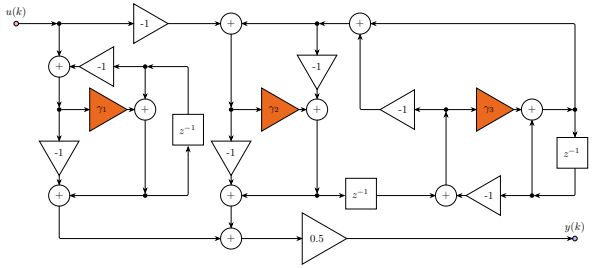
Frequency domain

$$H(z) = \frac{0.0495329964 - 0.148598989z^{-2} + \dots}{1 - 2.12060288z^{-1} + 2.7247492z^{-2} + \dots}$$

coefficient quantization
in Fixed-Point arithmetic

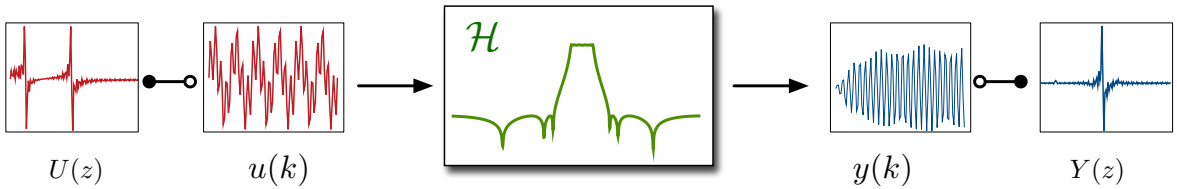
Time domain

Finite-precision
arithmetic operations



$$\gamma_1 = 0.34765625$$
$$\gamma_2 = 0.3359375$$
$$\gamma_3 = 0.0859375$$

Linear digital filters

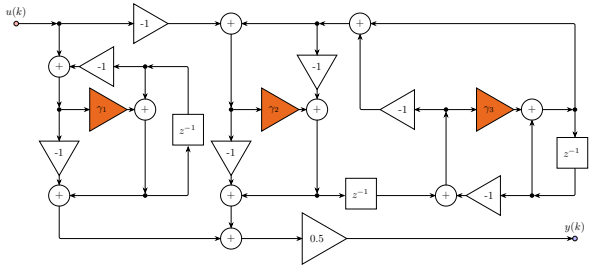


Frequency domain

$$H(z) = \frac{0.0495329964 - 0.148598989z^{-2} + \ldots}{1 - 2.12060288z^{-1} + 2.7247492z^{-2} + \ldots}$$

coefficient quantization
in Fixed-Point arithmetic

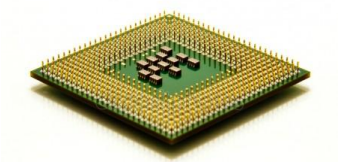
Time domain



Finite-precision
arithmetic operations

$\gamma_1 = 0.34765625$
$\gamma_2 = 0.3359375$
$\gamma_3 = 0.0859375$

Circuit / Code

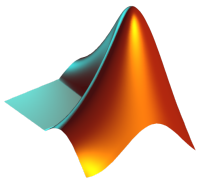


Constraints w.r.t. surface, speed, accuracy ...

Reliable and accurate digital filters

Majority of applications

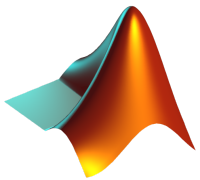
- do not need numerical guarantee
- well-studied subject
- straightforward implementations
- commercial and free tools



Reliable and accurate digital filters

Majority of applications

- do not need numerical guarantee
- well-studied subject
- straightforward implementations
- commercial and free tools



Safety-critical applications

- require numerical guarantees
- have high(er) cost
- limited literature
- require expert knowledge from engineers
- no out-of-box solution

Reliable and accurate digital filters

Majority of applications

- do not need numerical guarantee
- well-studied subject
- straightforward implementations
- commercial and free tools

Safety-critical applications

- require numerical guarantees
- have high(er) cost
- limited literature
- require expert knowledge from engineers
- no out-of-box solution

Context: safety-critical applications

Goal: generate Fixed-Point codes reliable by construction

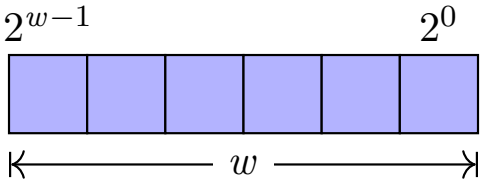
Means: interval, floating-point, multiple precision arithmetics

Outline

- Finite precision, filters and what's the issue
- FiXiF toolbox: digital filters reliable by construction
  - Worst-case analysis
  - Format choice
  - Frequency spec verification
- Last-bit accurate hardware
  - Cost of the reliability

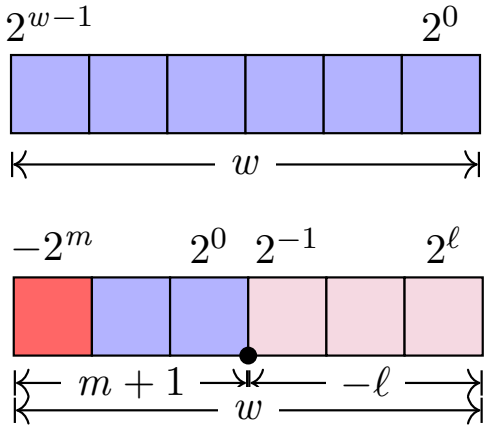
Arithmetics

- Integer arithmetic:
  $y = Y$



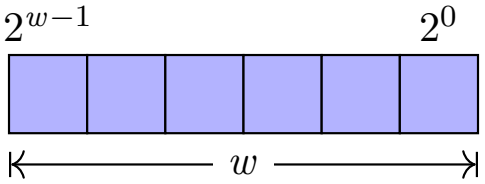
Arithmetics

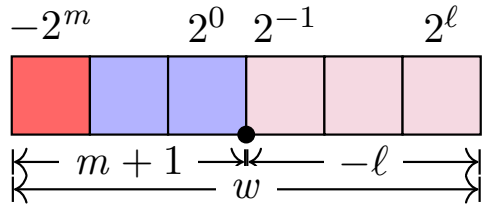
- Integer arithmetic:
  $y = Y$
- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor

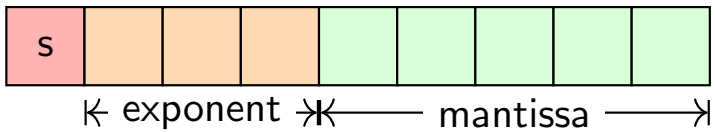


Arithmetics

- Integer arithmetic:
  $y = Y$
- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor
- Floating-Point arithmetic:
  $y = (-1)^s \cdot Y \cdot 2^e$
  where e is an *explicit* factor

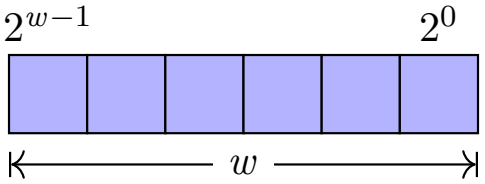


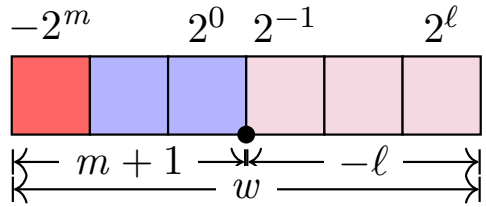


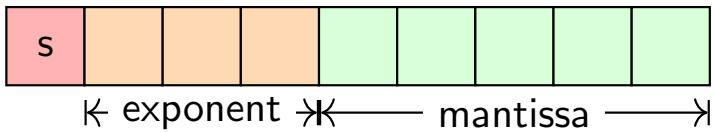


Arithmetics

- Integer arithmetic:
  $y = Y$
- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor
- Floating-Point arithmetic:
  $y = (-1)^s \cdot Y \cdot 2^e$
  where e is an *explicit* factor
- Interval arithmetic:
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$

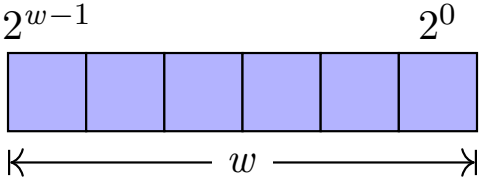


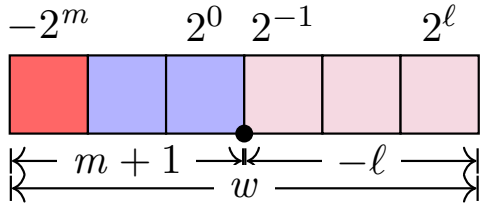


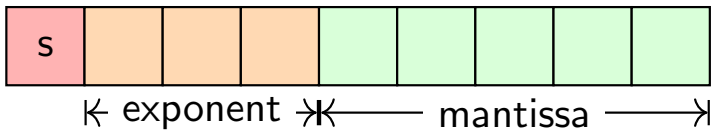


Arithmetics

- Integer arithmetic:
  $y = Y$
- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor
- Floating-Point arithmetic:
  $y = (-1)^s \cdot Y \cdot 2^e$
  where e is an *explicit* factor
- Interval arithmetic:
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$
- Multiple-Precision arithmetic: the size of the mantissa varies dynamically







