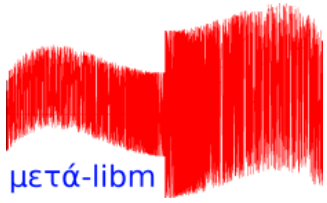
Towards reliable implementation of digital filters

How digital signal processing and computer arithmetic meet

Anastasia Volkova

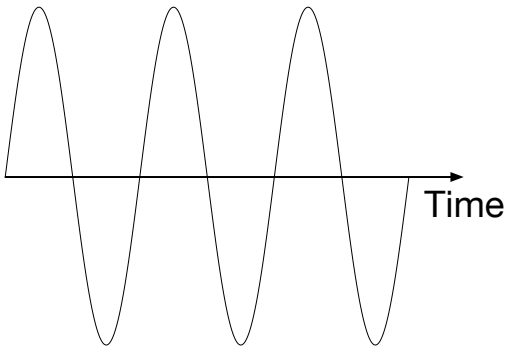
Director: Jean-Claude Bajard

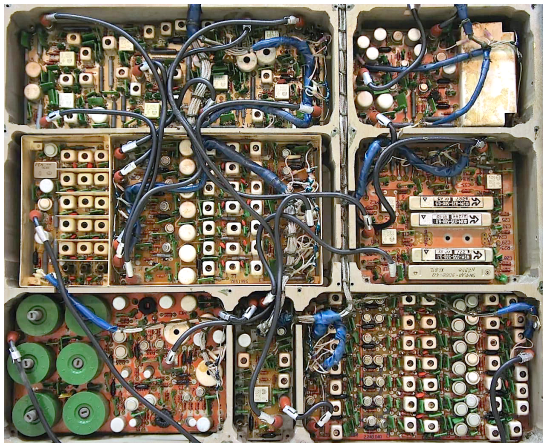
Advisors: Thibault Hilaire, Christoph Lauter



Signals are everywhere

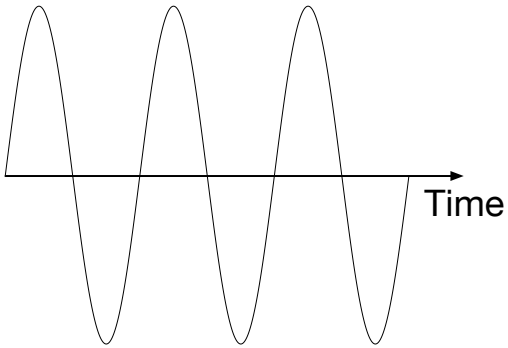
- Analog signals: continuous-time

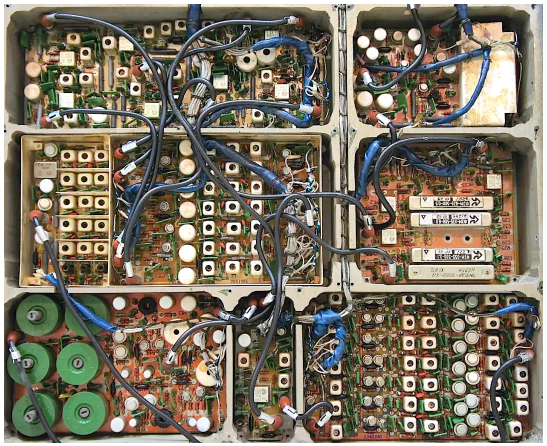




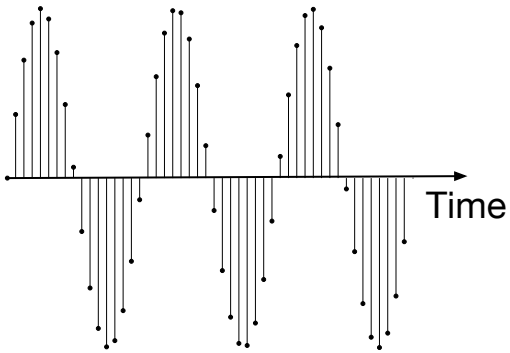
Signals are everywhere

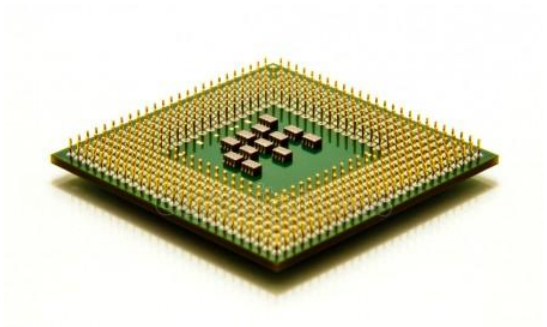
- Analog signals: continuous-time





- Digital signals: discrete-time





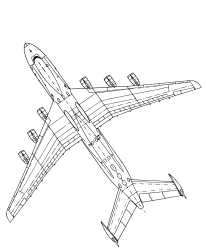
Applications: reliable systems

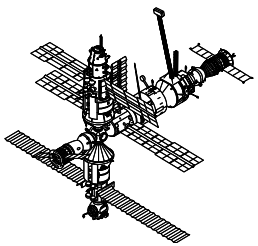
Digital filters:

Algorithms that transform digital signals

- Do not need guarantee in the majority of applications

- A guarantee is necessary in other applications.





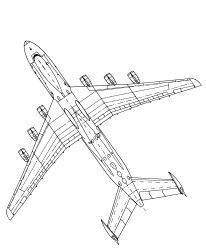
Applications: reliable systems

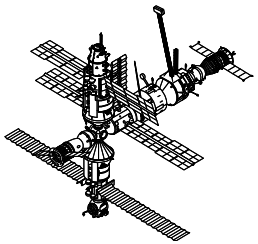
Digital filters:

Algorithms that transform digital signals

- Do not need guarantee in the majority of applications

- A guarantee is necessary in other applications.

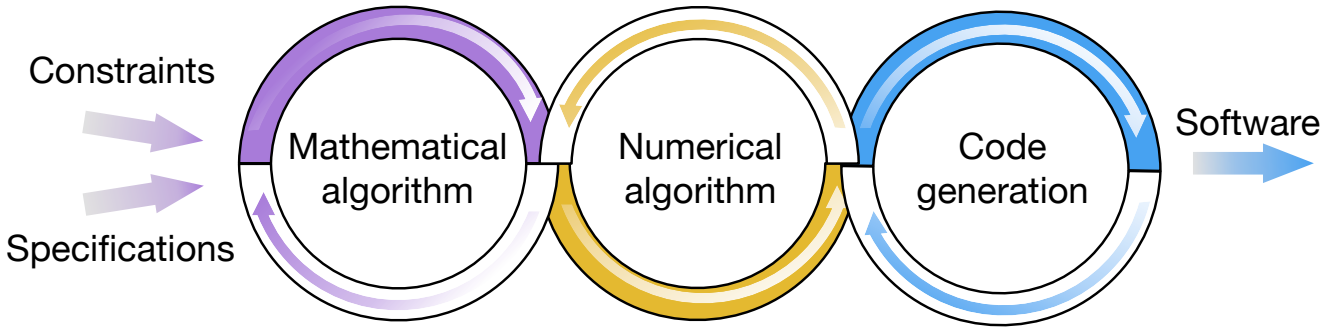




We are interested in guarantees related to the implementation of numerical algorithms, especially in the embedded systems.

Automation

The implementation is done in several steps:



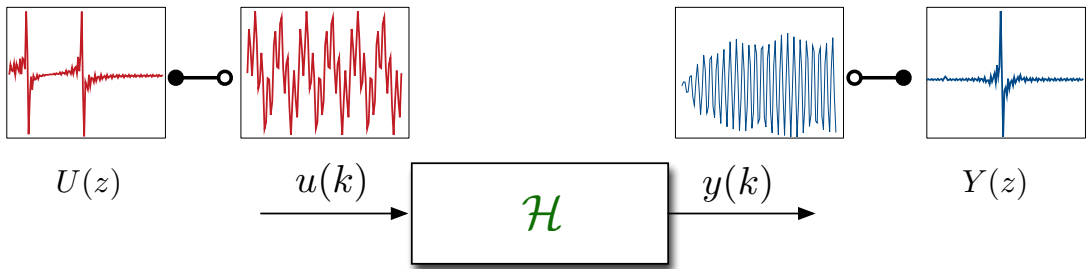
Numerous constraints:

- performance
- surface
- accuracy
- energy consumption
- memory
- etc.

We are interested in the **automated** process of **reliable** implementation.

Filters

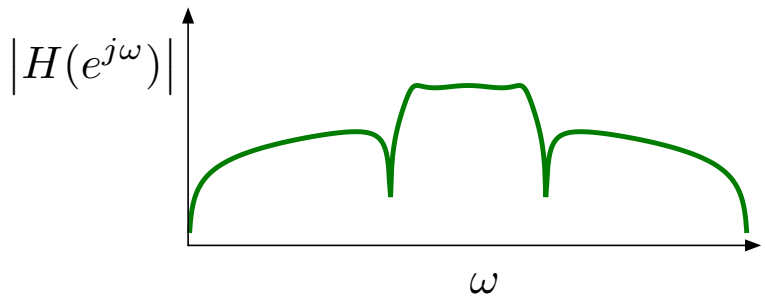
Frequency domain



Linear Time-Invariant filters $\mathcal{H}$: transformation of **spectral** properties

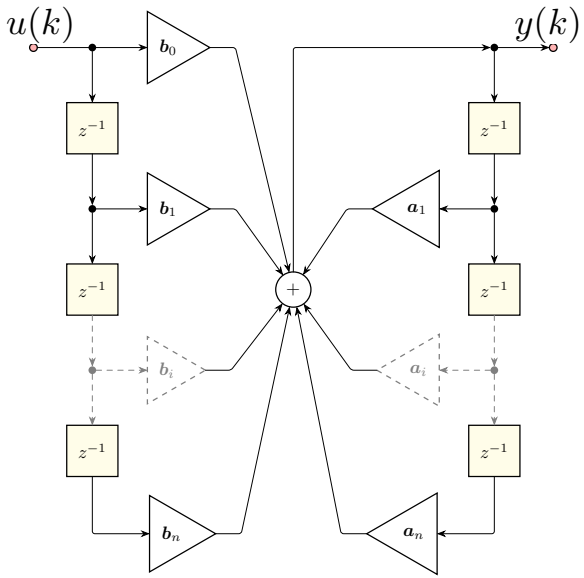
Transfer function $H(z)$, $z \in \mathbb{C}$

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



Filters

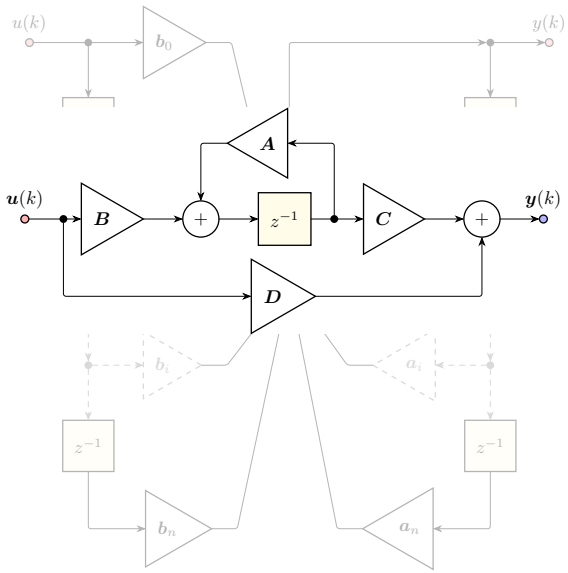
Time domain



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

Filters

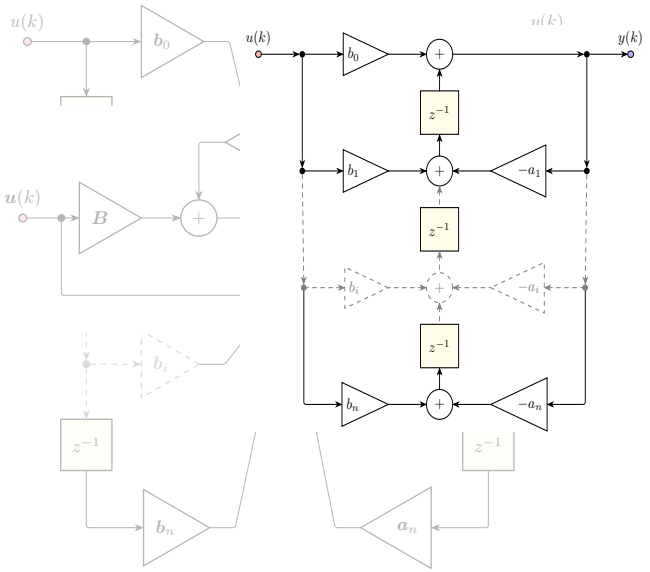
Time domain



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

Filters

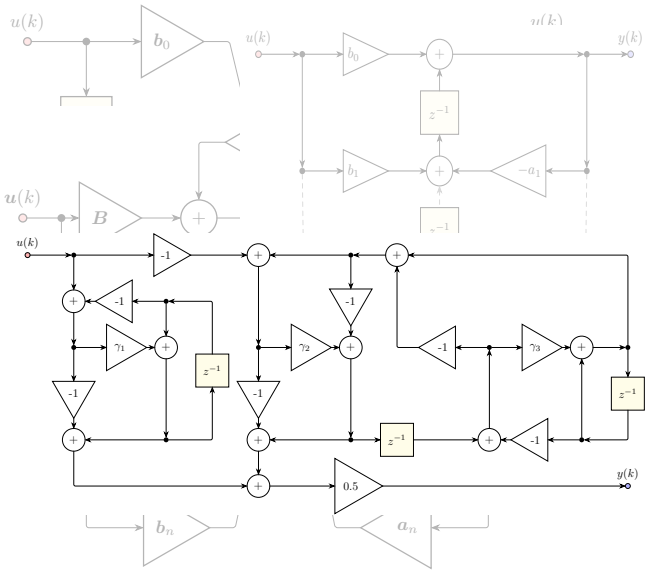
Time domain



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

Filters

Time domain



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

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

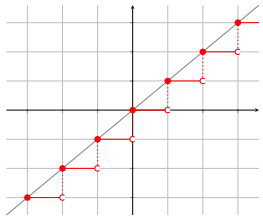
Mathematically speaking, different algorithms compute the same output.

