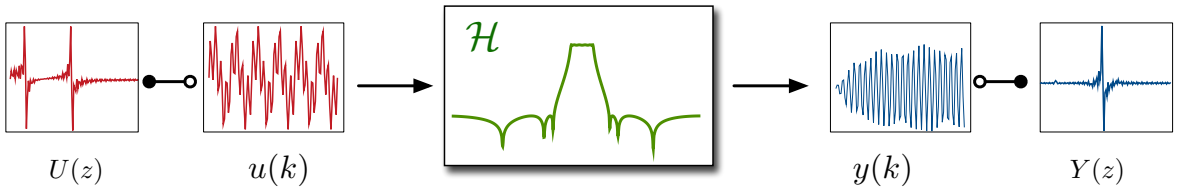
FiXiF toolbox: numerically reliable digital filters

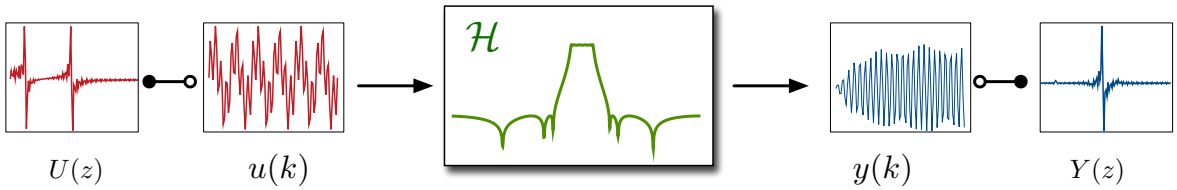
Anastasia Volkova

SCAN 2018
September 12, 2018

Linear digital filters



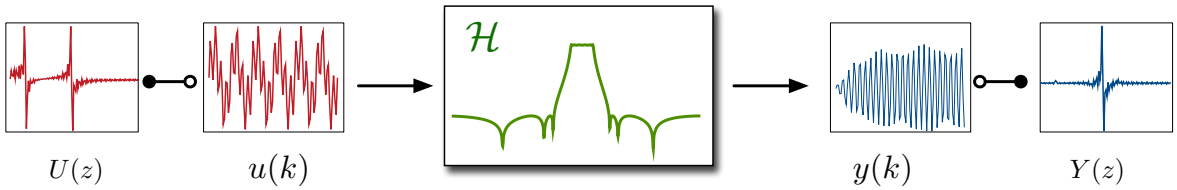
Linear digital filters



Frequency domain

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

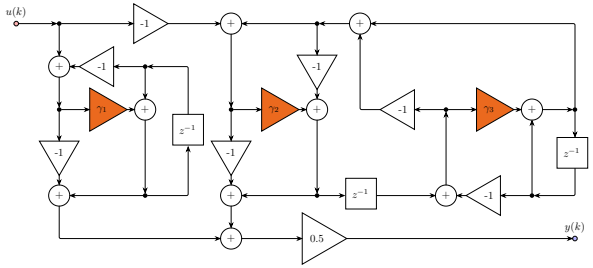
Linear digital filters



Frequency domain

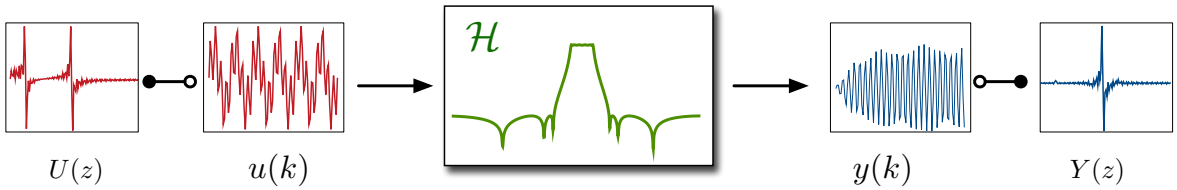
$$H(z) = \frac{0.0495329964 - 0.148598989z^{-2} + \dots}{1 - 2.12060288z^{-1} + 2.7247492z^{-2} + \dots}$$

Time domain



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

Linear digital filters



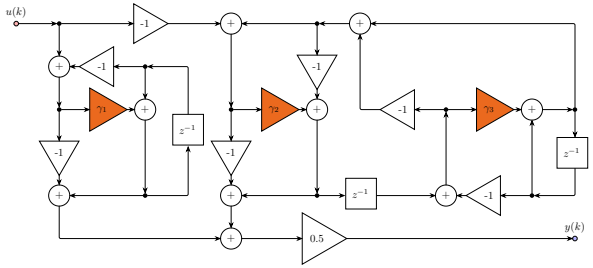
Frequency domain

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

coefficient quantization
in Fixed-Point arithmetic

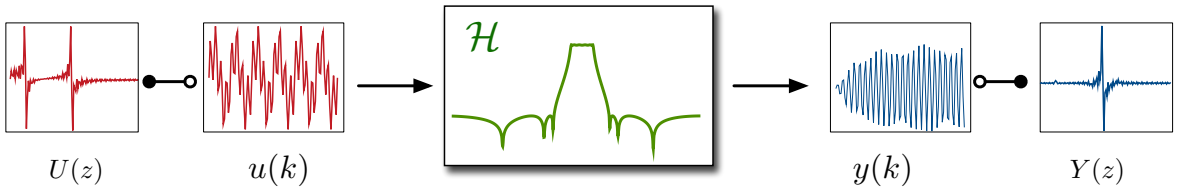
Time domain

Finite-precision
arithmetic operations



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

Linear digital filters



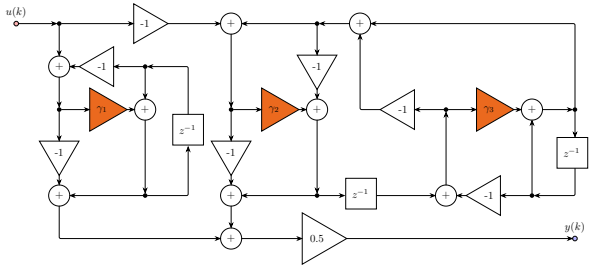
Frequency domain

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

coefficient quantization
in Fixed-Point arithmetic

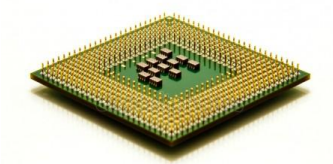
Time domain

Finite-precision
arithmetic operations



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

Circuit / Code

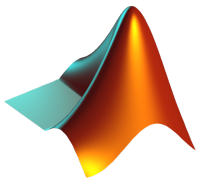


Constraints w.r.t. surface, speed, accuracy ...