Arithmetics

- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor

  For the implementation

- Floating-Point arithmetic:
  $y = (-1)^s \cdot Y \cdot 2^e$
  where e is an *explicit* factor

  For the error analysis

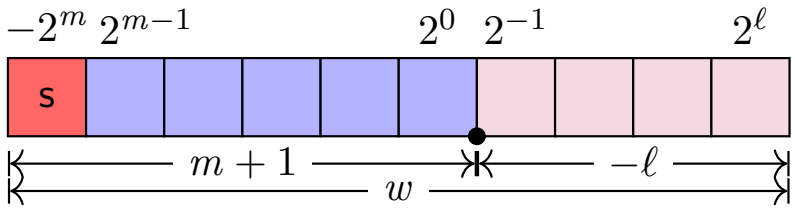
- Interval arithmetic:
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$

  For the error analysis

- Multiple-Precision arithmetic: the size of the mantissa varies dynamically

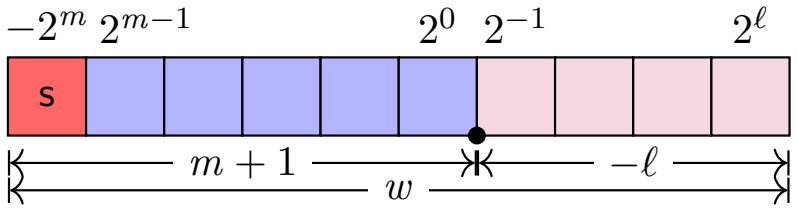
  For the error analysis

Fixed-Point Arithmetic



w	wordlength	hard constraint
m	Most Significant Bit	must choose
ℓ	Least Significant Bit	must choose

Fixed-Point Arithmetic

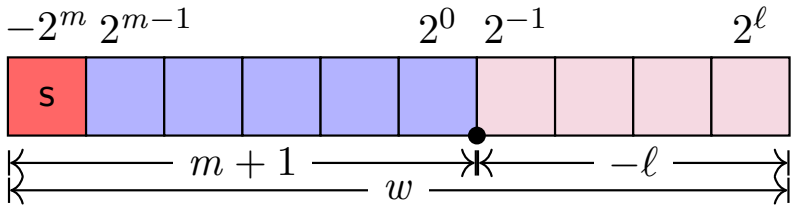


w	wordlength	hard constraint
m	Most Significant Bit	must choose
ℓ	Least Significant Bit	must choose

Example

Algorithm: vector normalization $\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$

Fixed-Point Arithmetic



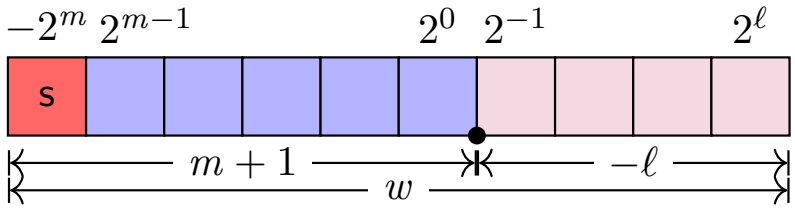
w	wordlength	hard constraint
m	Most Significant Bit	must choose
ℓ	Least Significant Bit	must choose

Example

Algorithm: vector normalization $\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$

I/O format: $w = 32, m = 15, \ell = -16$

Fixed-Point Arithmetic



w	wordlength	hard constraint
m	Most Significant Bit	must choose
ℓ	Least Significant Bit	must choose

Example

Algorithm: vector normalization $\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$

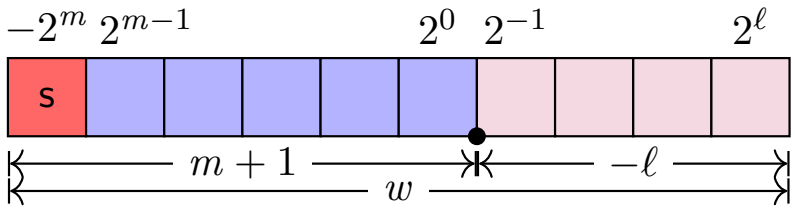
I/O format: $w = 32, m = 15, \ell = -16$

Input: $v = (125, 125, 125)$

Overflow: $125^2 + 125^2 + 125^2 = 46875 \notin [-2^{15}; 2^{15} - 1]$

Output: depends on the system

Fixed-Point Arithmetic



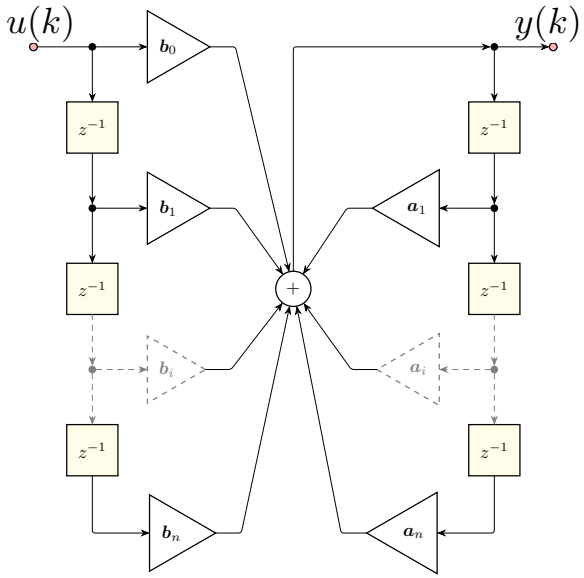
w	wordlength	hard constraint
m	Most Significant Bit	must choose
ℓ	Least Significant Bit	must choose

Format choice problem:

- fix wordlength
- guarantee no overflow
- maximize the precision
- bound the output error

Filter algorithms

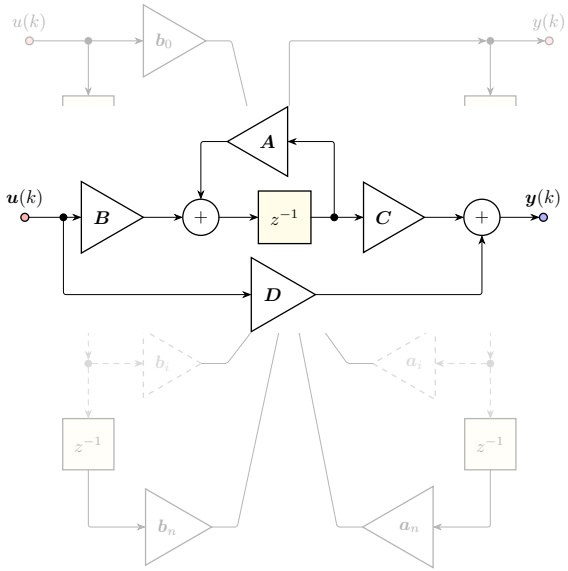
Typical algorithm: input $u(k)$, internal state $\boldsymbol{x}(k)$, output $y(k)$



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

Filter algorithms

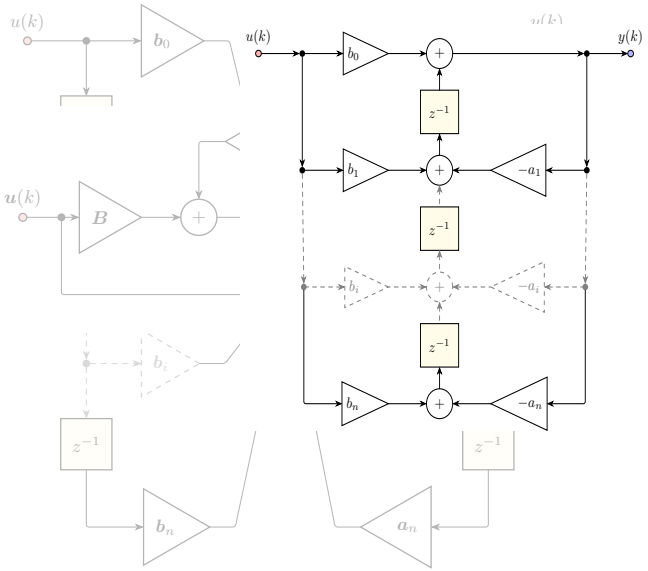
Typical algorithm: input $u(k)$, internal state $\boldsymbol{x}(k)$, output $y(k)$



- $y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$
- $$\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Filter algorithms

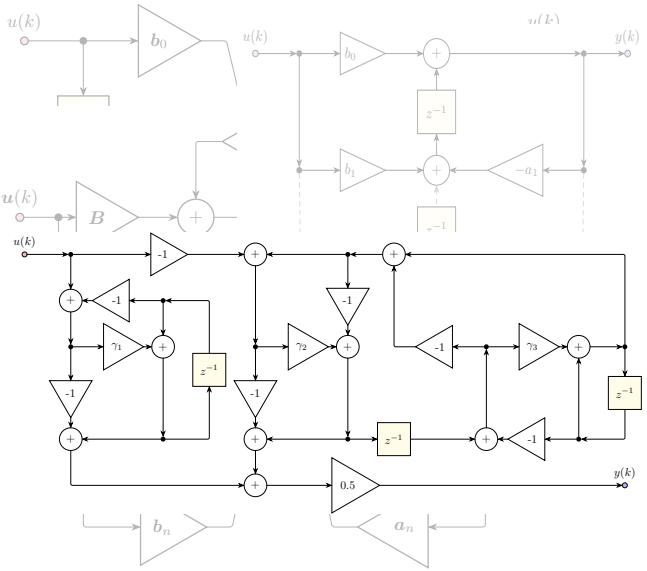
Typical algorithm: input $u(k)$, internal state $\boldsymbol{x}(k)$, output $y(k)$



- $y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$
- $\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$
- ...