Numerical algorithms

Choice of the arithmetic and its parameters determines the numerical quality of the implemented algorithm.

Why?

- the signals are discrete in their value
- the instructions for the evaluation can induce errors
- the propagation and compensation of errors depend on the instructions

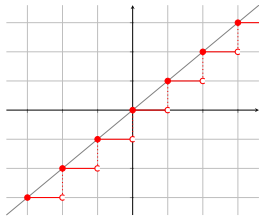


Numerical algorithms

Choice of the arithmetic and its parameters determines the numerical quality of the implemented algorithm.

Why?

- the signals are discrete in their value
- the instructions for the evaluation can induce errors
- the propagation and compensation of errors depend on the instructions

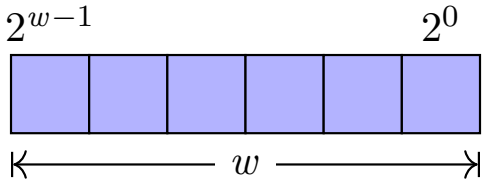


In addition, choice of the arithmetic and its parameters influence:

- the speed of computations
- the surface
- ...

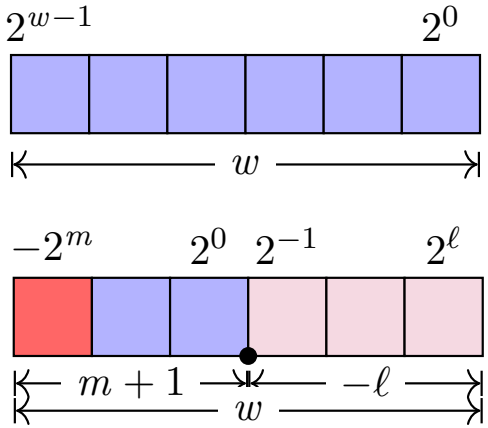
Arithmetics

- Integer arithmetic:
  $y = Y$



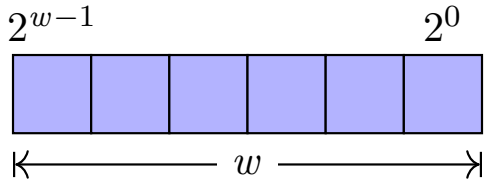
Arithmetics

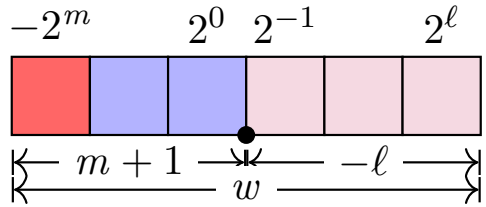
- Integer arithmetic:
  $y = Y$
- Fixed-Point arithmetic:
  $y = Y \cdot 2^{\ell}$
  where ℓ is an *implicit* factor

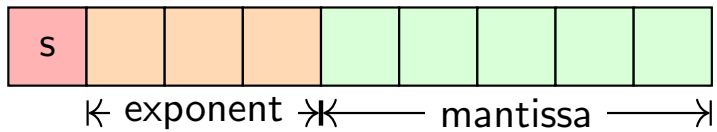


Arithmetics

- 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

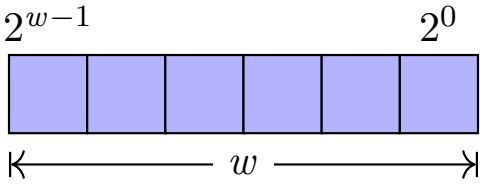


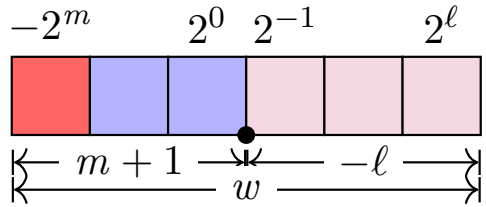


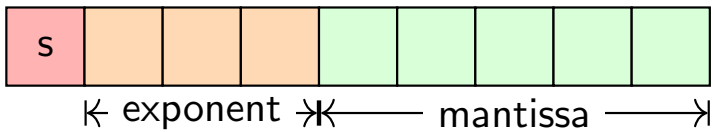


Arithmetics

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

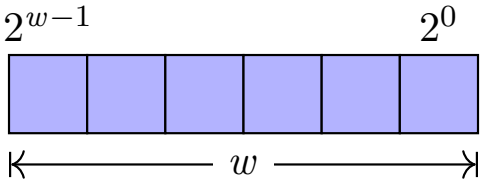


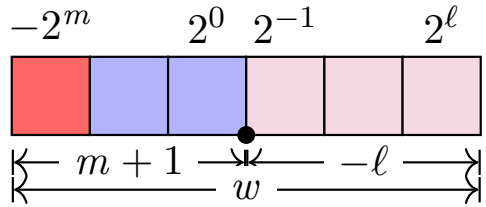


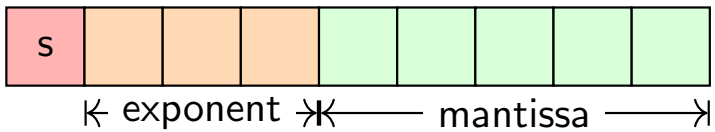


Arithmetics

- 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

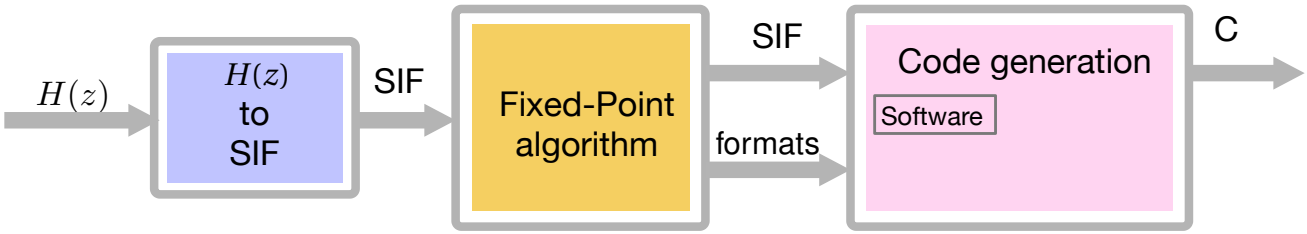
- Interval arithmetic:
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$

  For the error analysis

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

  For the error analysis

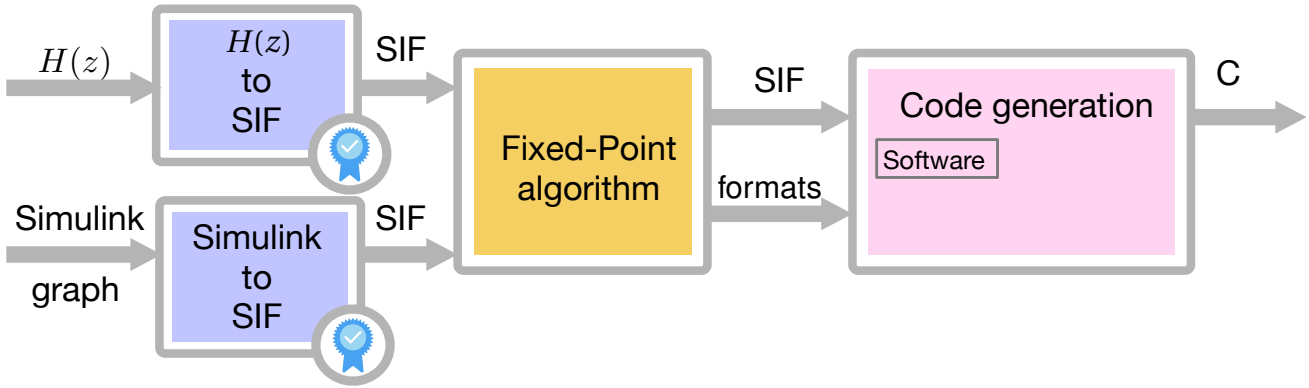
Towards reliable implementation of digital filters



Before this work

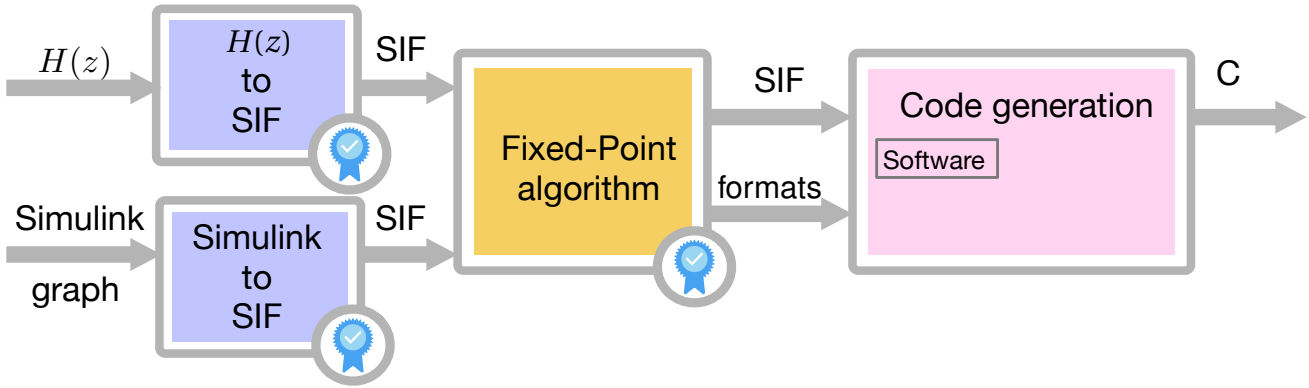
- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
- Fixed-Point implementation: unified, but not reliable
- Code generation: C for the software

Towards reliable implementation of digital filters



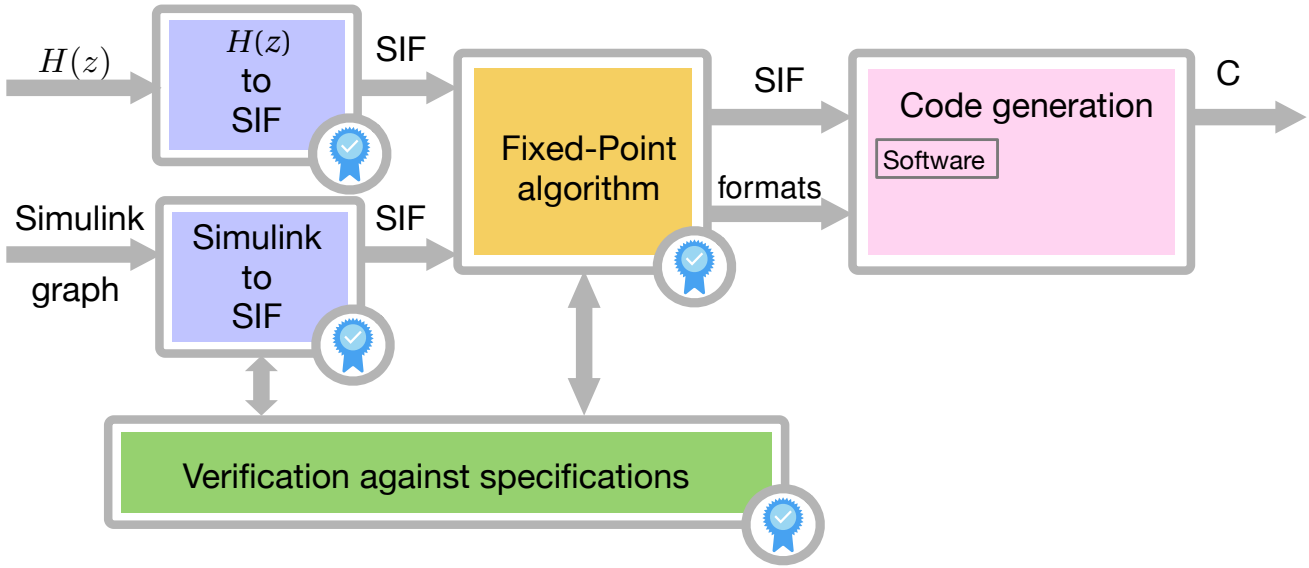
- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
  - **Additional input formats**
- Fixed-Point implementation: unified, but not reliable
- Code generation: C for the software