Reliable and accurate digital filters

Majority of applications

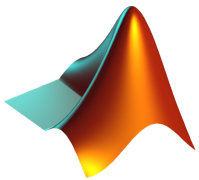
- do not need numerical guarantee
- well-studied subject
- straightforward implementations
- commercial and free tools



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

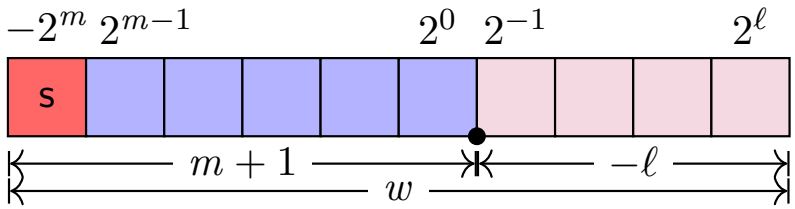
- require numerical guarantees
- have high(er) cost
- limited literature
- require expert knowledge from engineers
- no out-of-box solution

Context: safety-critical applications

Goal: generate Fixed-Point codes reliable by construction

Means: interval, floating-point, multiple precision arithmetics

Fixed-Point Arithmetic



w	wordlength
m	Most Significant Bit
ℓ	Least Significant Bit

Problem setting for MSB/LSB choice:

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

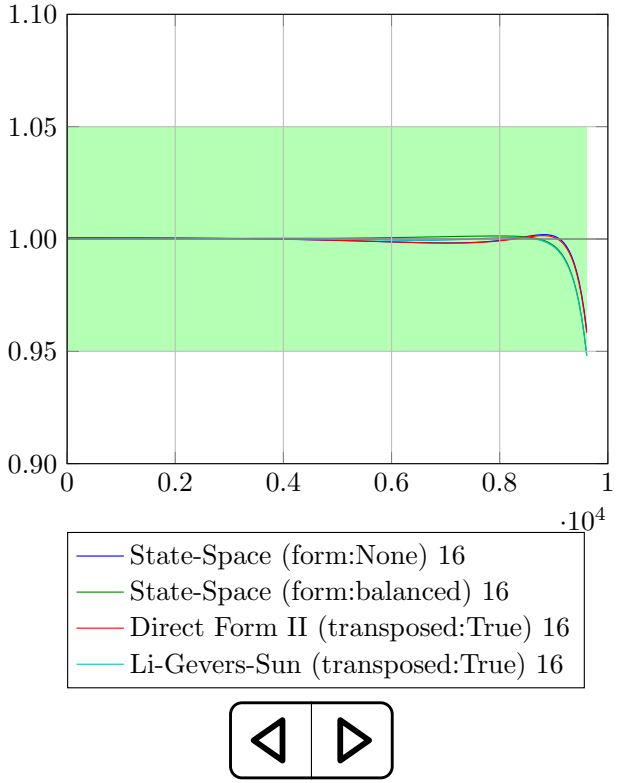
Example: a 15^{th} order lowpass filter

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

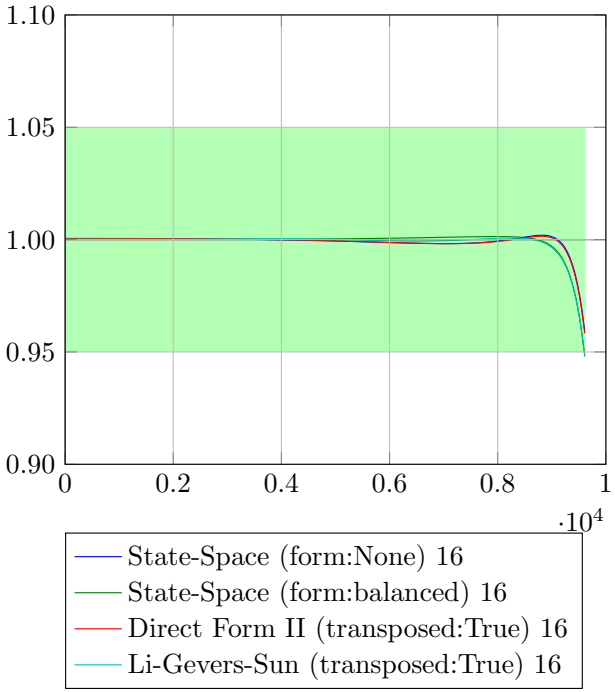
Coefficient quantization



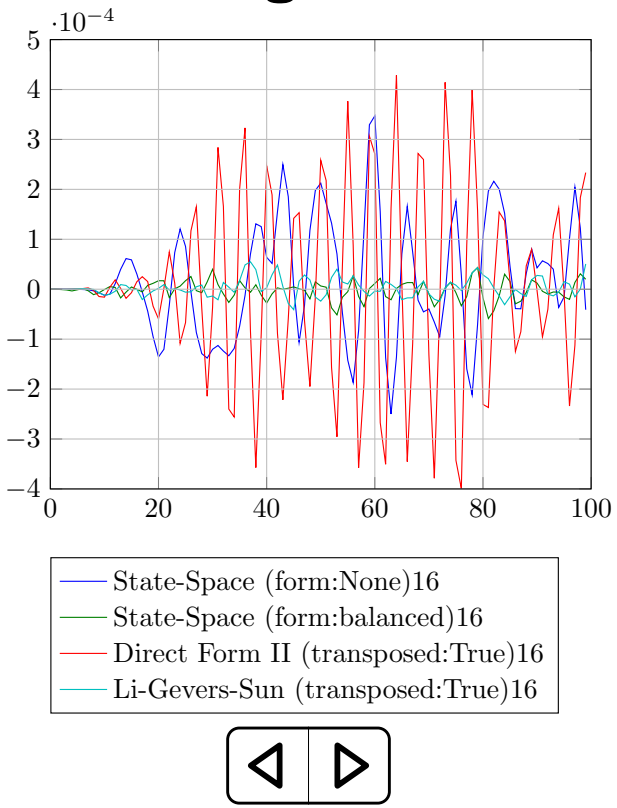
Example: a 15th order lowpass filter

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 "