Filter algorithms

Typical algorithm: input $u(k)$, internal state $\boldsymbol{x}(k)$, output $y(k)$

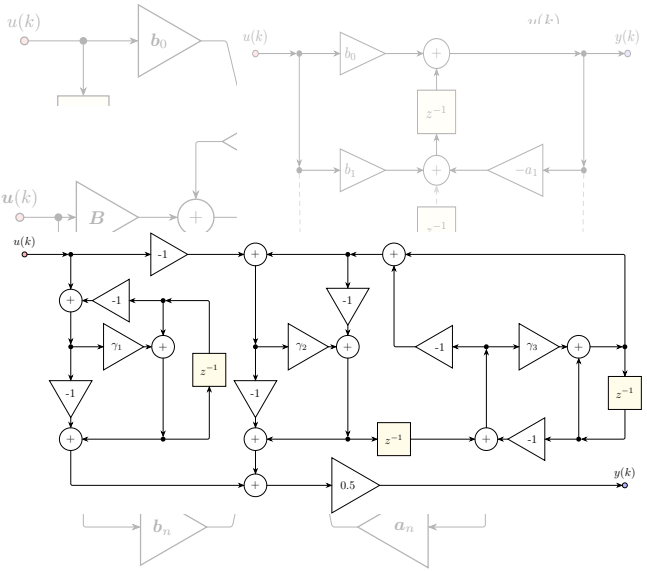


- $y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$
- $\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$
- ...

Math: "It's all the same thing"

Filter algorithms

Typical algorithm: input $u(k)$, internal state $\boldsymbol{x}(k)$, output $y(k)$



- $y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$
- $\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$
- ...

Math: "It's all the same thing"

"Not anymore!", **finite-precision**.

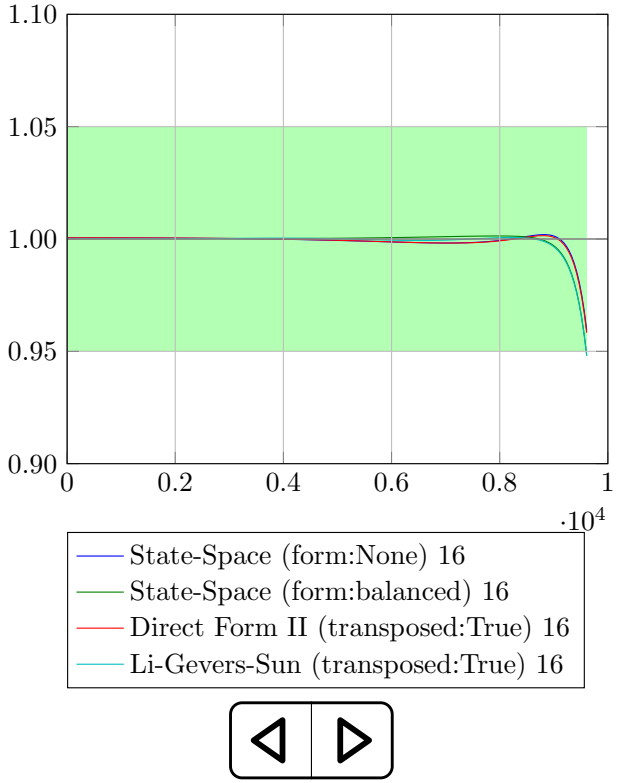
Example: a 15^{th} order lowpass filter

Algorithm	# coefficients
State-Space (canonical)	31
State-Space (balanced)	256
Direct Form II (transposed)	31
Li-Gevers-Sun	102

Example: a 15^{th} order lowpass filter

Algorithm	# coefficients
State-Space (canonical)	31
State-Space (balanced)	256
Direct Form II (transposed)	31
Li-Gevers-Sun	102

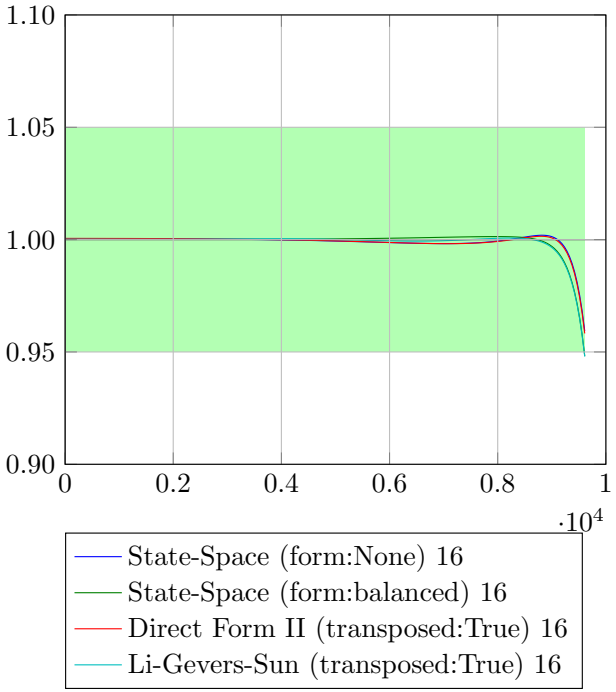
Coefficient quantization



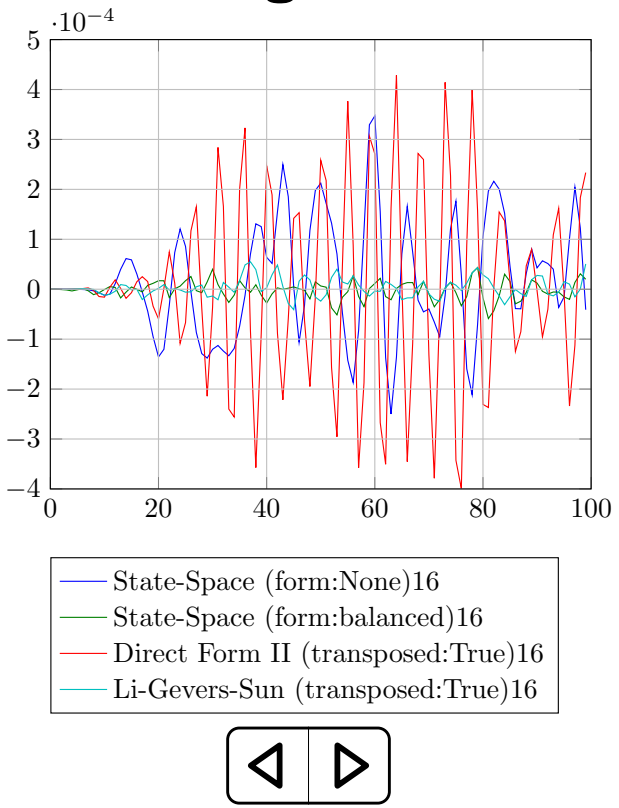
Example: a 15^{th} order lowpass filter

Algorithm	# coefficients
State-Space (canonical)	31
State-Space (balanced)	256
Direct Form II (transposed)	31
Li-Gevers-Sun	102

Coefficient quantization



Rounding errors



Existing approaches on FxP implementation

- simulations [Matlab], [D. Báez-López,2001]
  - non-exhaustive
- noise propagation models [Menard, 2008]
  - does not give *intervals* for outputs
- interval arithmetic [Carreras, 1999], [Vakili, 2013]
  - wrapping effect for recursive systems
- affine arithmetic [Puschel, 2003], [Constantinides, 2006]
  - need to bound ℓ_∞ norm of the filter's output
  - static unroll of the loops

FIXIF

TOOLBOX

"Digital filters reliable by construction "