Towards reliable implementation of digital filters



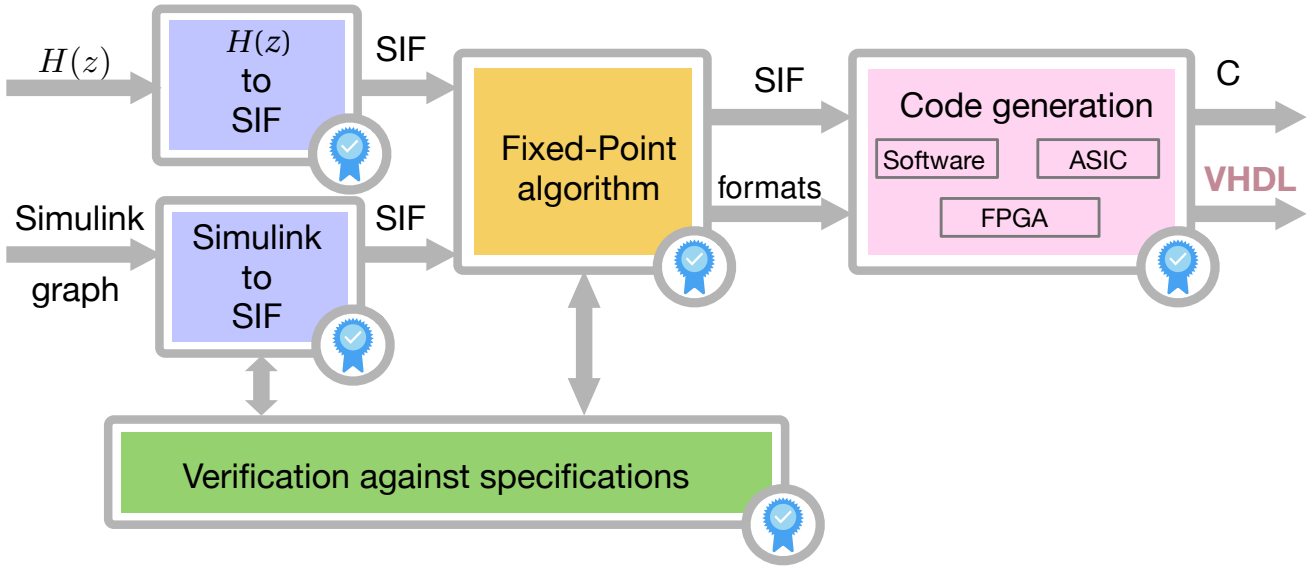
- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
  - **Additional input formats**
- Fixed-Point implementation: unified, **reliable**
- Code generation: C for the software

Towards reliable implementation of digital filters



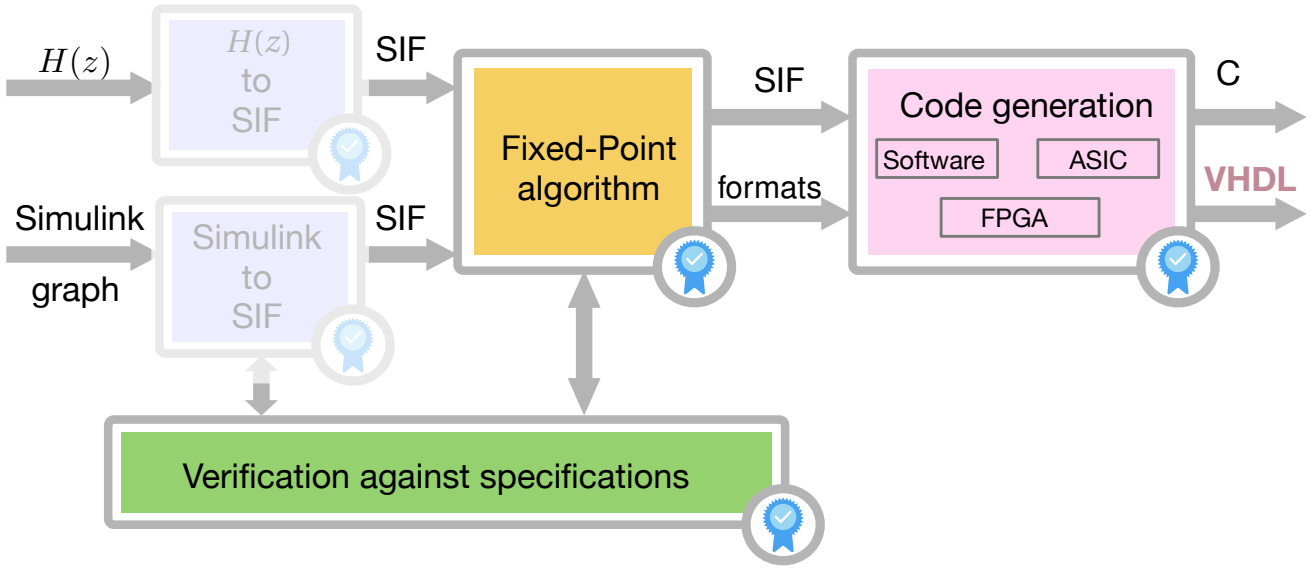
- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
  - **Additional input formats**
- Fixed-Point implementation: unified, **reliable** and **verified**
- Code generation: C for the software

Towards reliable implementation of digital filters



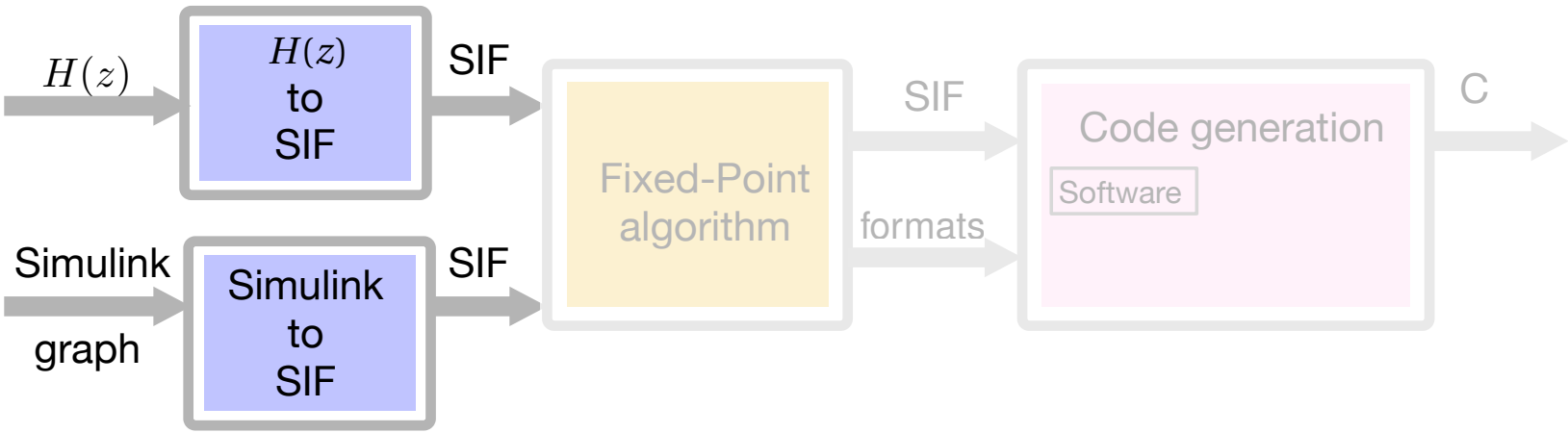
- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
  - **Additional input formats**
- Fixed-Point implementation: unified, **reliable** and **verified**
- Code generation: C for the software, **VHDL for the FPGAs**

Towards reliable implementation of digital filters



- A unified representation of linear filters (SIF)
  - Numerous algorithms were manually converted to SIF
  - **Additional input formats**
- Fixed-Point implementation: unified, **reliable** and **verified**
- Code generation: C for the software, **VHDL for the FPGAs**

Additional input formats



SIF: Specialized Implicit form

- Matrix form based on a system of linear equations.
  - ☞ Different from the graph-based form
- Order of computations is expressed in the equations
- Any LTI filter can be expressed in SIF.

SIF: Specialized Implicit form

- Matrix form based on a system of linear equations.
  - ☞ Different from the graph-based form
- Order of computations is expressed in the equations
- Any LTI filter can be expressed in SIF.

Basic idea:

$$y = m_2 m_1 u$$

SIF: Specialized Implicit form

- Matrix form based on a system of linear equations.
  - ☞ Different from the graph-based form
- Order of computations is expressed in the equations
- Any LTI filter can be expressed in SIF.

Basic idea:

$y \longleftarrow m_2(m_1\, u)$

1: $t \longleftarrow m_1\, u$

2: $y \longleftarrow m_2\, t$

SIF: Specialized Implicit form

- Matrix form based on a system of linear equations.
  - ☞ Different from the graph-based form
- Order of computations is expressed in the equations
- Any LTI filter can be expressed in SIF.

Basic idea:

$y \longleftarrow m_2(m_1 u)$

1: $t \longleftarrow m_1 u$

2: $y \longleftarrow m_2 t$

$$\begin{pmatrix} 1 & 0 \\ -m_2 & 1 \end{pmatrix} \begin{pmatrix} t \\ y \end{pmatrix} = \begin{pmatrix} m_1 \\ 0 \end{pmatrix} u$$

SIF: Specialized Implicit form

- Matrix form based on a system of linear equations.
  - ☞ Different from the graph-based form
- Order of computations is expressed in the equations
- Any LTI filter can be expressed in SIF.

Basic idea:

$y \longleftarrow m_2(m_1 u)$

1: $t \longleftarrow m_1 u$

2: $y \longleftarrow m_2 t$

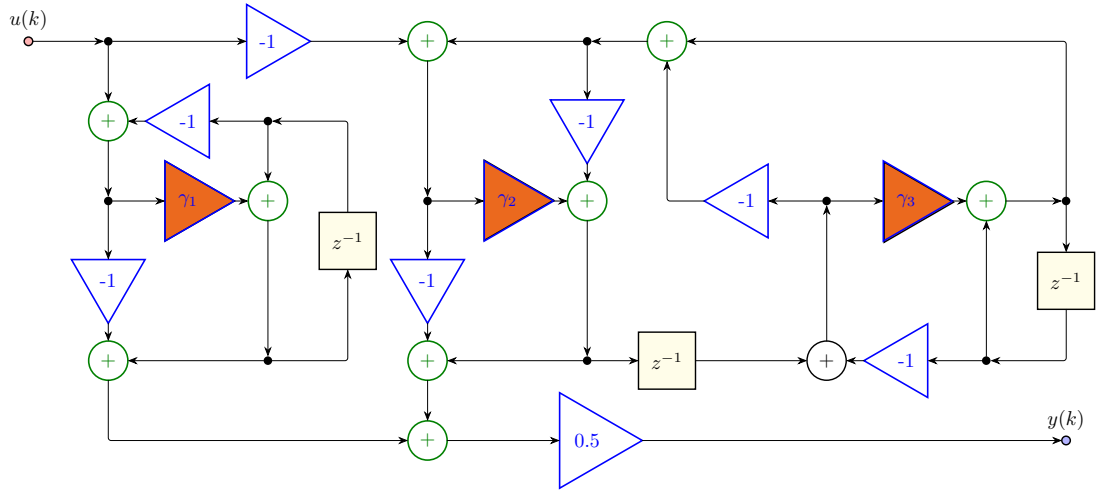
$$\begin{pmatrix} 1 & 0 \\ -m_2 & 1 \end{pmatrix} \begin{pmatrix} t \\ y \end{pmatrix} = \begin{pmatrix} m_1 \\ 0 \end{pmatrix} u$$