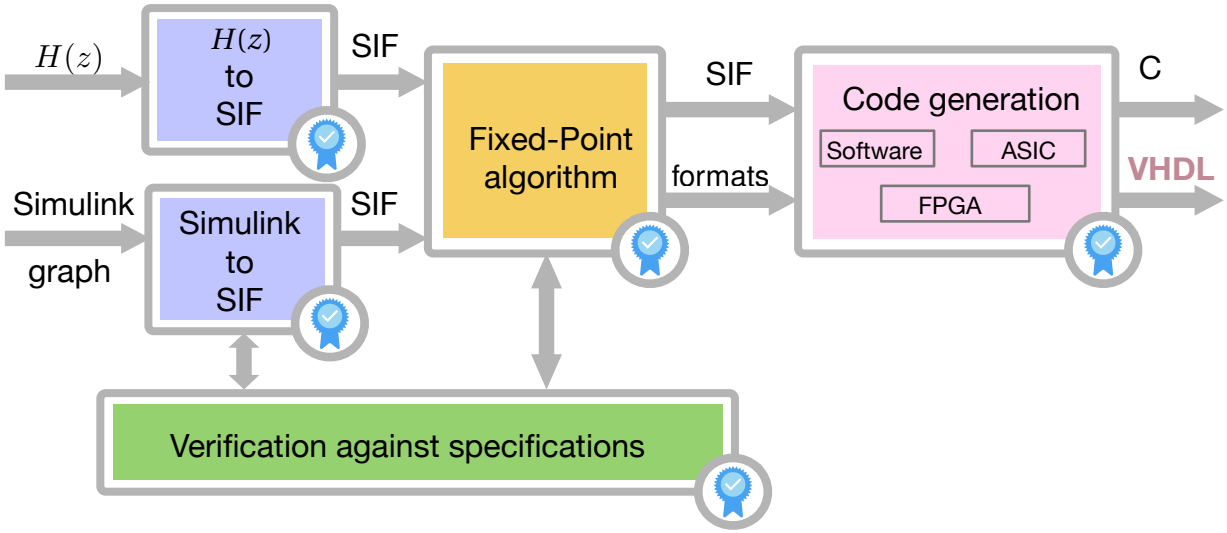
Available at `https://github.com/fixif`

Joint work with

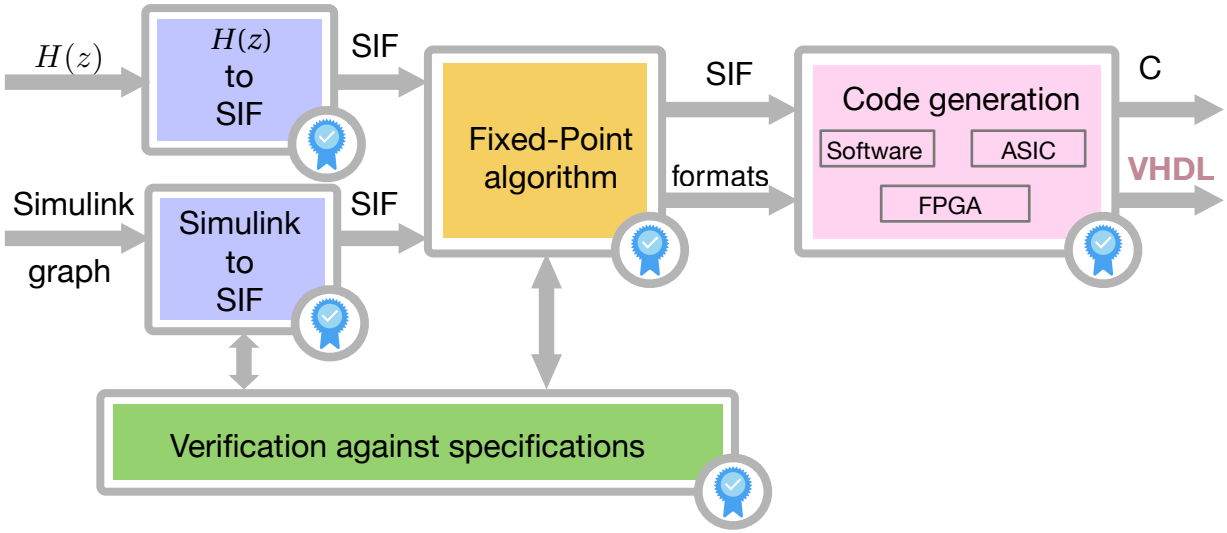
Thibault Hilaire
`thibault.hilaire@lip6.fr`