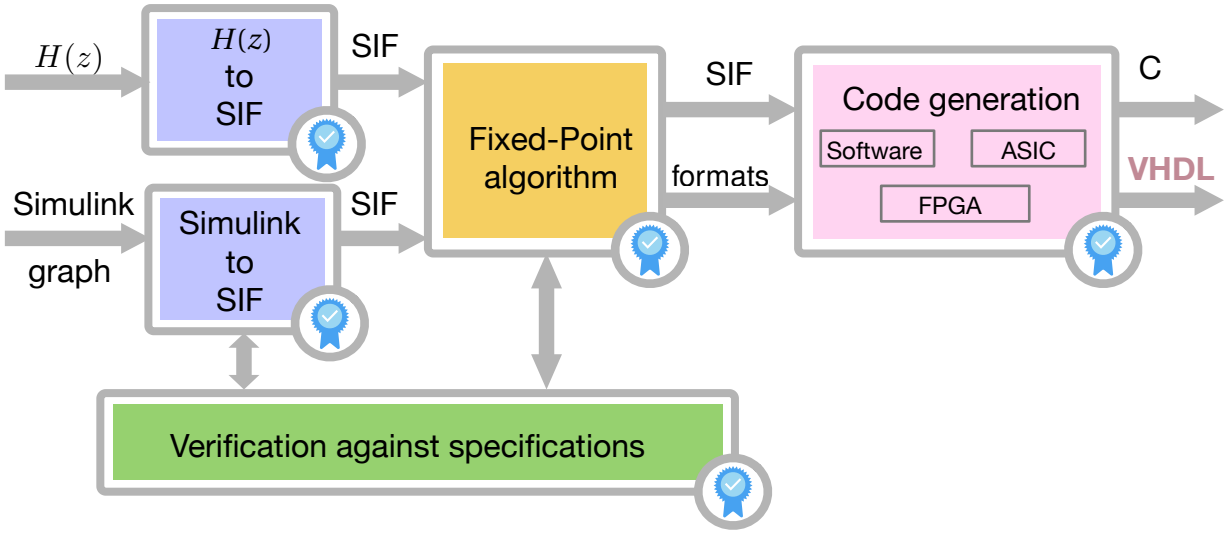
Available at `https://github.com/fixif`

Joint work with

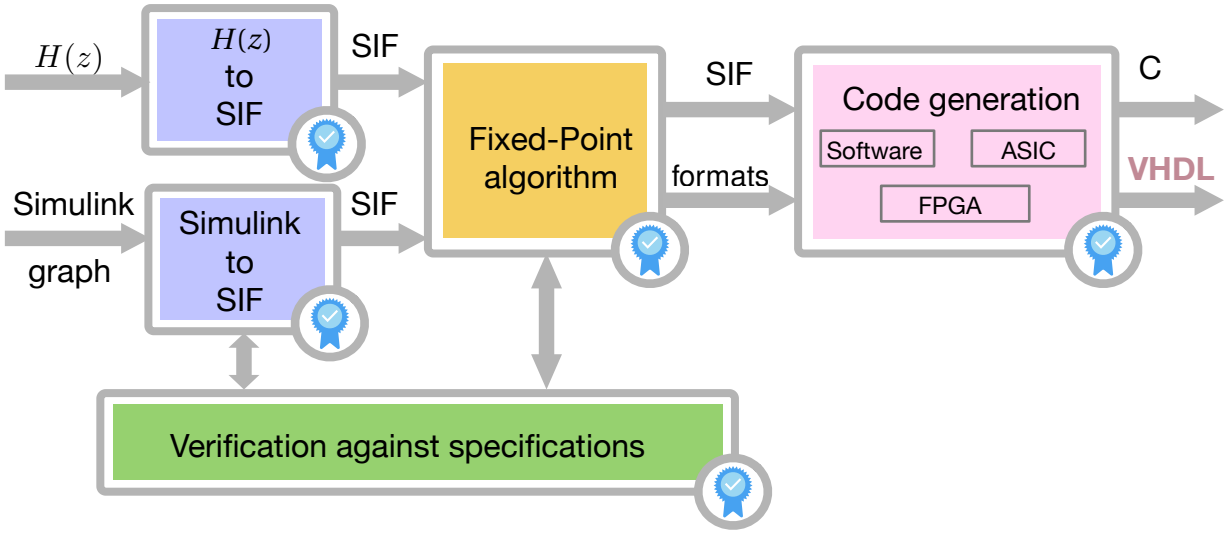
Thibault Hilaire
`thibault.hilaire@lip6.fr`

Christoph Lauter
`chirstoph.lauter@lip6.fr`

Compiler overview

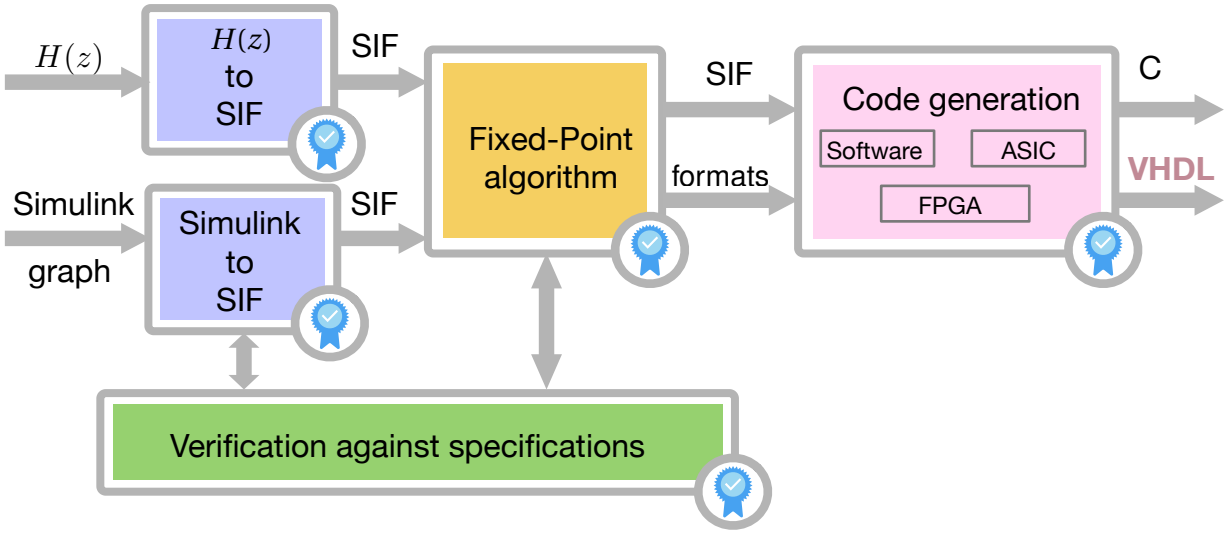


Compiler overview



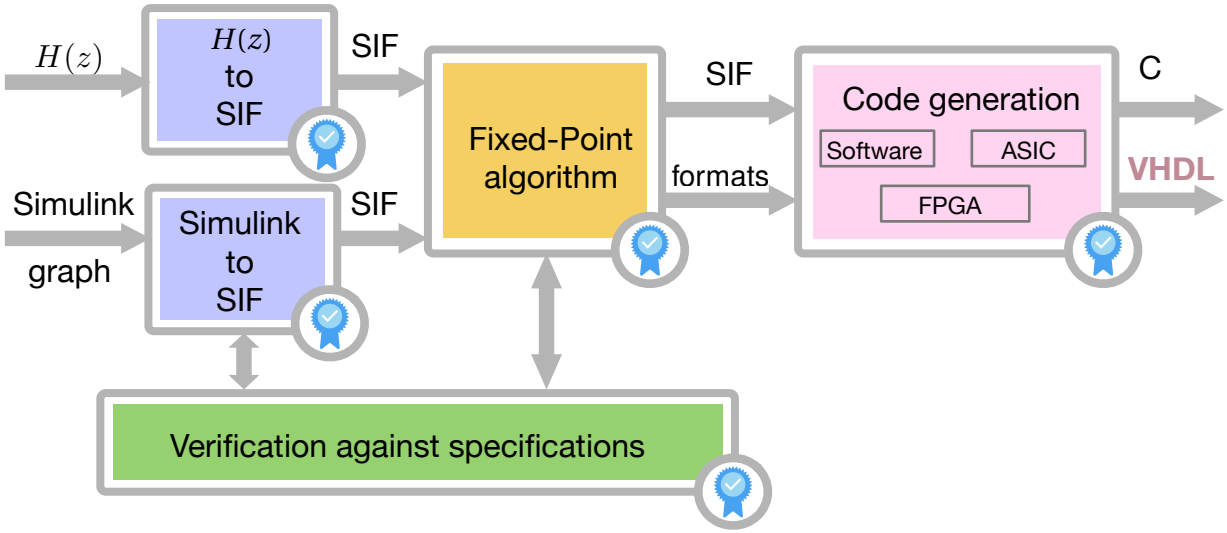
- Internal representation: SIF (matrix-based data-flow description)

Compiler overview



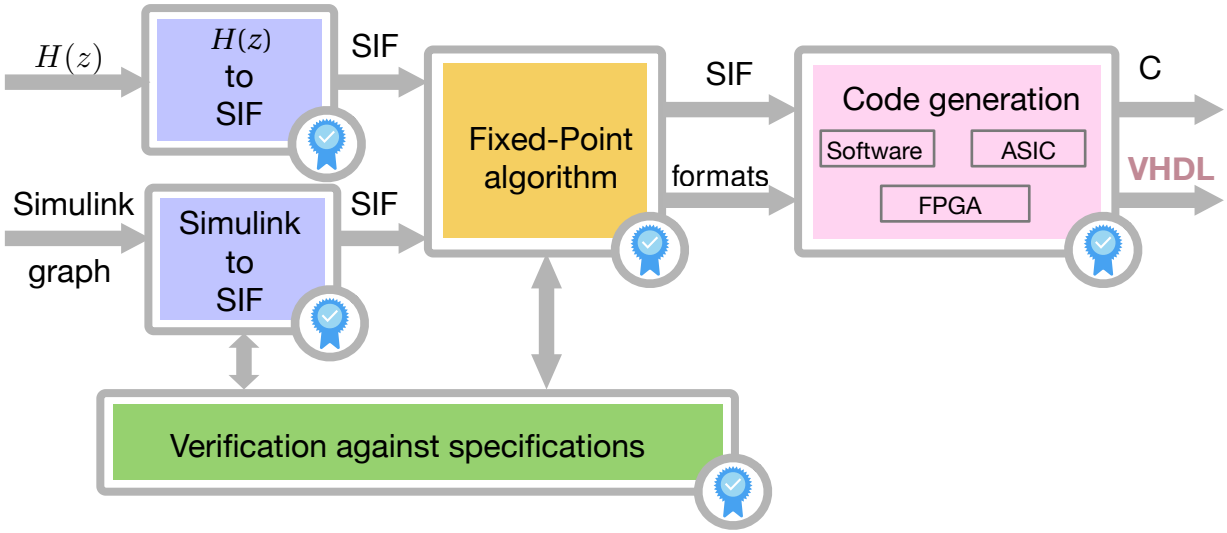
- Internal representation: SIF (matrix-based data-flow description)
- Rounding errors: unified, reliable and fast FxP algorithm generation