Contributions:

- Conversion of Lattice Wave Digital Filters to SIF
- Automatic conversion of Simulink graphs to SIF

Conversion of a Simulink graph

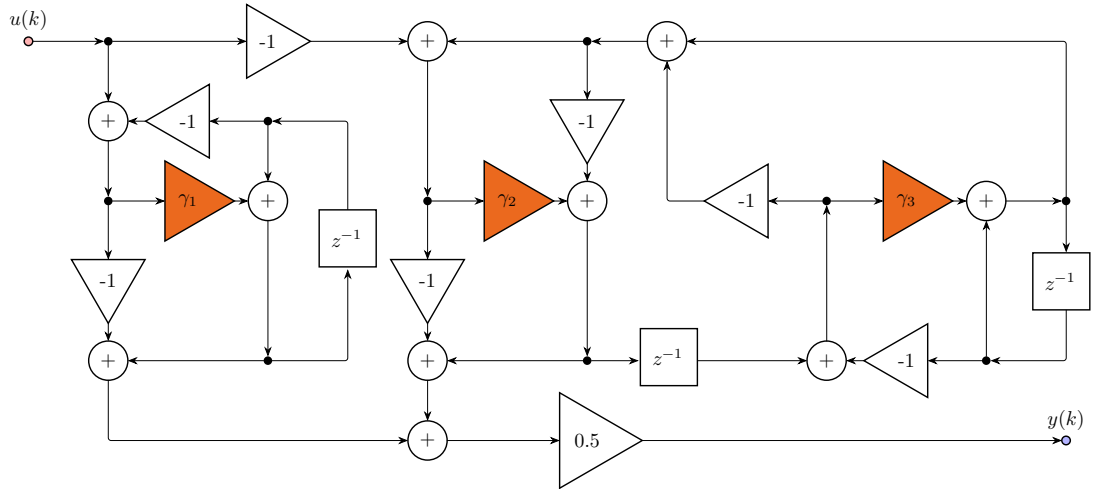
Simulink:



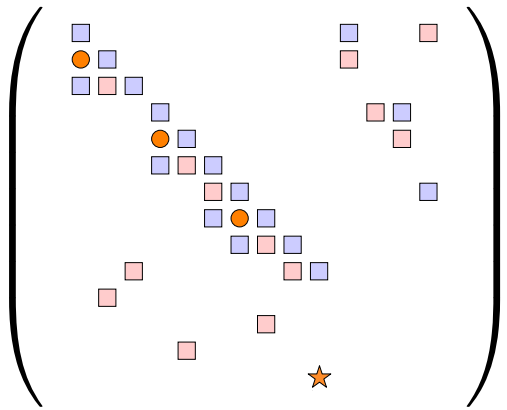
$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

Conversion of a Simulink graph

Simulink:



SIF:



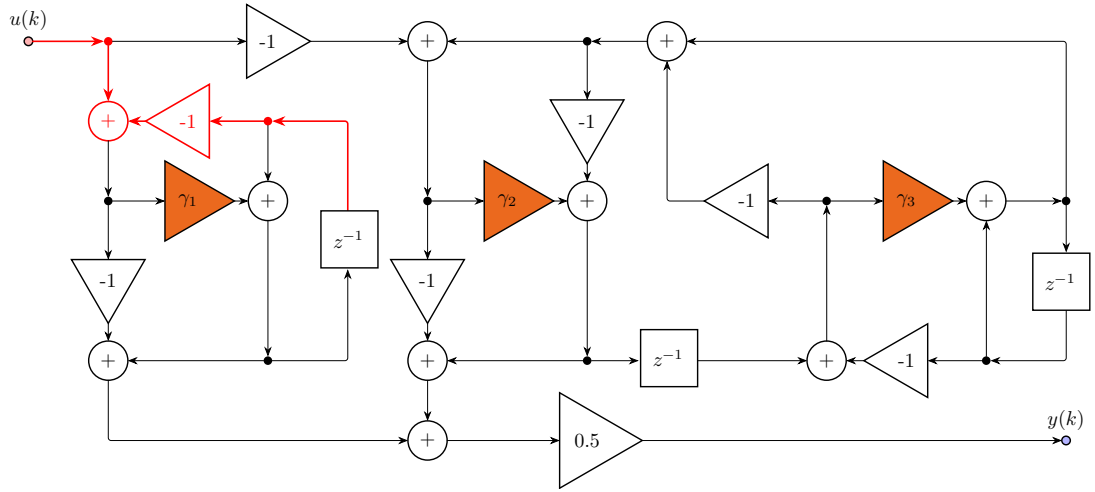
$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

Key idea of our conversion algorithm:

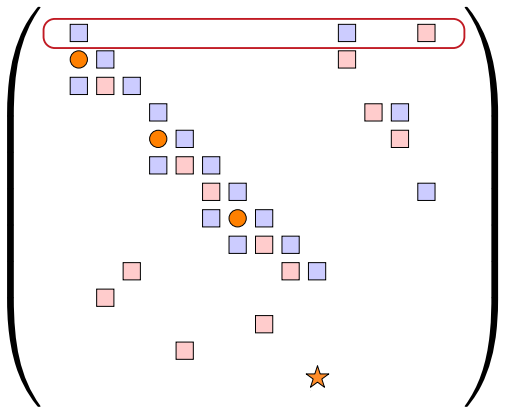
- Identification of inputs, outputs, states, intermediate variables
- Construction of equations
- Topological sort
- Exact copy of the coefficients

Conversion of a Simulink graph

Simulink:



SIF:



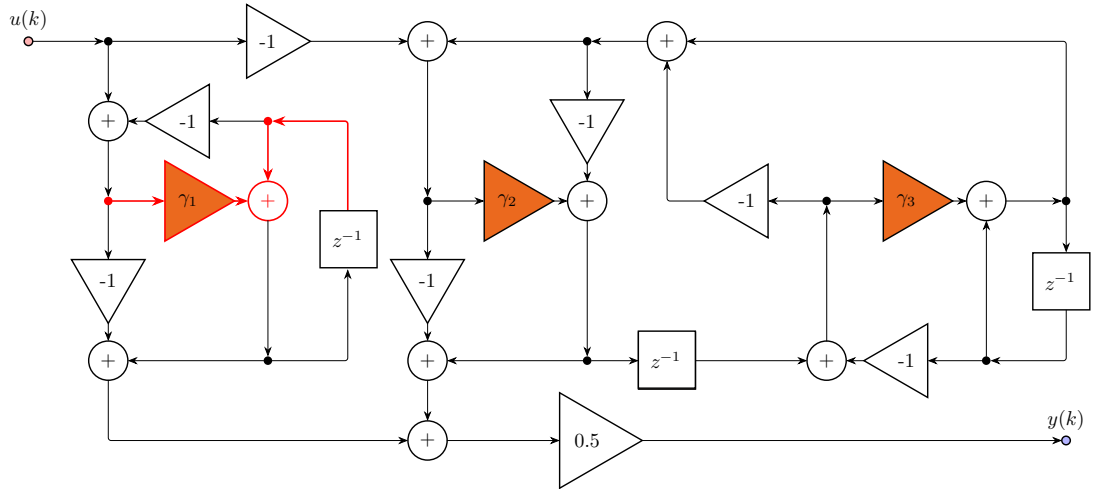
$$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$$

Key idea of our conversion algorithm:

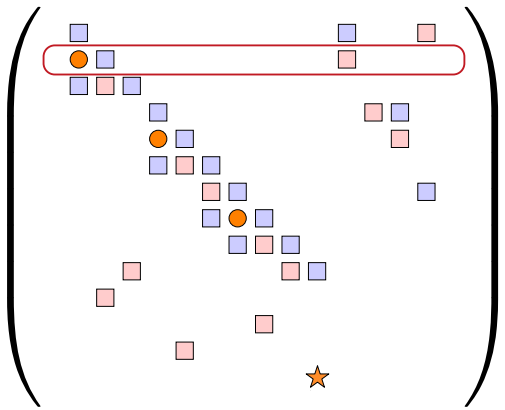
- Identification of inputs, outputs, states, intermediate variables
- Construction of equations
- Topological sort
- Exact copy of the coefficients

Conversion of a Simulink graph

Simulink:



SIF:

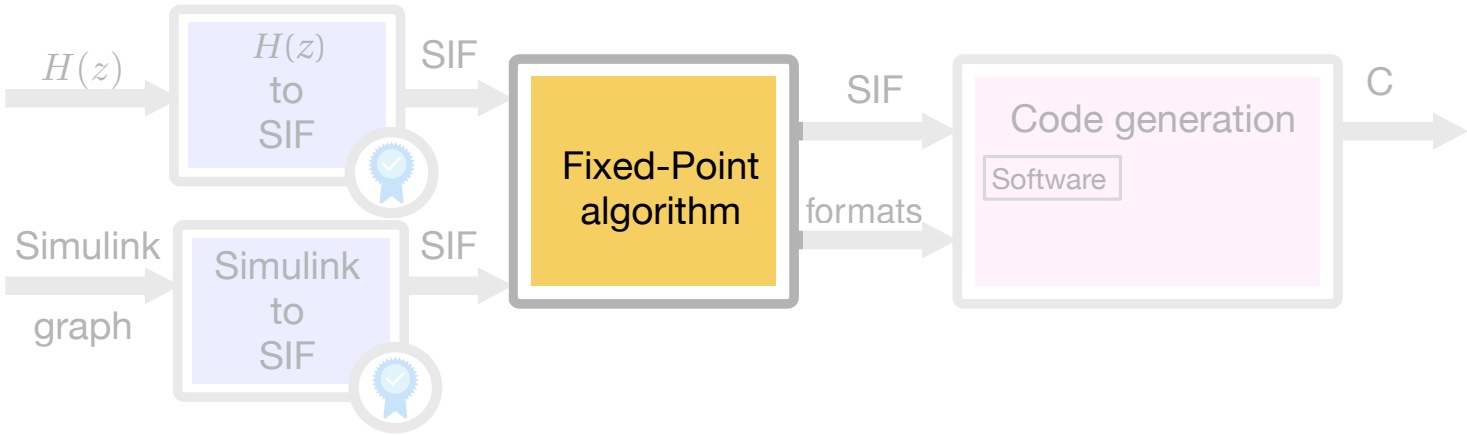


$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

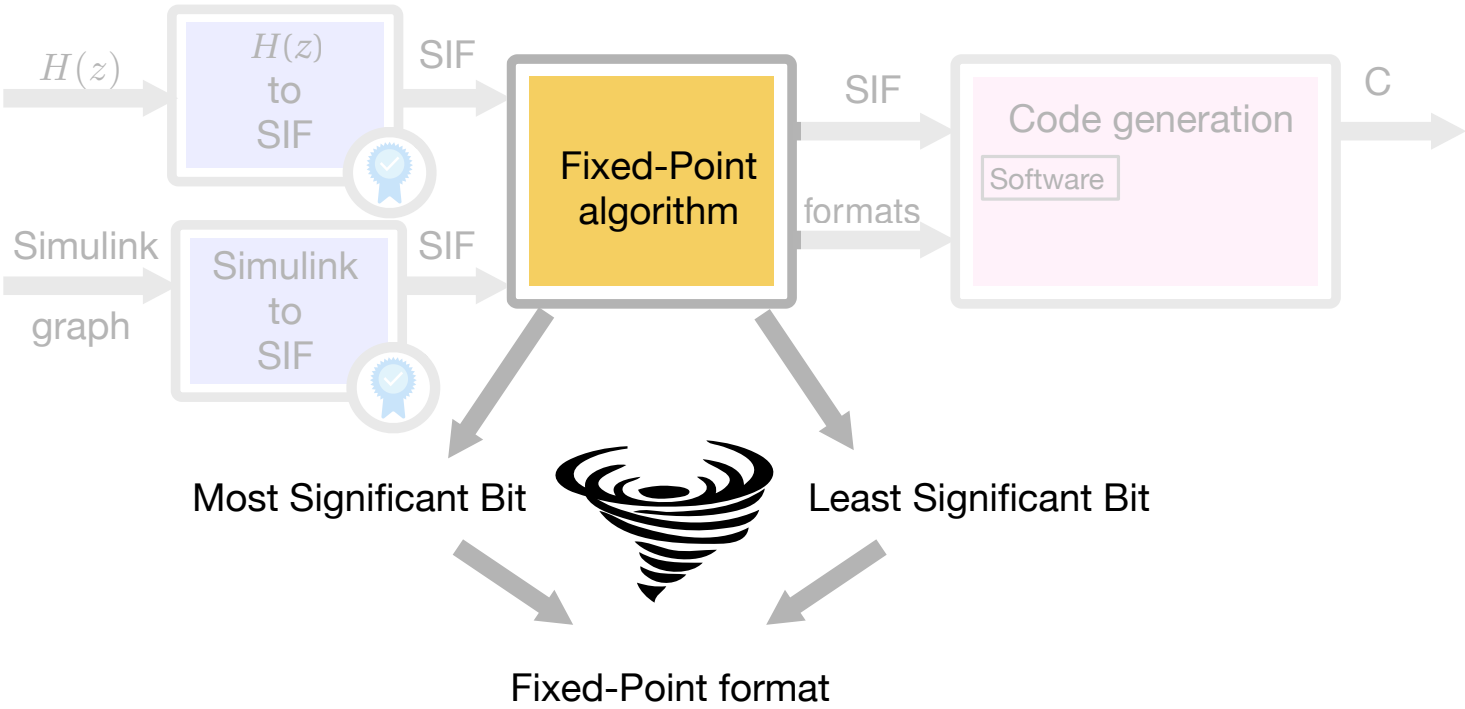
Key idea of our conversion algorithm:

- Identification of inputs, outputs, states, intermediate variables
- Construction of equations
- Topological sort
- Exact copy of the coefficients

Reliable numerical algorithms



Reliable numerical algorithms

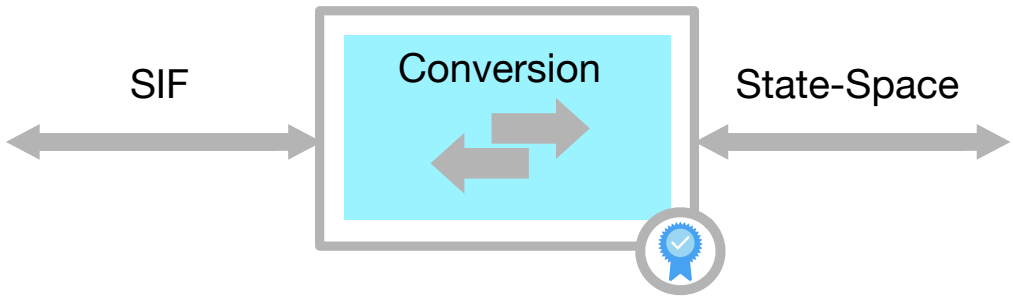


State-space representation

State-space representation of an LTI filter $\mathcal{H}$:

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

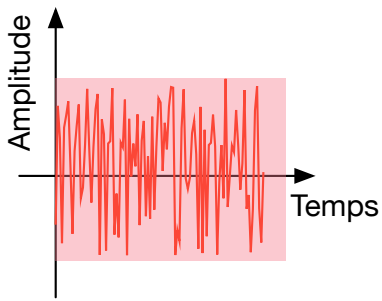
Conversions between SIF and state-space are exact.

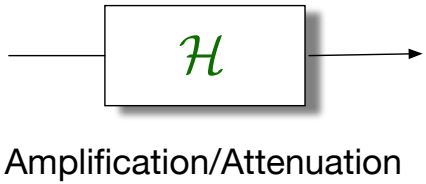


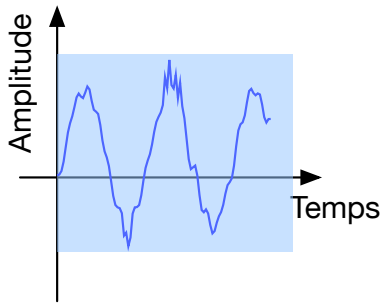
Range of variables

Worst-Case Peak Gain

Input $u(k)$ Stable filter Output $y(k)$







$$\forall k, |u(k)| \leq \bar{u} \qquad \forall k, |y(k)| \leq \langle\langle \mathcal{H} \rangle\rangle \bar{u}$$

Worst-Case Peak Gain: $\langle\langle \mathcal{H} \rangle\rangle = \|h\|_1 = |d| + \sum_{k=0}^{\infty} \left| \boldsymbol{c} \boldsymbol{A}^k \boldsymbol{b} \right|$

Problem of the format choice

- **Constraints**: wordlength w_y is fixed for the output variable y
- **But**: no overflows, rigorous error bounds
- **Bonus**: minimize the errors

Problem of the format choice

- **Constraints**: wordlength w_y is fixed for the output variable y
- **But**: no overflows, rigorous error bounds
- **Bonus**: minimize the errors

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}\,u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problem: find the smallest MSB position m_y such that for all k

$$|y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Problem of the format choice

- **Constraints**: wordlength w_y is fixed for the output variable y
- **But**: no overflows, rigorous error bounds
- **Bonus**: minimize the errors

$$\mathcal{H} \begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}\,u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problem: find the smallest MSB position m_y such that for all k

$$\langle\langle \mathcal{H} \rangle\rangle \bar{u} = |y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Mathematical solution:

$$m_y = \left\lceil \log_2\left(\langle\langle \mathcal{H} \rangle\rangle\, \bar{u}\right) - \log_2\left(1 - 2^{1-w_y}\right)\right\rceil$$

Problem of the format choice

- **Constraints**: wordlength w_y is fixed for the output variable y
- **But**: no overflows, rigorous error bounds
- **Bonus**: minimize the errors

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}\,u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problem: find the smallest MSB position m_y such that for all k

$$\langle\langle\mathcal{H}\rangle\rangle\bar{u} = |y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Mathematical solution:

$$m_y = \left\lceil \log_2\left(\langle\langle\mathcal{H}\rangle\rangle\,\bar{u}\right) - \log_2\left(1 - 2^{1-w_y}\right)\right\rceil$$

Practical solution: control the accuracy of the WCPG such that

$$0 \leq \widehat{m}_y - m_y \leq 1$$

Propagation of the rounding errors

Exact filter $\mathcal{H}$ is:

$$\mathcal{H} \begin{cases} \boldsymbol{x}\ (k+1) = & \boldsymbol{A}\boldsymbol{x}\ (k) + \boldsymbol{b}\,u(k) \\ y\ (k) = & \boldsymbol{c}\boldsymbol{x}\ (k) + du(k) \end{cases}$$

Propagation of the rounding errors

Implemented filter $\mathcal{H}^{\Diamond}$ is:

$$\mathcal{H}^{\Diamond} \left\{ \begin{array}{rcl} \boldsymbol{x}^{\Diamond}(k+1) & = & \Diamond_{m_x}(\boldsymbol{A}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{b}\,u(k)) \\ y^{\Diamond}(k) & = & \Diamond_{m_y}(\boldsymbol{c}\boldsymbol{x}^{\Diamond}(k) + du(k)) \end{array} \right.$$

where $\Diamond_m$ is an operator guaranteeing the faithful rounding:

$$|\Diamond_m(x) - x| \leq 2^{m-w+1}$$

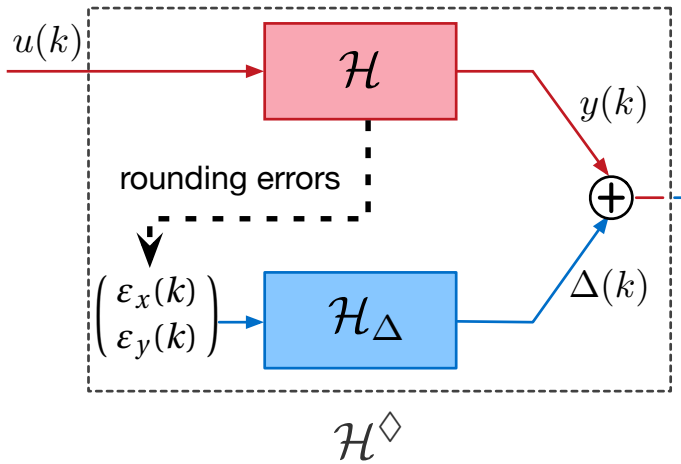
Propagation of the rounding errors

Implemented filter $\mathcal{H}^\Diamond$ is:

$$\mathcal{H}^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) = \boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{b}u(k) + \boldsymbol{\varepsilon}_x(k) \\ y^\Diamond(k) = \boldsymbol{c}\boldsymbol{x}^\Diamond(k) + du(k) + \varepsilon_y(k) \end{cases}$$

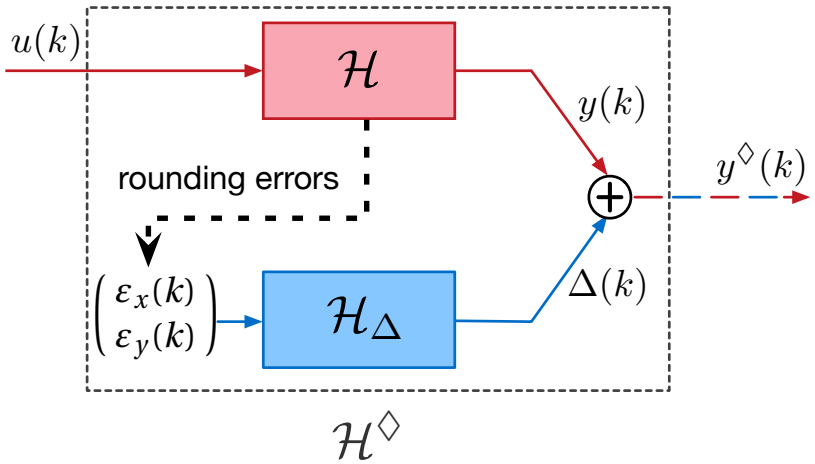
with

$$|\boldsymbol{\varepsilon}_x(k)| \leq 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1} \quad \text{and} \quad |\varepsilon_y(k)| \leq 2^{m_y - w_y + 1}$$



$$|\Delta(k)| \leq \langle\langle \mathcal{H}_\Delta \rangle\rangle \begin{pmatrix} 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1} \\ 2^{m_y - w_y + 1} \end{pmatrix}$$

Reliable algorithm



1. Initial estimation of the MSB m_y for the exact filter $\mathcal{H}$
2. Taking into account the errors induced by the initial formats with the help of the filter $\mathcal{H}_\Delta$, then computation of the new MSB $m_y^{\Diamond}$
3. If $m_y^{\Diamond} = m_y$, return $m_y^{\Diamond}$
   else $m_y \leftarrow m_y + 1$ and return to the step 2.

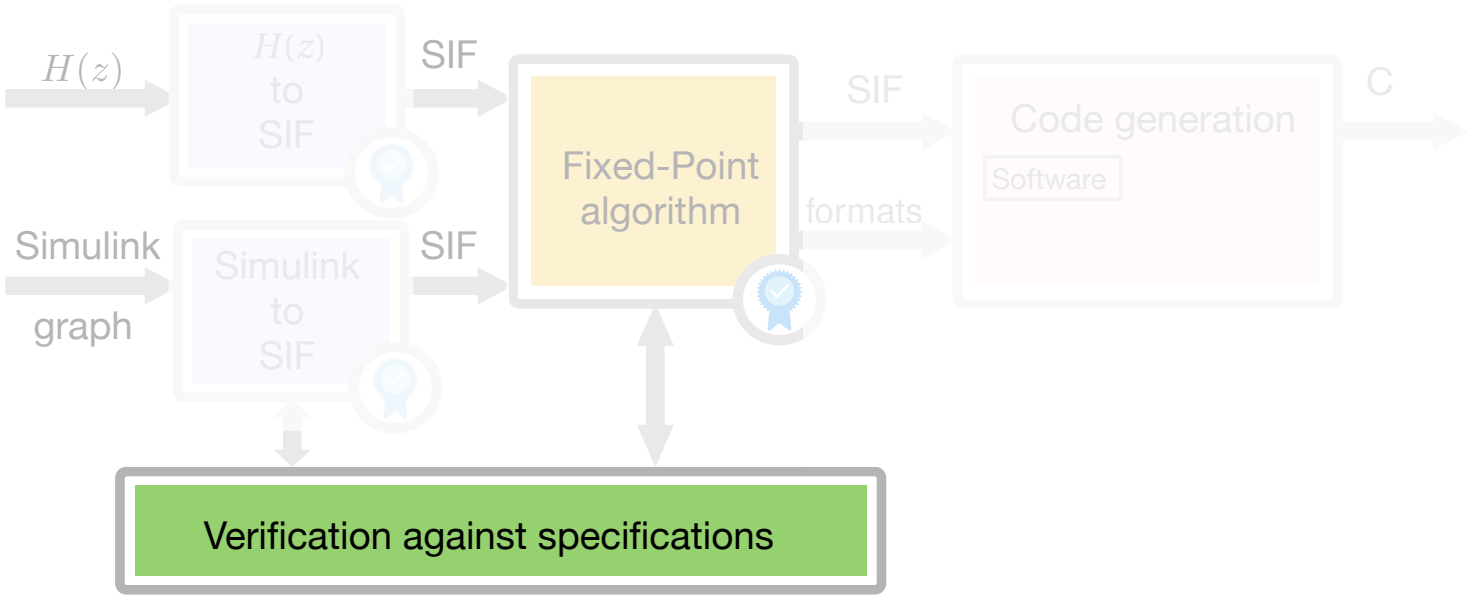
Numerical example

Wordlengths: 16 bits
Input: $\forall k,\ -1 \leq u(k) < 1$
Fixed-Point formats:

	u	x_1	x_2	x_3	y
most significant bit positions	0	5	5	5	1
least significant bit positions	-15	-10	-10	-10	-14