Christoph Lauter
`chirstoph.lauter@lip6.fr`

Compiler overview

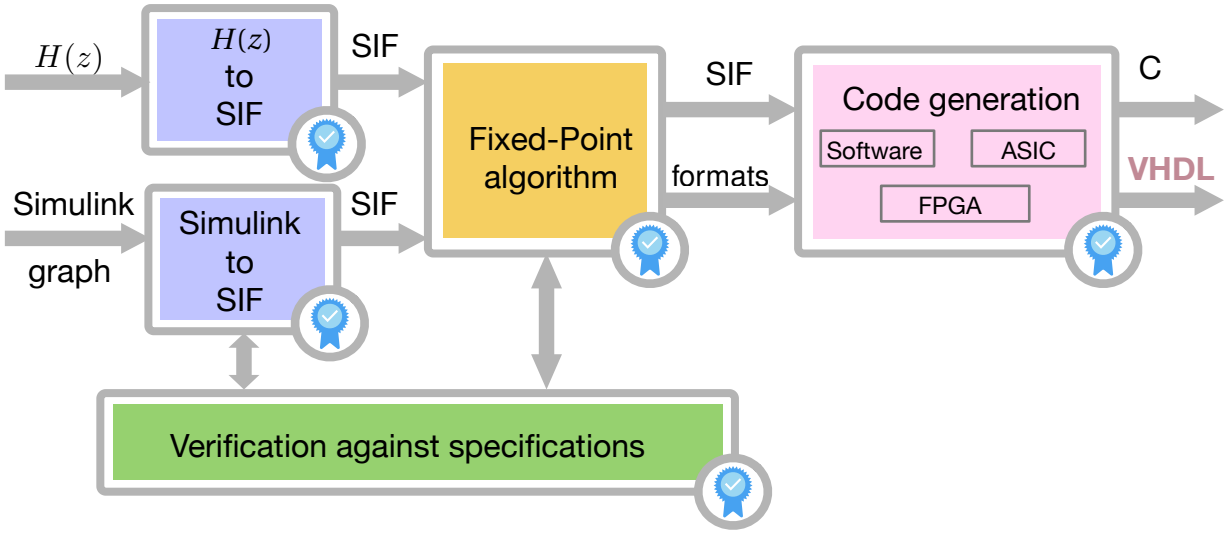


Compiler overview



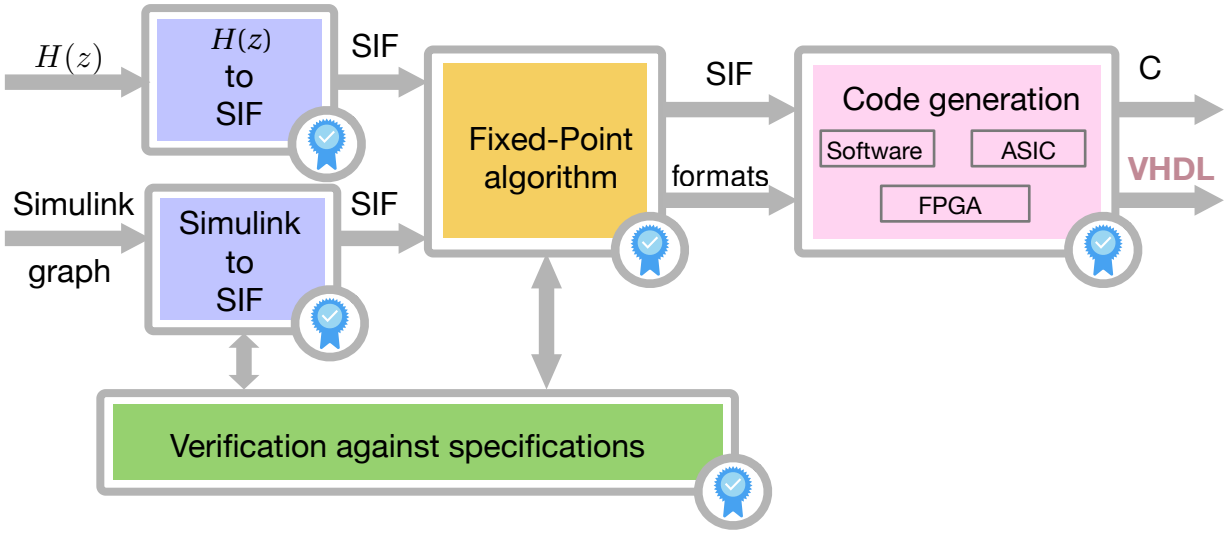
- Internal representation: SIF (matrix-based description)

Compiler overview



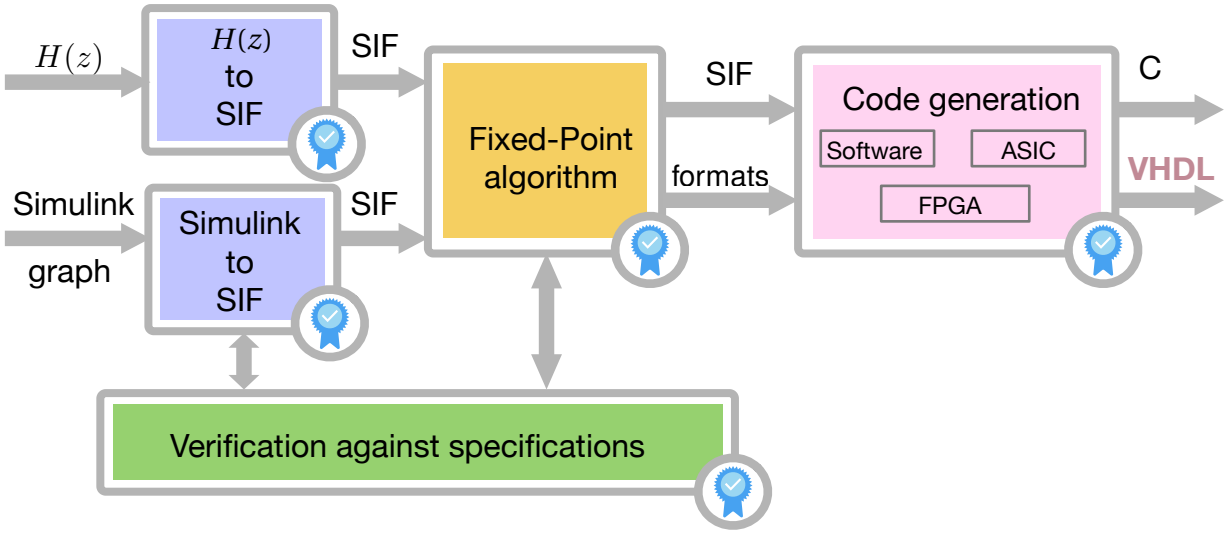
- Internal representation: SIF (matrix-based description)
- Rounding errors: unified, reliable and fast FxP algorithm generation