Compiler overview



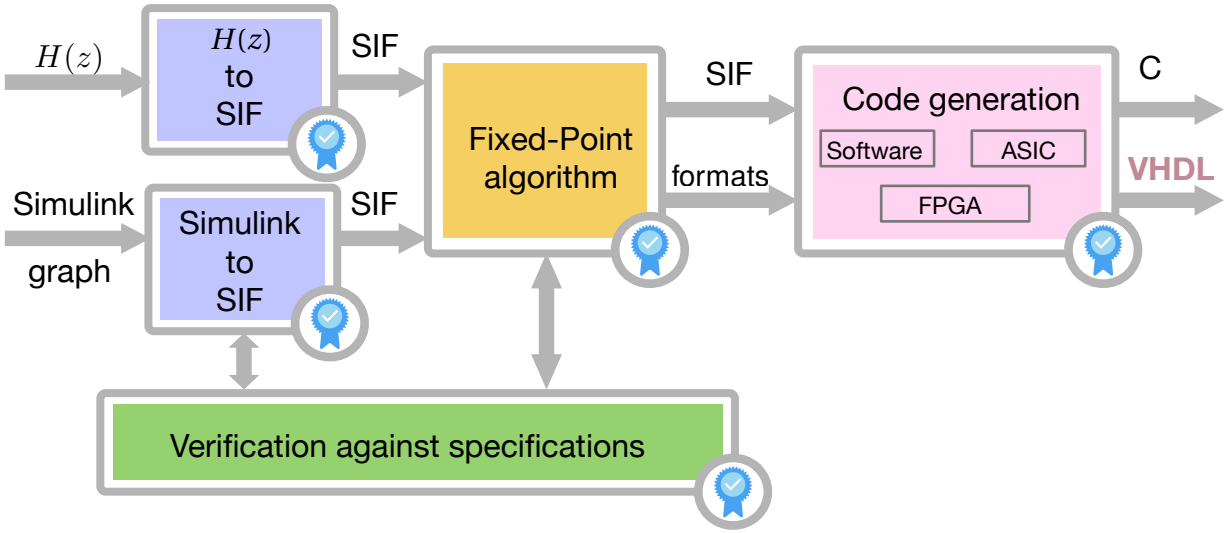
- Internal representation: SIF (matrix-based data-flow description)
- Rounding errors: unified, reliable and fast FxP algorithm generation
- Quantization errors: rigorous verification against frequency specs

Compiler overview



- Internal representation: SIF (matrix-based data-flow description)
- Rounding errors: unified, reliable and fast FxP algorithm generation
- Quantization errors: rigorous verification against frequency specs
- Software implementation: floating- and fixed-point C

Compiler overview



- Internal representation: SIF (matrix-based data-flow description)
- Rounding errors: unified, reliable and fast FxP algorithm generation
- Quantization errors: rigorous verification against frequency specs
- Software implementation: floating- and fixed-point C
- Hardware implementation: VHDL generation based on FloPoCo[a]

[a] `http://flopoco.gforge.inria.fr/`

FiXiF tool: under the hood

Front-end

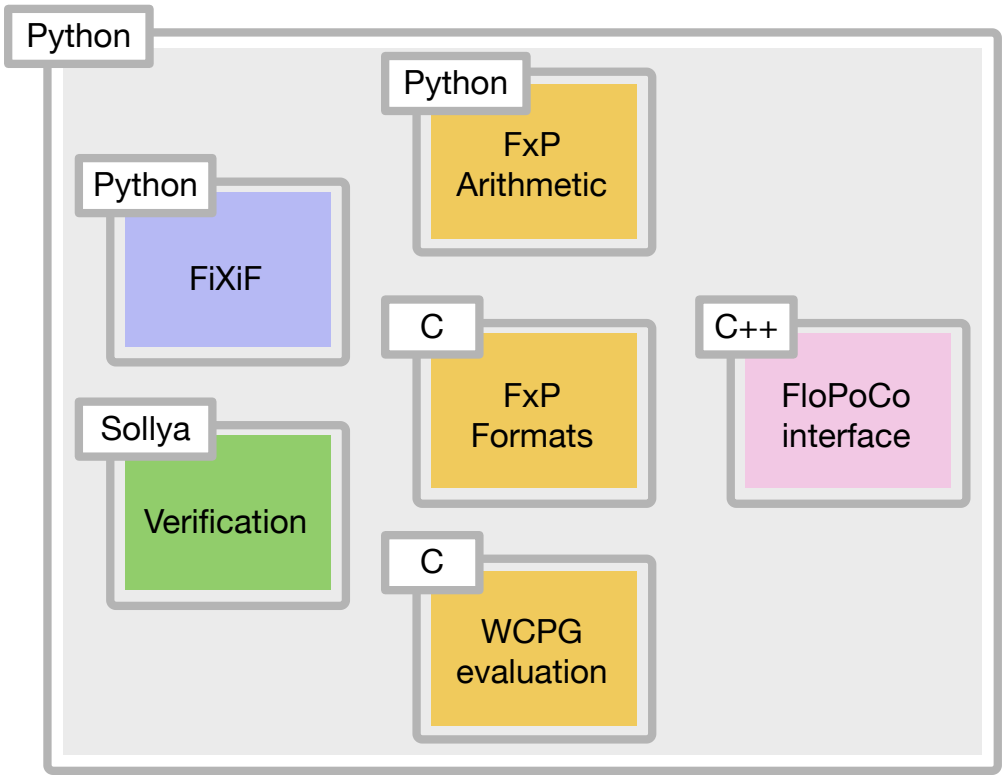
- Python
- Collection of classes

Back-end

- C/C++ libraries
- Sollya procedures
- FloPoCo tool

Requirements:

- `MPFR, MPFI, LAPACK`
- `mpmath`



FiXiF – 1

- Filter specifications and transfer function design

```
from fixif import *
g = Gabarit(48000, [(0, 9600), (12000, None)], [(0.95, 1.05), -20])
H = g.to_dTF(ftype="butter", method="scipy")
F = Filter(tf=H)
```

FiXiF – 1

- Filter specifications and transfer function design

```
from fixif import *
g = Gabarit(48000, [(0, 9600), (12000, None)], [(0.95, 1.05), -20])
H = g.to_dTF(ftype="butter", method="scipy")
F = Filter(tf=H)
```

```
Type:  lowpass (Fs=48000Hz)
Freq.  [0Hz,9600Hz]:  Passband in [0.95dB, 1.05dB]
Freq.  [12000Hz,24000.0Hz]:  Stopband at -20dB

      4.58056194e-05 + 6.87084291e-04 z^-1 + 4.8095900e-03z^-2 + ...
H(z) = ------------------------------------------------------------
                1.0 + -1.6206385z^-1 + 3.20872535z^-2 + ...
```

FiXiF – 1

- Filter specifications and transfer function design

```
from fixif import *
g = Gabarit(48000, [(0, 9600), (12000, None)], [(0.95, 1.05), -20])
H = g.to_dTF(ftype="butter", method="scipy")
F = Filter(tf=H)
```

- Filter realization (algorithm)

```
R_SS = State_Space(Filter(tf=H))
R_SS_balanced = State_Space(Filter(tf=H), form="balanced")
R_DFII = DFII(Filter(tf=H), transposed=True)
R_LGS = LGS(Filter(tf=H), transposed=True)
```

State-Space and an error model

Take a State-Space algorithm

$$\mathcal{H} \begin{cases} \boldsymbol{x}\ (k+1) &= & \boldsymbol{A}\boldsymbol{x}\ (k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}\ (k) &= & \boldsymbol{C}\boldsymbol{x}\ (k) + \boldsymbol{D}\boldsymbol{u}(k) \end{cases}$$

State-Space and an error model

Take a State-Space algorithm and add rounding

$$\mathcal{H}^{\Diamond} \begin{cases} \boldsymbol{x}^{\Diamond}(k+1) &= \Diamond_{m_x}(\boldsymbol{A}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{B}\boldsymbol{u}(k)) \\ \boldsymbol{y}^{\Diamond}(k) &= \Diamond_{m_y}(\boldsymbol{C}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{D}\boldsymbol{u}(k)) \end{cases}$$

where $\Diamond_m$ is some operator ensuring faithful rounding:

$$|\Diamond_m(x) - x| \leq 2^{\ell}$$

State-Space and an error model

Take a State-Space algorithm and add rounding

$$\mathcal{H}^{\Diamond} \begin{cases} \boldsymbol{x}^{\Diamond}(k+1) &= \boldsymbol{A}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{B}\boldsymbol{u}(k) &+ \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^{\Diamond}(k) &= \boldsymbol{C}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{D}\boldsymbol{u}(k) &+ \boldsymbol{\varepsilon}_y(k) \end{cases}$$

with

$$|\boldsymbol{\varepsilon}_x(k)| \leq 2^{\boldsymbol{\ell}_x} \qquad \text{and} \qquad |\boldsymbol{\varepsilon}_y(k)| \leq 2^{\boldsymbol{\ell}_y}$$