Error bound: $|\Delta(k)| \leq 2^{-9}$
Time: 0.012 s

Back to the generator

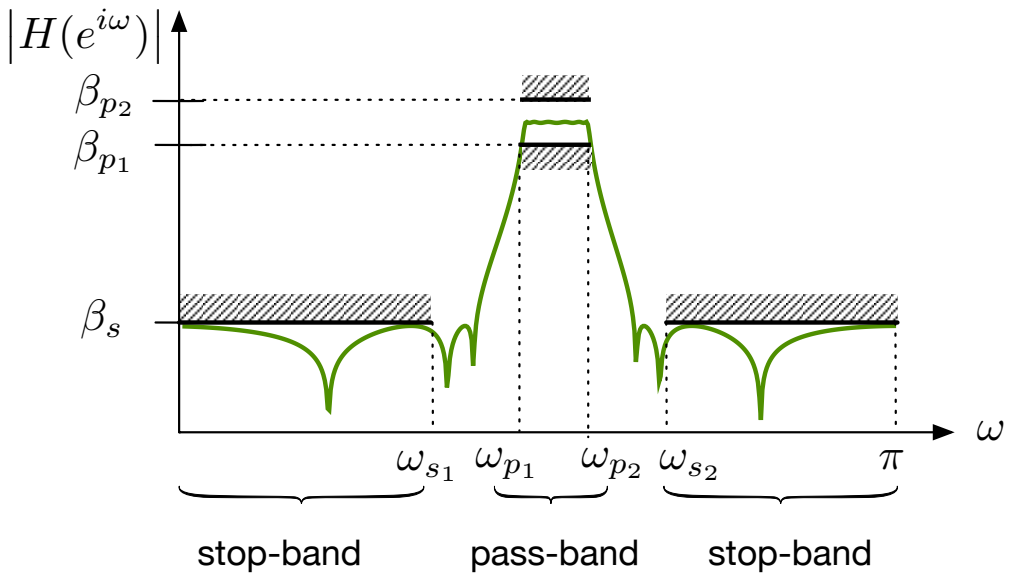


Frequency specifications

Frequency response:

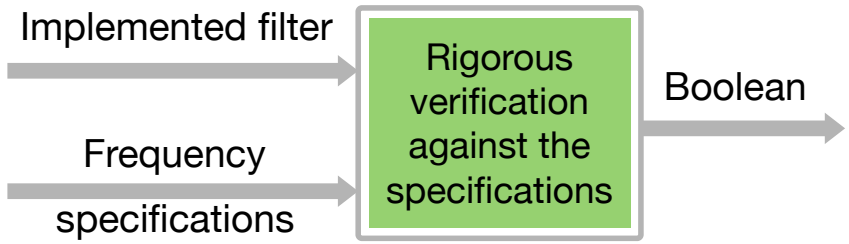
$$H\left(e^{j\omega}\right) = \underbrace{\left|H\left(e^{j\omega}\right)\right|}_{\text{magnitude}} e^{\overbrace{\measuredangle H\left(e^{j\omega}\right)}^{\text{phase}}}$$

Specifications:

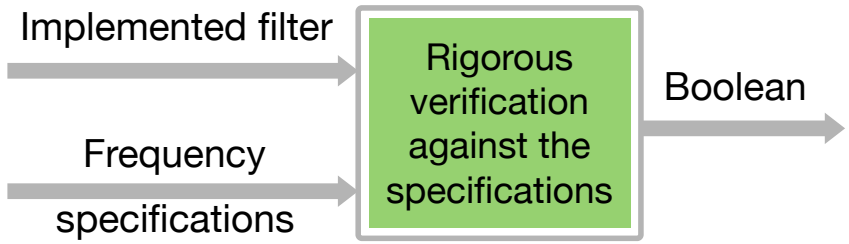


$$\underline{\beta} \leq \left|H(e^{j\omega})\right| \leq \overline{\beta}, \quad \forall \omega \in [\omega_1, \omega_2] \subseteq [0, \pi]$$

Verification of an implementation



Verification of an implementation



Existing approaches:

- by simulations
- approximation of the frequency response

Our reliable approach:

- no simulations, only proofs
- rational and interval arithmetic

Goal:

- Guarantee upon an implementation
- Fast computation of this guarantee

Verification of a transfer function

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

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

Verification of a transfer function

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

$$\underline{\beta}^2 \leq |H(z)|^2 \leq \overline{\beta}^2$$

Verification of a transfer function

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

$$\underline{\beta}^2 \leq |H(z)|^2 \leq \overline{\beta}^2$$

where

$$|H(z)|^2 = \frac{|b(z)|^2}{|a(z)|^2} = \frac{b(z)\overline{b(\overline{z})}}{a(z)\overline{a(\overline{z})}} = \frac{b(z)b(\frac{1}{z})}{a(z)a(\frac{1}{z})} =: \frac{v(z)}{w(z)},$$

$v(z)$ and $w(z)$ are polynomials with real coefficients.

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$

No need to deal with complex numbers

Change of variable: $t = \tan \frac{\omega}{2}$

$$z = e^{j\omega} = \frac{1 - t^2}{1 + t^2} + j \frac{2t}{1 + t^2}$$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{v(\frac{1-t^2}{1+t^2} + j\frac{2t}{1+t^2})}{w(\frac{1-t^2}{1+t^2} + j\frac{2t}{1+t^2})} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan\frac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \underbrace{\frac{r(t) + j\text{ж}(t)}{s(t) + j\text{щ}(t)}}_{\in\mathbb{R} \text{ since } |H(z)|^2} \leq \overline{\beta}^2$$

Polynomials $r, s, \text{ж}, \text{щ} \in \mathbb{R}[t]$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Now we work only with reals

$t = \tan \frac{\omega}{2}$ maps ω on the whole $\mathbb{R}$

Change of variable: $\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(\frac{1-2\xi}{\xi(1-\xi)})}{s(\frac{1-2\xi}{\xi(1-\xi)})} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$
$$\downarrow$$
$$\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$$
$$\xi \in \Xi \subseteq [0, 1]$$

Simplifying the problem

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{p(\xi)}{q(\xi)} \leq \overline{\beta}^2$$

We compute the PGCD(p, q) with a rigorous heuristics by Char *et al.*

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$
$$\downarrow$$
$$\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$$
$$\xi \in \Xi \subseteq [0, 1]$$

Even simpler

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

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

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

$$f(\xi) = q(\xi)^2 \left(\overline{\beta}^2 - \underline{\beta}^2\right)^2 - \left(p(\xi) - \left(\underline{\beta}^2 + \overline{\beta}^2\right) q(\xi)\right)^2$$

Even simpler

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

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

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

$$f(\xi) = q(\xi)^2 \left(\overline{\beta}^2 - \underline{\beta}^2\right)^2 - \left(p(\xi) - \left(\underline{\beta}^2 + \overline{\beta}^2\right) q(\xi)\right)^2$$

Verification that f is positive or zero

Our algorithm is based on:

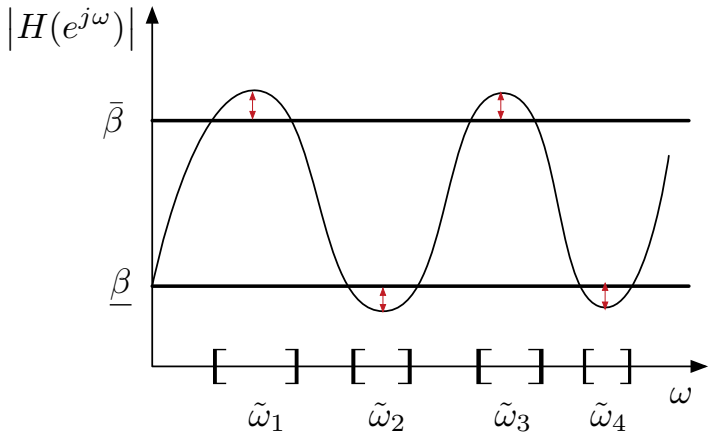
- the Sturm's technique that gives the number of real roots of a polynomial
- a subdivision on sub-intervals
- evaluations in Multiple-Precision Interval Arithmetic

Verification algorithm

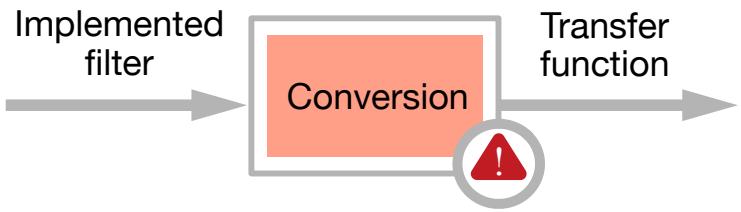
Does the transfer function verify the frequency specifications?

Yes

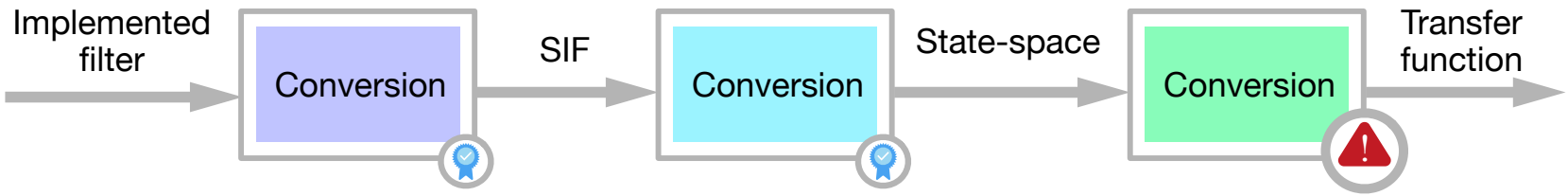
No



Computing the transfer function



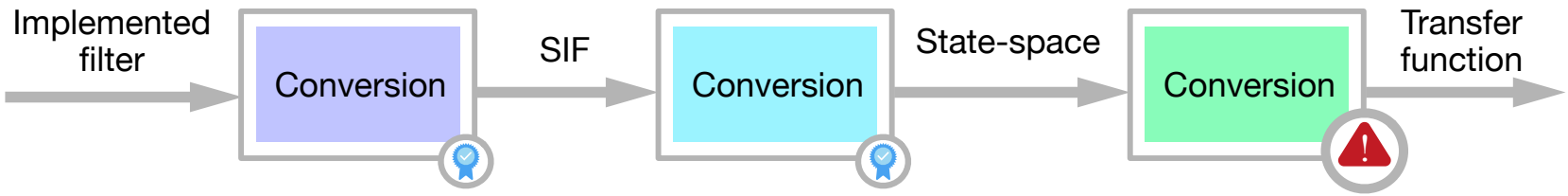
Computing the transfer function



Transfer function of a state-space:

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d$$

Computing the transfer function



Transfer function of a state-space:

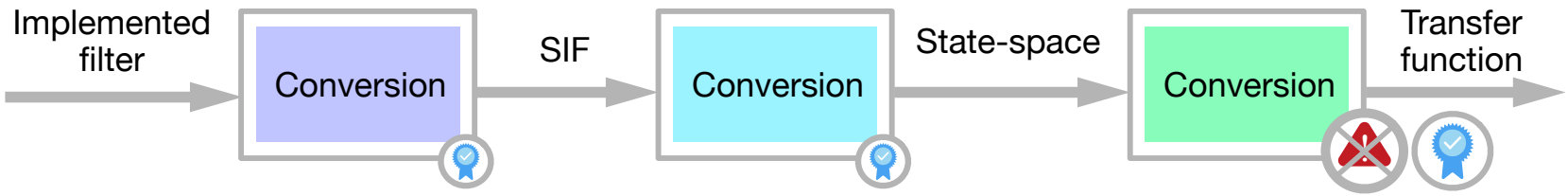
$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1})^{-1}\boldsymbol{b} + d$$

We compute an approximation $\widehat{H}(z)$ of $H(z)$:

$$\widehat{H}(z) = \frac{\sum_i \widehat{b}_i z^{-i}}{\sum_i \widehat{a}_i z^{-i}}$$

☞ The error $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$ may be arbitrarily small.