Compiler overview



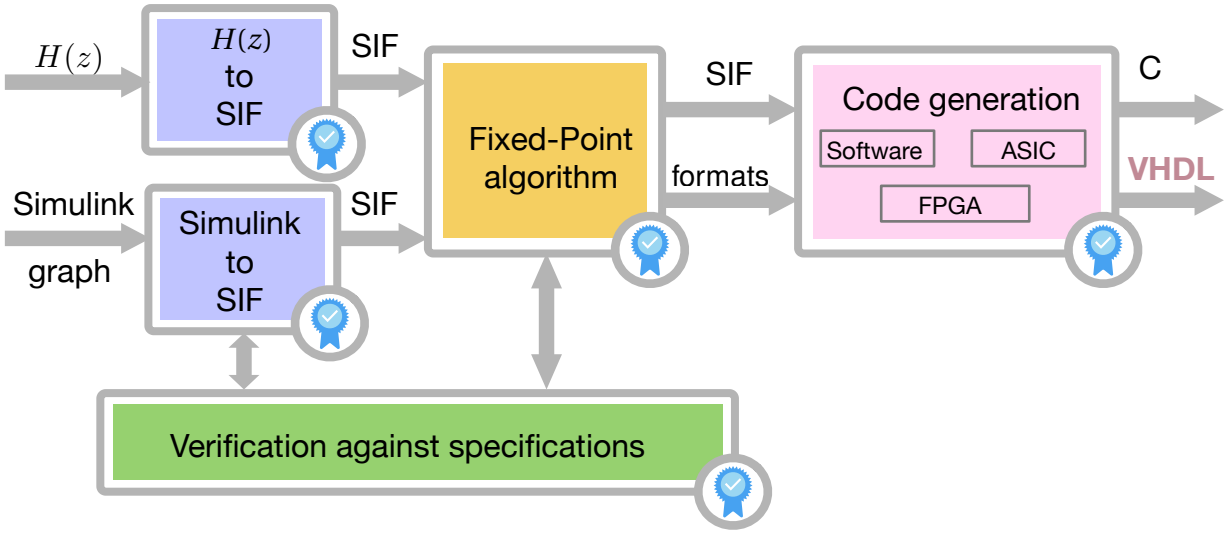
- Internal representation: SIF (matrix-based description)
- Rounding errors: unified, reliable and fast FxP algorithm generation
- Quantization errors: rigorous verification against frequency specs

Compiler overview



- Internal representation: SIF (matrix-based 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 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

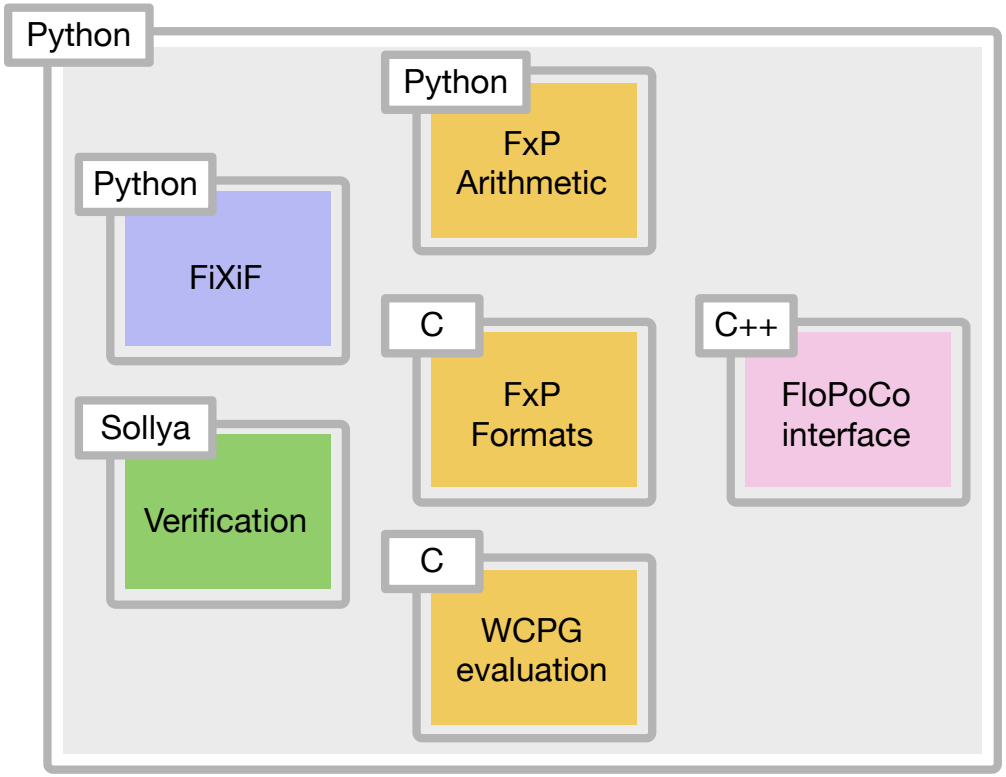
- Python
- Collection of classes

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

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

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

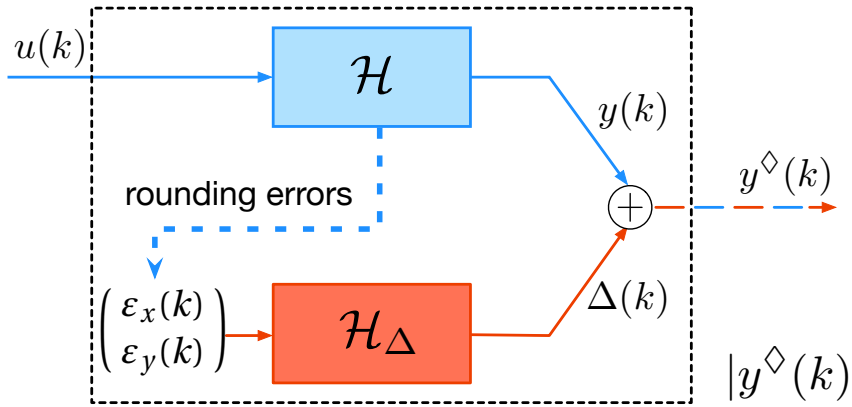
Take a State-Space algorithm and add rounding