State-Space and an error model

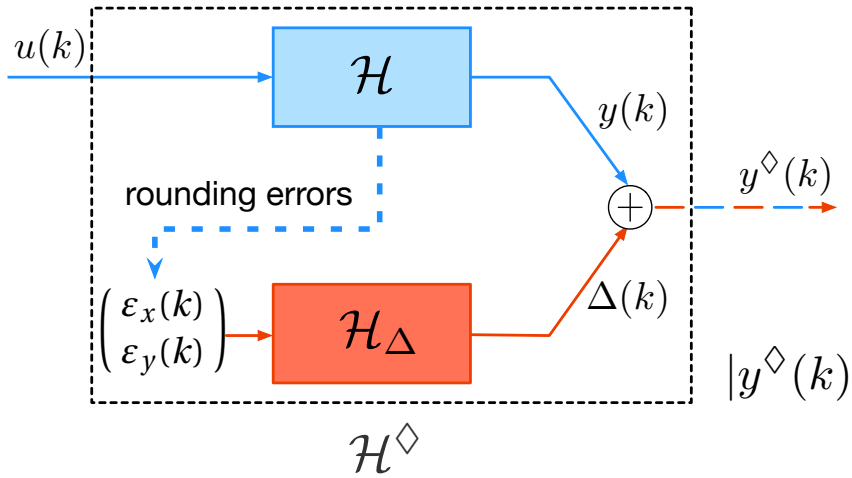
Take a State-Space algorithm and add rounding

$$\mathcal{H}^{\Diamond} \begin{cases} \boldsymbol{x}^{\Diamond}(k+1) &= \boldsymbol{A}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{B}\boldsymbol{u}(k) &+ \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^{\Diamond}(k) &= \boldsymbol{C}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{D}\boldsymbol{u}(k) &+ \boldsymbol{\varepsilon}_y(k) \end{cases}$$

with

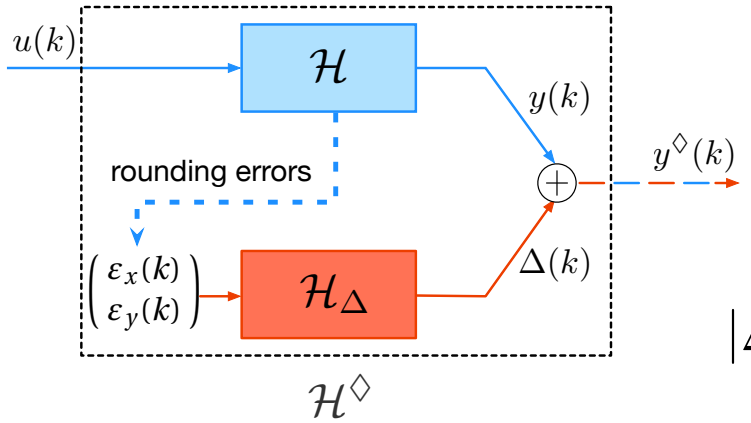
$$|\boldsymbol{\varepsilon}_x(k)| \leq 2^{\boldsymbol{\ell}_x} \qquad \text{and} \qquad |\boldsymbol{\varepsilon}_y(k)| \leq 2^{\boldsymbol{\ell}_y}$$

We express $\boldsymbol{x}(k) - \boldsymbol{x}^{\Diamond}$ and $\boldsymbol{y}(k) - \boldsymbol{y}^{\Diamond}(k)$ ... using a new filter:



$$|y^{\Diamond}(k)| \leq |\Delta(k)| + |y(k)|$$

Reliable implementation



$$|\Delta(k)| \leq \langle\langle \mathcal{H}_\Delta \rangle\rangle \begin{pmatrix} 2^{\ell_x} \\ 2^{\ell_y} \end{pmatrix}$$

Basic bricks:

- Reliable bound on a linear filter's output
  - Worst-Case Peak Gain measure [V-Lauter-Hilaire'15]
- Take into account error propagation
  - Iterative refinement of MSB [V-Lauter-Hilaire'16]
  - Never overestimate MSB by more than one

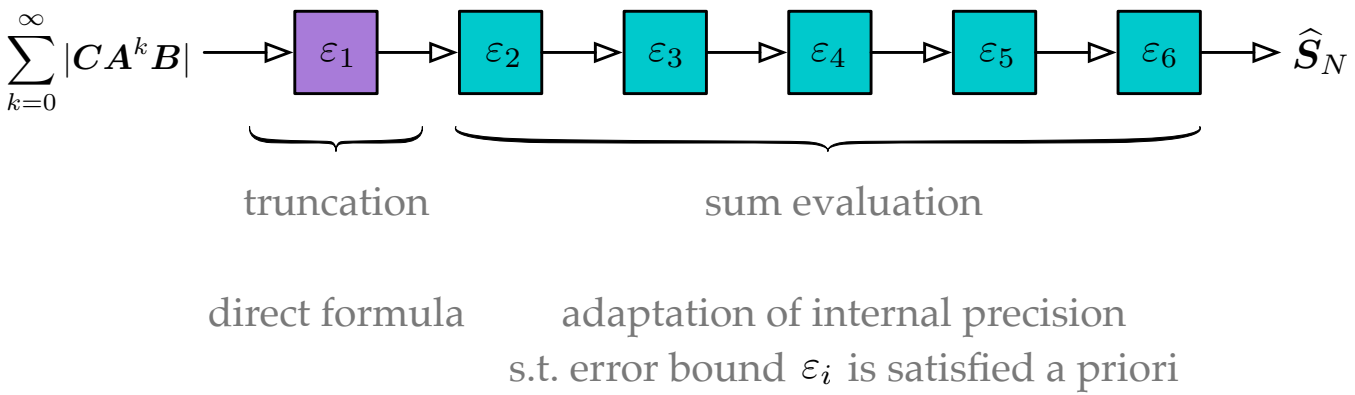
Worst-Case Peak Gain

- Matrix $\langle\langle \mathcal{S} \rangle\rangle = \sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$ [Balakrishnyan-Boyd'91]

Problem: approximation and evaluation of the WCPG with an *a priori* absolute error bound ε

$$\left| \langle\langle \mathcal{S} \rangle\rangle - \langle\langle \widehat{\mathcal{S}} \rangle\rangle \right| \leq \left| \langle\langle \mathcal{S} \rangle\rangle - \langle\langle \mathcal{S}_N \rangle\rangle \right| + \left| \langle\langle \mathcal{S}_N \rangle\rangle - \langle\langle \widehat{\mathcal{S}} \rangle\rangle \right| < \varepsilon$$

Solution [V-Hilaire-Lauter'15]**:**



Techniques: a priori floating-point error analysis, verified inclusions for eigendecomposition, Gershgorin circles computation

FiXiF – 2

- Determine the Fixed-Point Formats

```
w = 16
while True:
        msb, lsb, error, additionalSteps = FXPF_ABCD(R_SS.A,
        R_SS.B, R_SS.C, R_SS.D, 1.0, w)
        w = w - 1
```

FiXiF – 2

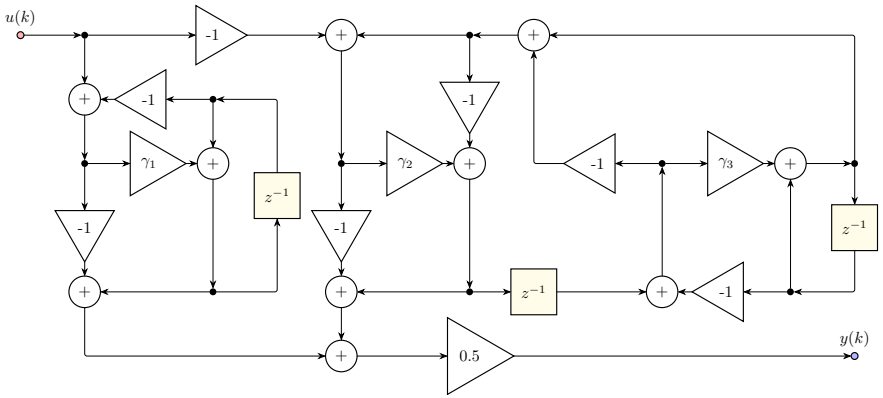
- Determine the Fixed-Point Formats

```
w = 16
while True:
        msb, lsb, error, additionalSteps = FXPF_ABCD(R_SS.A,
        R_SS.B, R_SS.C, R_SS.D, 1.0, w)
        w = w - 1
```

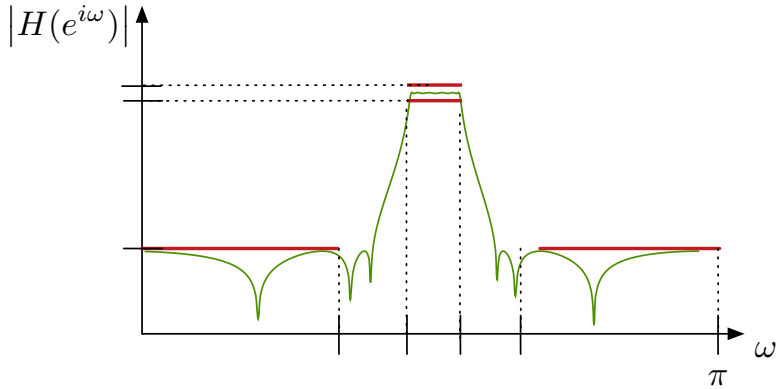
```
  Wordlength=[16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16]

Additional steps:  1
LSB: [-13 -13 -14 -14 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15]
Errors:  [[ 0.00086691 0.00121799 0.00152309 0.00172836 0.00158205
0.00121137 0.00079471 0.00048683 0.00029988 0.0001935 0.00013235
0.00009573 0.00007329 0.00005811 0.00004509 0.00152433]]
⋮
Wordlength=[5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
Error:  LSB reached point when it is larger than initial MSB estimation
Error determining Fixed-Point Formats.
```