Computing the transfer function



Transfer function of a state-space:

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1})^{-1}\boldsymbol{b} + d$$

We compute an approximation $\widehat{H}(z)$ of $H(z)$:

$$\widehat{H}(z) = \frac{\sum_i \widehat{b}_i z^{-i}}{\sum_i \widehat{a}_i z^{-i}}$$

☞ The error $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$ may be arbitrarily small.

Problem:

We need a rigorous bound on the error $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$.

How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:

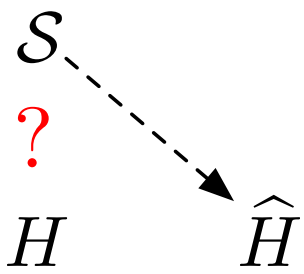
$\mathcal{S}$

?

H

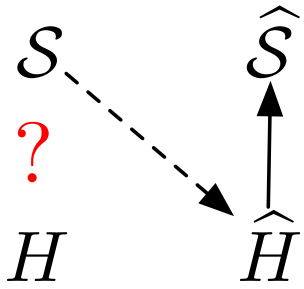
How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:



How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:



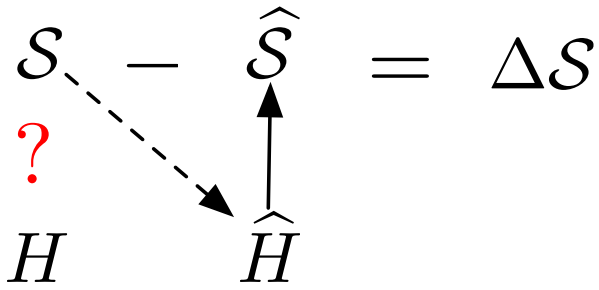
Transformation from $\widehat{H}$ to $\widehat{\mathcal{S}}$ is exact:

$$\widehat{\boldsymbol{A}} = \begin{pmatrix} -\widehat{a}_1 & 1 & & \\ \vdots & & \ddots & \\ \vdots & & & 1 \\ -\widehat{a}_n & 0 & \dots & 0 \end{pmatrix} \quad \widehat{\boldsymbol{b}} = \begin{pmatrix} \widehat{b}_1 - \widehat{a}_1\widehat{b}_0 \\ \vdots \\ \vdots \\ \widehat{b}_n - \widehat{a}_n\widehat{b}_0 \end{pmatrix}$$

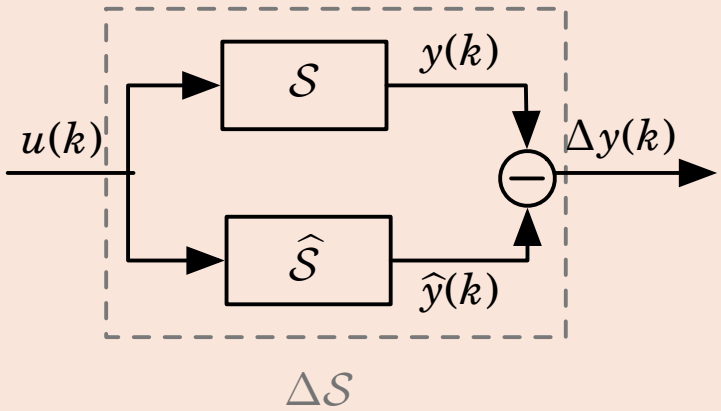
$$\widehat{\boldsymbol{c}} = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \qquad \widehat{d} = \widehat{b}_0$$

How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:

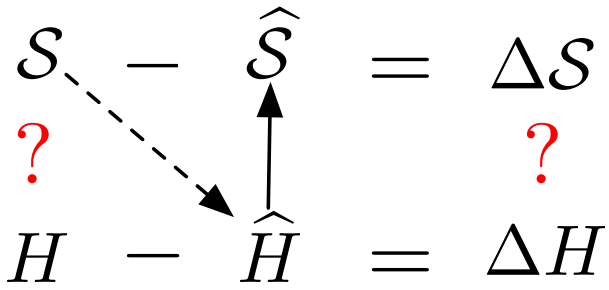


The difference of two filters is defined as:



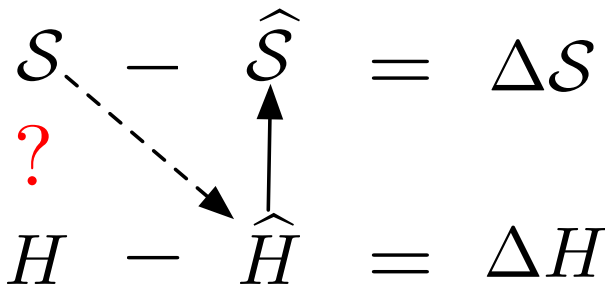
How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:



How to bound the error

Computing the transfer function $H(z)$ of the state-space system $\mathcal{S}$:



Relationship between $\Delta\mathcal{S}$ and ΔH:

$$\left|\left(H-\widehat{H}\right)(e^{j\omega})\right| \leq \langle\langle \Delta\mathcal{S} \rangle\rangle, \quad \forall\omega \in [0, 2\pi]$$

where $\langle\langle \Delta\mathcal{S} \rangle\rangle$ is again the WCPG of the system $\Delta\mathcal{S}$.

Reliable evaluation of the WCPG

Reliable WCPG is required for:

- determination of the range of variables
- analysis of the errors induced by finite-precision computations
- bounding the error of the approximation of the transfer function

Reliable evaluation of the WCPG

Reliable WCPG is required for:

- determination of the range of variables
- analysis of the errors induced by finite-precision computations
- bounding the error of the approximation of the transfer function

Problem in the case of MIMO filters

Compute $\boldsymbol{S}$, approximation of the matrix

$$\langle\langle \mathcal{H} \rangle\rangle = |\boldsymbol{D}| + \sum_{k=0}^{\infty} \left| \boldsymbol{C}\,\boldsymbol{A}^k\,\boldsymbol{B} \right|,$$

such that for an *a priori* given ε

$$|\langle\langle \mathcal{H} \rangle\rangle - \boldsymbol{S}| < \varepsilon$$

☞ We use the *dynamic Multiple-Precision* arithmetic.

Truncation bound

We determine N such that the truncation error verifies

$$\left| \sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right| \quad - \quad \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right| \right| \leq \varepsilon_1$$

Mathematical solution:

$$N \geq \left\lceil \frac{\log_2 \frac{\varepsilon_1}{\|\boldsymbol{M}\|_{min}}}{\log_2 \rho(\boldsymbol{A})} \right\rceil$$

where

$$\boldsymbol{M} = \sum_{l=1}^{n} \frac{|\boldsymbol{R}_l|}{1-|\boldsymbol{\lambda}_l|} \frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})}$$

$$(\boldsymbol{R}_l)_{i,j} = \boldsymbol{C}_{i,l}\boldsymbol{B}_{l,j}$$

Practical solution:

- Eigenvalues computed with LAPACK
- Interval Arithmetic
- Theory of Verified Inclusions (Rump)

Evaluation: powering

$$\sum_{k=0}^{N} \left| C A^k B \right|$$

We propose:

$$A = X E X^{-1}$$

thus, $A^k = X E^k X^{-1}$

where

X are the eigenvectors

E are the eigenvalues

Evaluation: powering

$$\left| \sum_{k=0}^{N} \left| C\,\boldsymbol{A}^k\,B \right| \right. \quad - \quad \left. \sum_{k=0}^{N} \left| C\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\,B \right| \right| \leq \varepsilon_2$$

We propose:

$$\boldsymbol{A} = \boldsymbol{X}\,\boldsymbol{E}\,\boldsymbol{X}^{-1}$$

thus, $\boldsymbol{A}^k = \boldsymbol{X}\,\boldsymbol{E}^k\,\boldsymbol{X}^{-1}$

where

$\boldsymbol{X}$ are the eigenvectors

$\boldsymbol{E}$ are the eigenvalues

In practice:

$$\boldsymbol{V} \approx \boldsymbol{X} \quad \text{et} \quad \boldsymbol{T} = \boldsymbol{V}^{-1}\,\boldsymbol{A}\,\boldsymbol{V} + \Delta$$

- Multiple precision operations with *a priori* given error bound
- Δ is controlled to satisfy the bound ε_2 on the propagated error
- $\|\boldsymbol{T}\|_2 \leq 1$ rigorously verified with the Gershgorin theorem

Evaluation: summation

We proceed by analogy

$$\left|\sum_{k=0}^{N} |\boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\boldsymbol{B}| - \sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{T}^k\,\boldsymbol{B}'|\right| \leq \varepsilon_3$$

$$\left|\sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{T}^k\,\boldsymbol{B}'| - \sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{P}_k\,\boldsymbol{B}'|\right| \leq \varepsilon_4$$

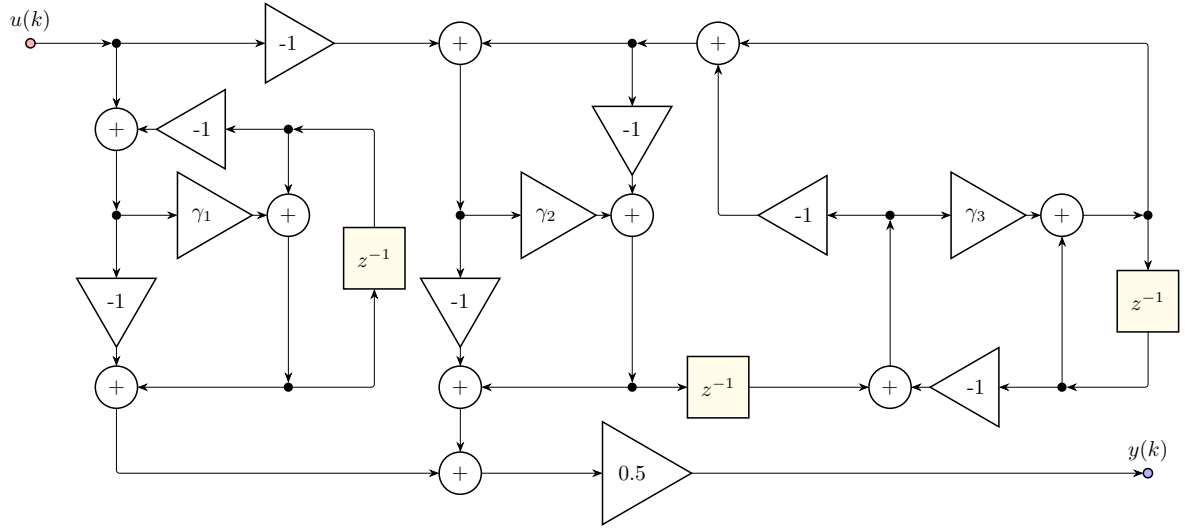
$$\left|\sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{P}_k\,\boldsymbol{B}'| - \sum_{k=0}^{N} |\boldsymbol{L}_k|\right| \leq \varepsilon_5$$

$$\left|\sum_{k=0}^{N} |\boldsymbol{L}_k| - \boldsymbol{S}_N\right| \leq \varepsilon_6$$

☞ **Three basic bricks:** $\boldsymbol{X}\,\boldsymbol{Y} + \boldsymbol{Z}$, $\boldsymbol{X} + |\boldsymbol{Y}|$, $\boldsymbol{X}^{-1}$

Example of frequency domain verification

An implementation of our example filter:



$$\gamma_1 = 89 \cdot 2^{-8}$$
$$\gamma_2 = 43 \cdot 2^{-7}$$
$$\gamma_3 = 11 \cdot 2^{-7}$$

Specifications:

$$\begin{cases} 10^{\frac{1}{20}} \leq \left|H(e^{j\omega})\right| \leq 10^{\frac{3}{20}} & \forall \omega \in [0, \frac{1}{10}\pi] \quad \text{(passe-bande)} \\ \left|H(e^{j\omega})\right| \leq 10^{-\frac{20}{20}} & \forall \omega \in [\frac{3}{10}\pi, \pi] \quad \text{(coupe-bande)} \end{cases}$$

Result: ✓ implemented filter respects the specifications

Verification time: 1.9 s

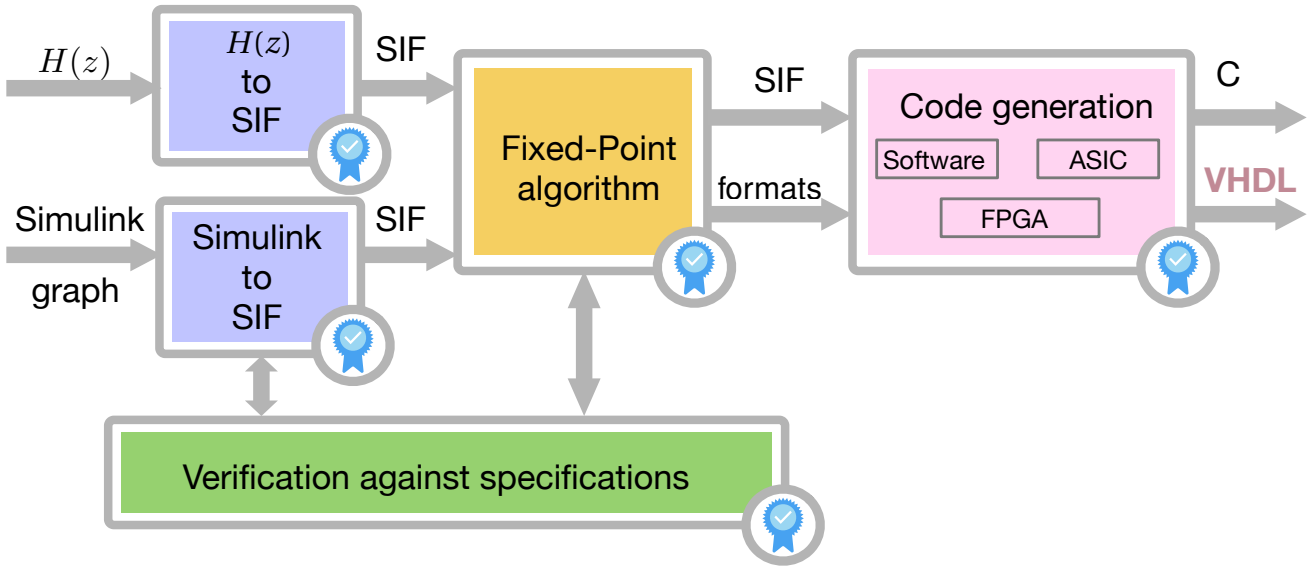
Behind the scenes

During these 1.9 seconds...