$$\mathcal{H}^{\diamondsuit} \left\{ \begin{array}{rcll} \boldsymbol{x}^{\diamondsuit}(k+1) & = & \boldsymbol{A}\boldsymbol{x}^{\diamondsuit}(k) + \boldsymbol{B}\boldsymbol{u}(k) & + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^{\diamondsuit}(k) & = & \boldsymbol{C}\boldsymbol{x}^{\diamondsuit}(k) + \boldsymbol{D}\boldsymbol{u}(k) & + \boldsymbol{\varepsilon}_y(k) \end{array} \right.$$

with

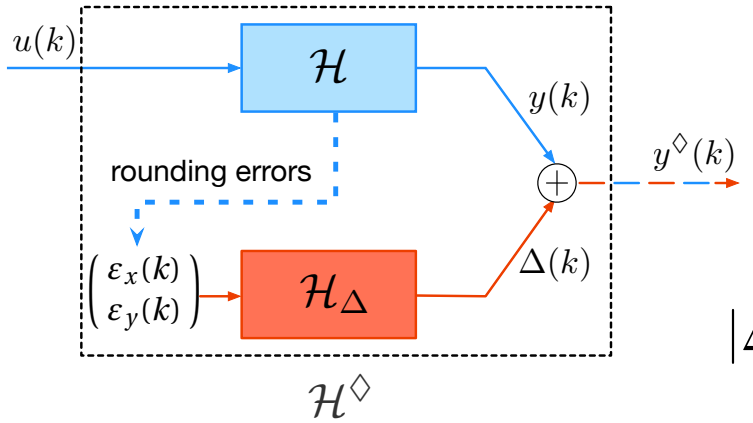
$$|\boldsymbol{\varepsilon}_x(k)| \leq 2^{\boldsymbol{\ell}_x} \qquad \text{and} \qquad |\boldsymbol{\varepsilon}_y(k)| \leq 2^{\boldsymbol{\ell}_y}$$

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



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

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

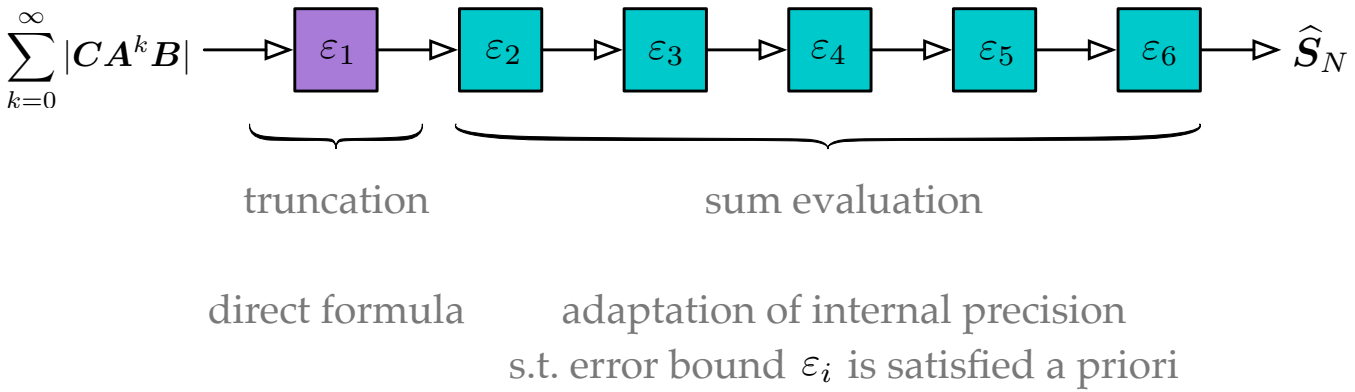
Worst-Case Peak Gain

- 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

$$\left| \sum_{k=0}^{\infty} \left| \boldsymbol{C}\,\boldsymbol{A}^k\,\boldsymbol{B} \right| - \widehat{\boldsymbol{S}}_N \right| < \varepsilon$$

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

FiXiF – 2

Determine the Fixed-Point Formats

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

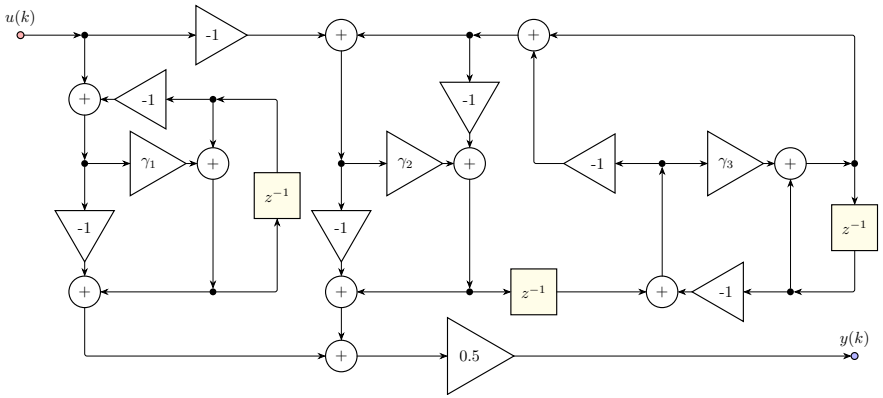
```
Wordlength=[16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16]
Additional steps:  1
LSB: [-13 -13 -14 -14 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15]
Errors:  [[ 0.00086691 0.00121799 0.00152309 0.00172836 0.00158205
0.00121137 0.00079471 0.00048683 0.00029988 0.0001935 0.00013235
0.00009573 0.00007329 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.
```