Verification of specifications

Implementation :



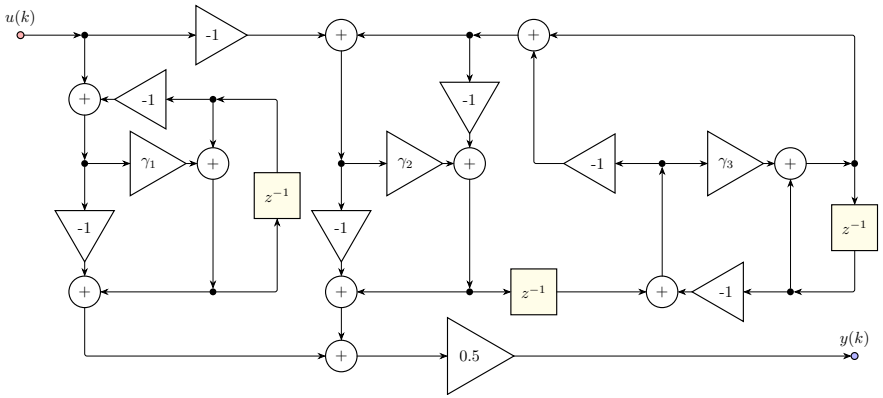
Verify the specifications:



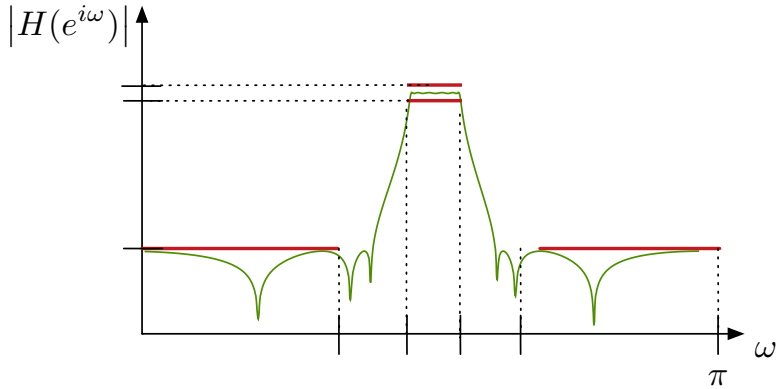
no false positives

Verification of specifications

Implementation :



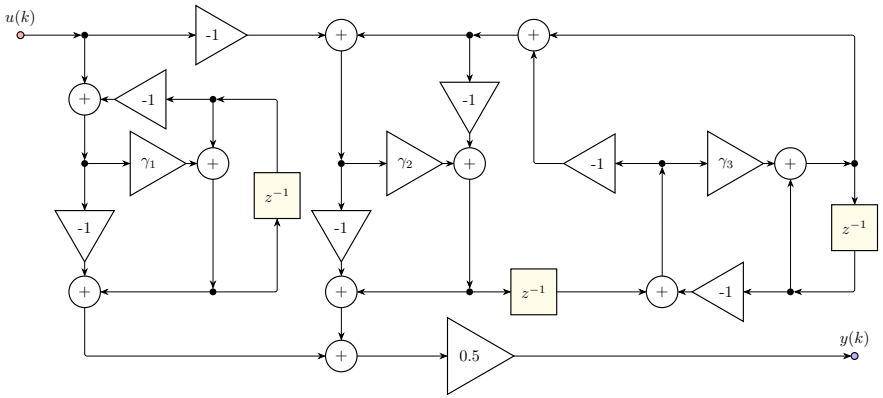
Verify the specifications:



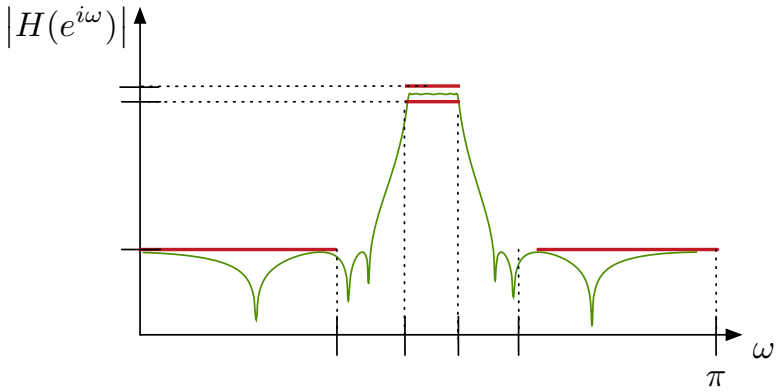
no false positives

Verification of specifications

Implementation :



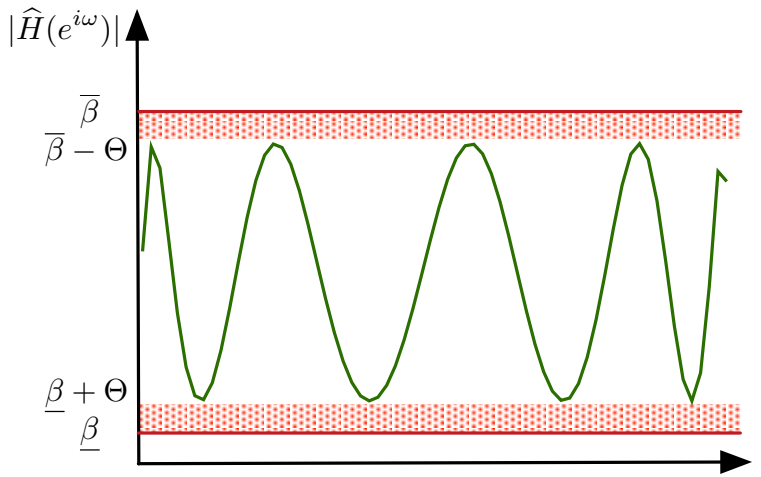
Verify the specifications:



no false positives

Solution [V-Lauter-Hilaire'17]

- MP approximation of $\widehat{\boldsymbol{H}}(z)$
  $\rightarrow$ error Θ bounded using WCPG [Balakrishnyan'92]
- Verification of $\widehat{\boldsymbol{H}}(z)$
  $\rightarrow$ proof of non-negativity of a polynomial



FiXiF – 3

- Verification

```
g = Gabarit(48000, [(0, 9600), (12000, None)], [(0.95, 1.05), -20])
CheckIfRealizationInGabarit(g, R_SS.quantize(16))
```

FiXiF – 3

- Verification

```
g = Gabarit(48000, [(0, 9600), (12000, None)], [(0.95, 1.05), -20])
CheckIfRealizationInGabarit(g, R_SS.quantize(16))
```

```
  Overall check okay:  false
Computing this result took 2258ms
The following issues have been found:
Issue in subdomain Omega = pi * [0;0.4] at omega = pi *
[0;5.3172964416593691658936355013934529087741417880979e-53]:
|H(exp(j*omega))| should be between 10^ 4.75e-2 and 10^ (1.05 / 20) but
evaluates to [1.000049779552370051923399028475943575964016437656 5;
1.000049779552370051923399028475943575964016437656 6] =
10^ ([4.323689366417695635232485092238116632596970479257e-4;
4.323689366417695635232485092238116632596970486685 8e-4]/20)
⋮
```

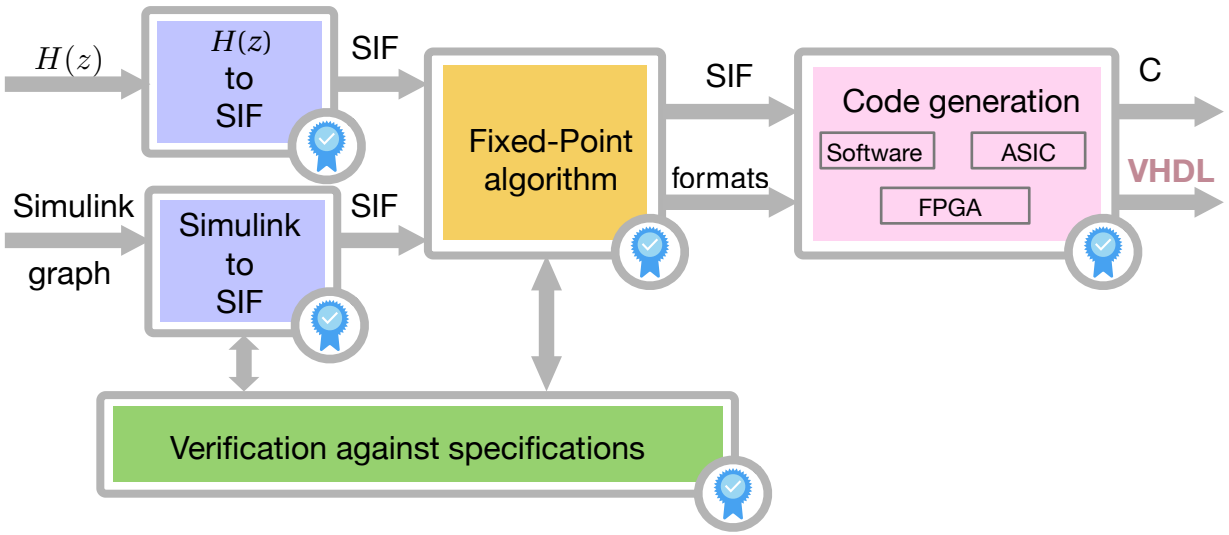
Behind the scenes

During these 2.26 seconds...

- Rational arithmetic
  - proof of non-negativity of a real polynomial
- Interval arithmetic
  - Verified Inclusions for the eigenvalues
- Dynamic Multiple-Precision arithmetic
  - Computation of Gershgorin circles
  - Evaluation of WCPG
- Floating-Point arithmetic
  - Computation of eigenvalues with LAPACK
  - Matrix arithmetic with *a priori* error bounds

. . . to verify an implementation in Fixed-Point arithmetic!

Back to the generator



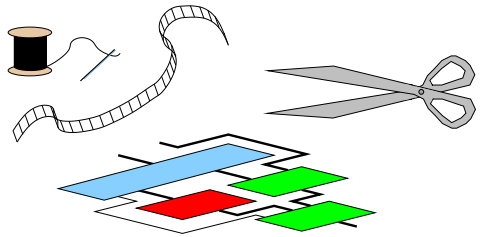
- Rigorous use of a combination of different arithmetics
- Numerical guarantees in time and frequency domains
- Unifying internal representation
- Easily extendable modular implementation

NEW Non-linear optimization to minimize circuit area [Hilaire, 2019]

Last-bit accurate hardware

FIXIF TOOLBOX

FloPoCo



http://flopoco.gforge.inria.fr/

Joint work with

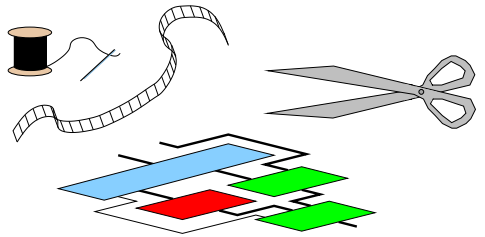
Florent de Dinechin
Florent.de-Dinechin@
insa-lyon.fr

Matei Istoan

Last-bit accurate hardware

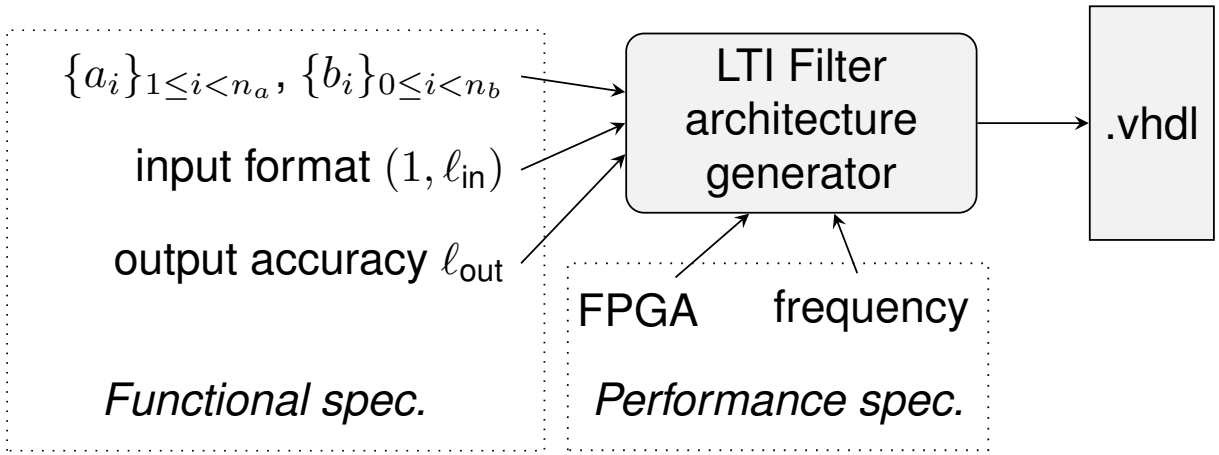
FIXIF TOOLBOX

FloPoCo



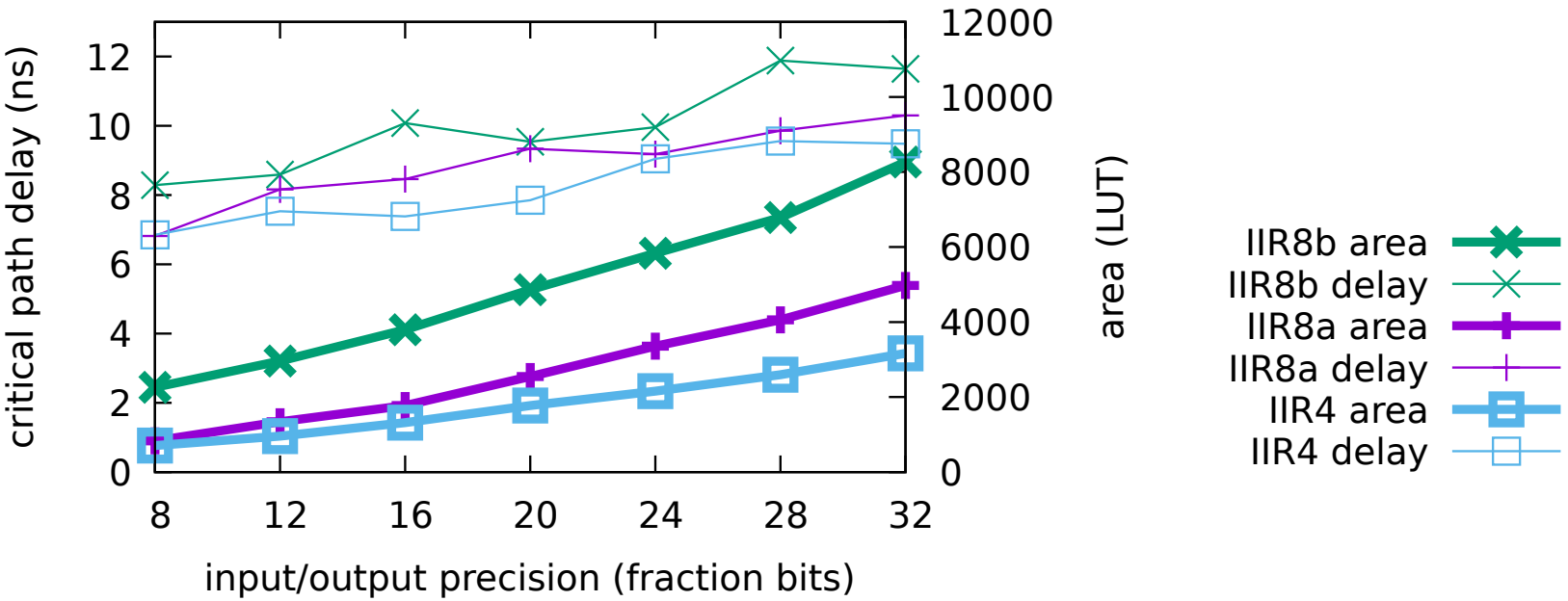
http://flopoco.gforge.inria.fr/

A push-button tool that generates filters accurate just-enough



The cost of reliability

Area and Delay vs. Accuracy



The cost of reliability

Pessimism of the method:

- (almost) worst-case input signal
- removing ≥ 2 bits yields failure

The cost of reliability

Pessimism of the method:

- (almost) worst-case input signal
- removing ≥ 2 bits yields failure

$\Rightarrow$ bounds are tight

The cost of reliability

Pessimism of the method:

- (almost) worst-case input signal
- removing ≥ 2 bits yields failure

$\Rightarrow$ bounds are tight

SQNR vs. Guaranteed accuracy

[Constantinides, 2004]	Our approach
16 bit output	8 (13) bit output
SQNR 49.3dB to 78.5dB	error bound $\leq 2^{-8}$ ($\leq 2^{-13}$)
8 to 13 meaningful bits	last-bit accurate
16 bit multipliers	18 (23) bits datapaths
16/32 bit accumulators	

The cost of reliability

Pessimism of the method:

- (almost) worst-case input signal
- removing ≥ 2 bits yields failure

$\Rightarrow$ bounds are tight

SQNR vs. Guaranteed accuracy

[Constantinides, 2004]	Our approach
16 bit output	8 (13) bit output
SQNR 49.3dB to 78.5dB	error bound $\leq 2^{-8}$ ($\leq 2^{-13}$)
8 to 13 meaningful bits	last-bit accurate
16 bit multipliers	18 (23) bits datapaths
16/32 bit accumulators	

$\Rightarrow$ we are competitive

FIXIF

TOOLBOX

"Digital filters reliable by construction "

`https://github.com/fixif`

Anastasia Volkova
`anastasia.volkova@inria.fr`
`avolkova.org`

Thibault Hilaire
`thibault.hilaire@lip6.fr`
`docmatic.fr`

Christoph Lauter
`chirstoph.lauter@lip6.fr`
`christoph-lauter.org`