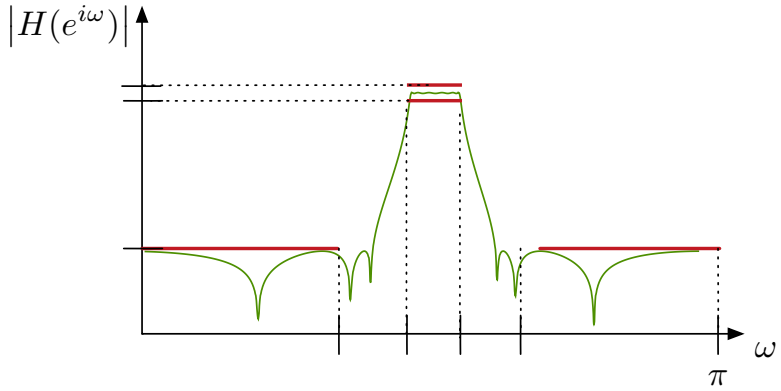
Verification of specifications

Implementation :



$\gamma_1 = 89 \cdot 2^{-8}, \gamma_1 = 43 \cdot 2^{-7}, \gamma_1 = 11 \cdot 2^{-7}$

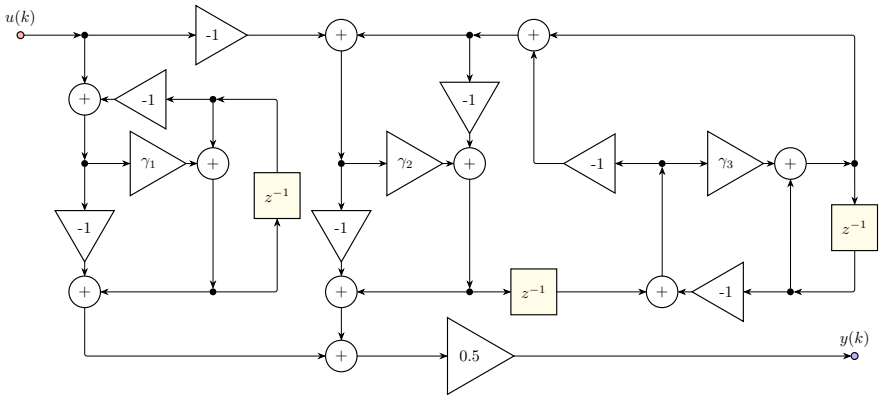
Verify the specifications:



no false positives

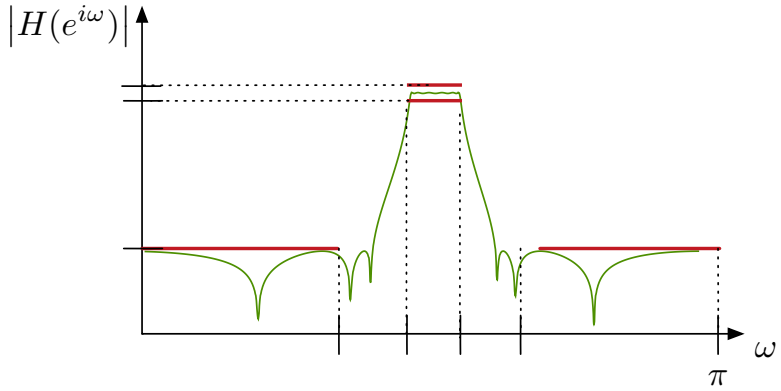
Verification of specifications

Implementation :



$\gamma_1 = 89 \cdot 2^{-8}, \gamma_1 = 43 \cdot 2^{-7}, \gamma_1 = 11 \cdot 2^{-7}$

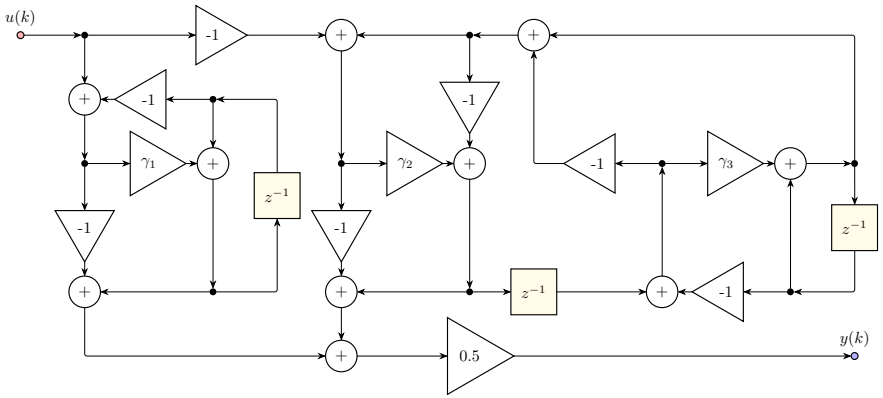
Verify the specifications:



no false positives

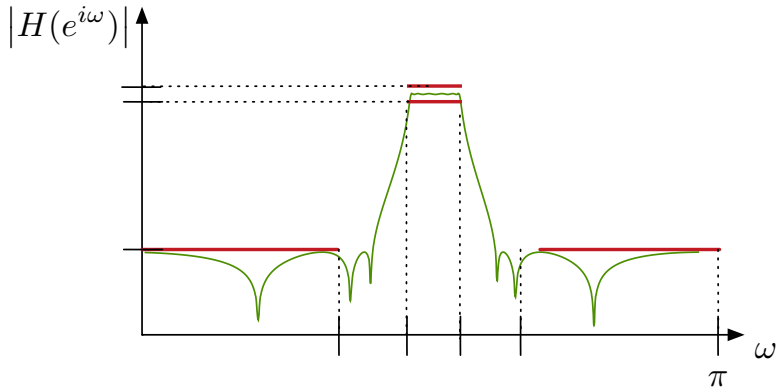
Verification of specifications

Implementation :



$\gamma_1 = 89 \cdot 2^{-8}, \gamma_1 = 43 \cdot 2^{-7}, \gamma_1 = 11 \cdot 2^{-7}$

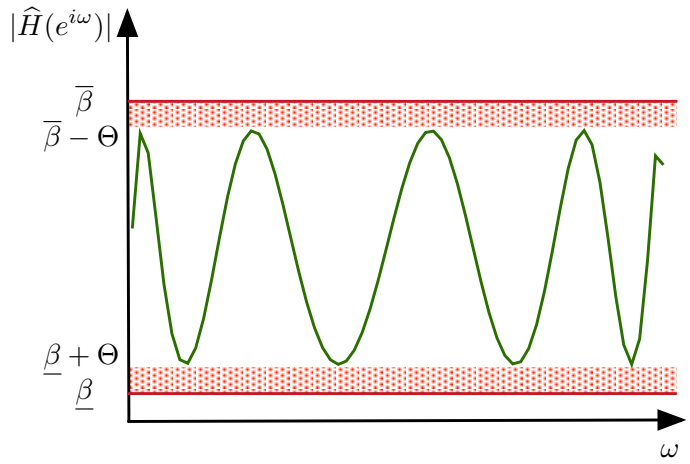
Verify the specifications:



no false positives

Solution [V-Lauter-Hilaire'17]

- MP approximation of $\widehat{\boldsymbol{H}}(z)$
  $\rightarrow$ error bound [Balakrishnyan'92]:
  $\left|\left(H - \widehat{\boldsymbol{H}}\right)(e^{i\omega})\right| \leq \Theta$
- Verification of $\widehat{\boldsymbol{H}}(z)$
  $\rightarrow$ proof of non-negativity of a polynomial



FiXiF – 3

Example of the verification

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

FiXiF – 3

Example of the 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)

⋮
```

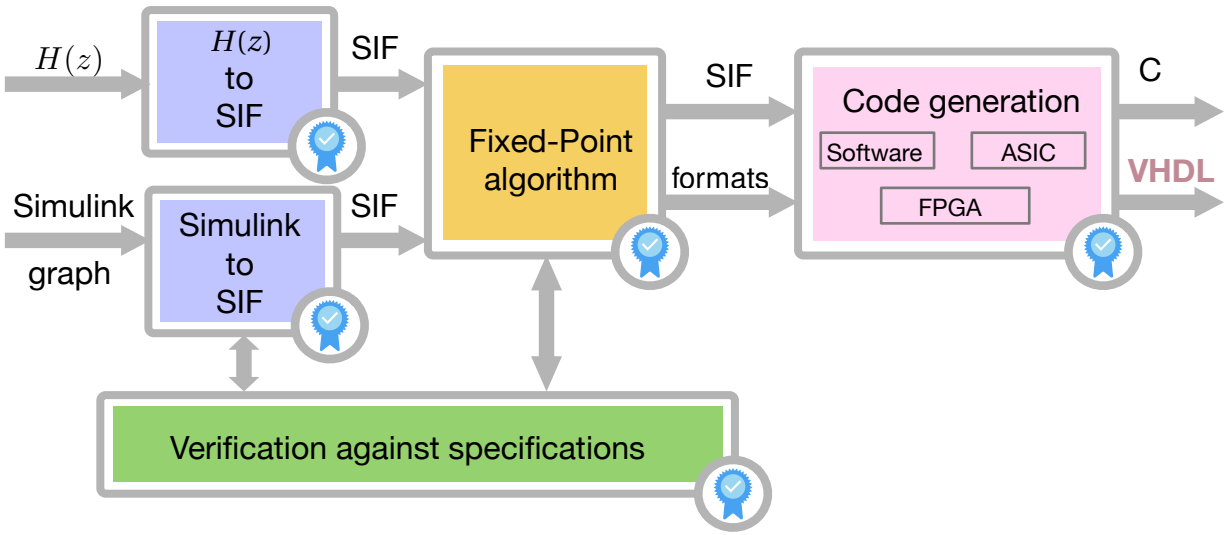
Behind the scenes

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

Beyond FiXiF

Verification of frequency specifications:

- software-hardware co-design of FIR filters[1]

Worst-Case Peak Gain evaluation:

- digital filters computed just right on FPGAs[2]

Internal representation SIF

- error analysis for multiplierless FFT[3]

Formal proofs of the code generator

- Coq proofs of the WCPG algorithm[4]

[1] with M. Kumm (Kassel University) and S. Filip (Inria Rennes)
[2] with F. de Dinechin (INSA Lyon) and M. Istoan (Imperial College)
[3] with F. Qureshi and J. Takala (Tampere University)
[4] with S. Boldo and D. Gallois (Inria Paris)

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`

Verification of a transfer function

We need to show that $\forall z = e^{i\omega}, \omega \in \Omega \subseteq [0, \pi]$

$$\underline{\beta} \leq |H(z)| \leq \overline{\beta}$$

Verification of a transfer function

We need to show that $\forall z = e^{i\omega}, \omega \in \Omega \subseteq [0, \pi]$

$$\underline{\beta} \leq |H(z)| \leq \overline{\beta}$$

...which boils down to showing that $\forall \xi \in \Xi \subseteq [0, 1]$

$$f(\xi) \geq 0$$

with $f \in \mathbb{R}[\xi]$.

Verification that f is positive or zero

Our algorithm is based on:

- the Sturm's technique that gives the number of real roots of a polynomial
- a subdivision on sub-intervals
- evaluations in Multiple-Precision Interval Arithmetic