- Rational arithmetic
  - Rewriting $H(z)$ as $f(\xi)$
  - Computation of Sturm sequences
- Interval arithmetic
  - Verified Inclusions for the eigenvalues
  - Truncation bound for N
- Dynamic Multiple-Precision arithmetic
  - Computation of Gershgorin circles
  - Evaluation du WCPG
- Floating-Point IEEE 754 arithmetic
  - Computation of eigenvalues with LAPACK

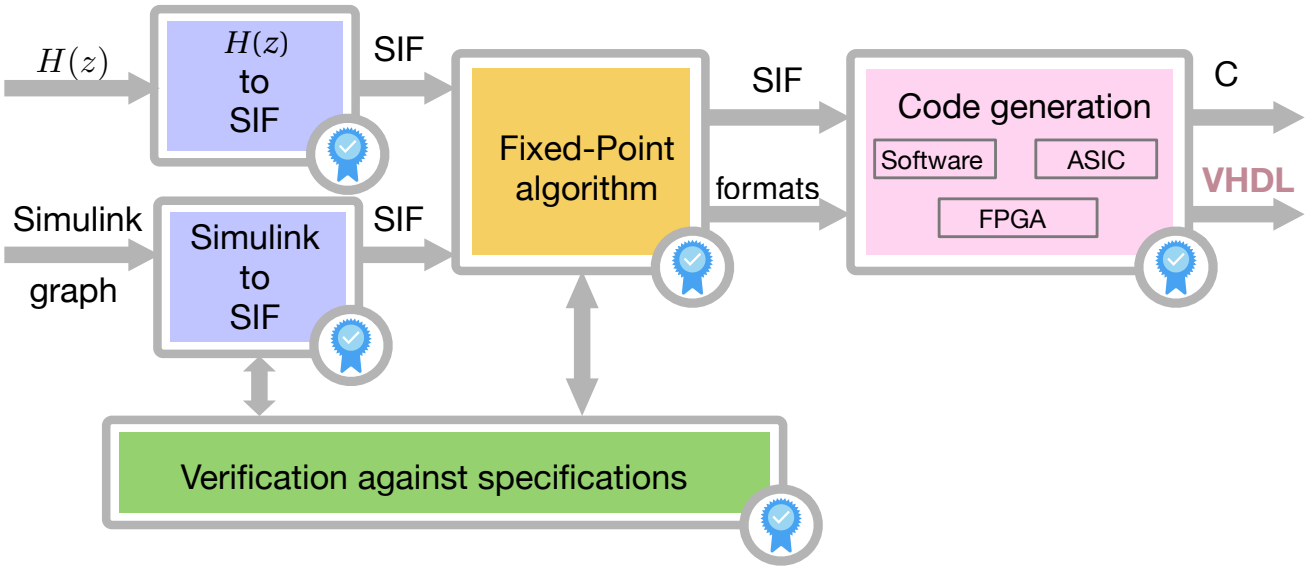
. . . to verify an implementation in Fixed-Point arithmetic!

Conclusion



- We made filter implementation reliable and automatic
  - Rigorous use of various arithmetics
  - Fixed-Point implementations that verify *a priori* error bounds in both frequency and time domains
- We extended the tool-chain of the filter code generator
  - New conversions from various input formats
  - Plugged in with the FloPoCo tool for FPGA implementations

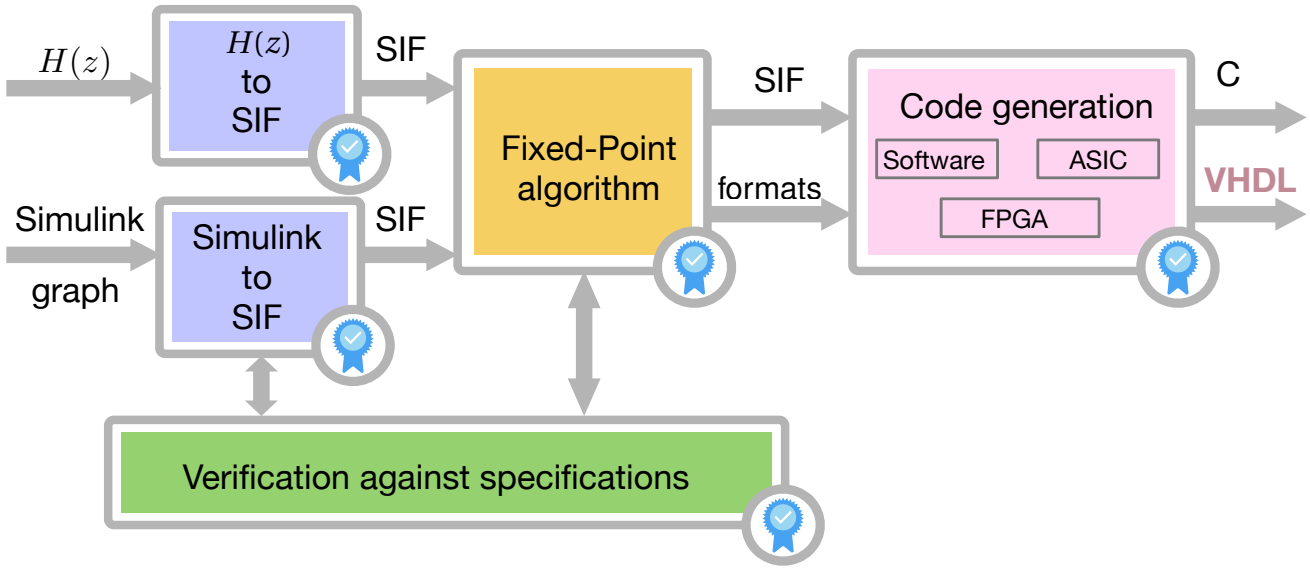
Tool development



- Open-source tool
- Implementation in C/C++, Sollya, Python and Matlab
- $\approx$ 15000 lines of code
- Generation of VHDL with FloPoCo

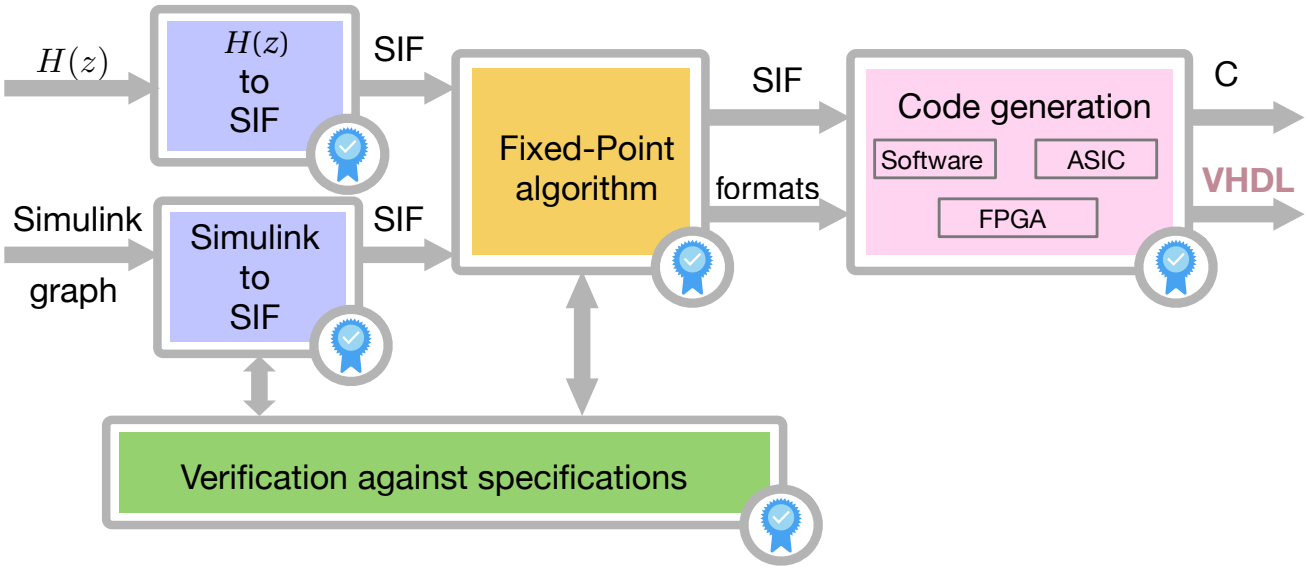
FIXIF TOOLBOX

Perspectives: tomorrow at 9h00...



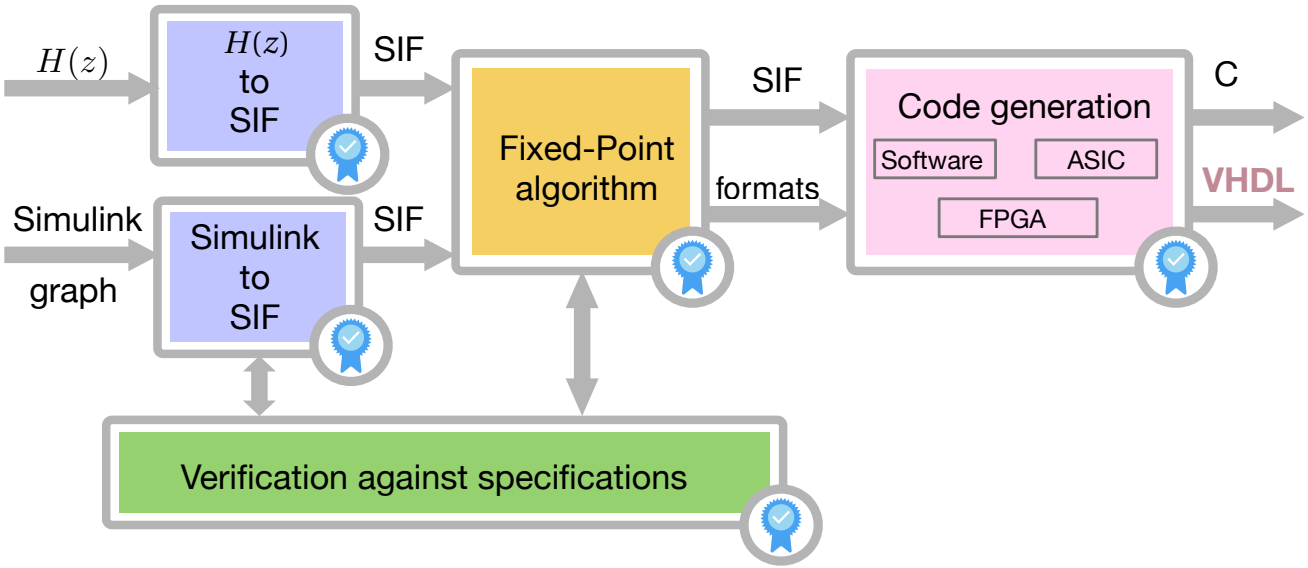
- Win some space
  - Solve the "off-by-one" problem $(0 \leq \widehat{m}_y - m_y \leq 1)$
- Win more space
  - Take into account additional properties of input signals (spectral properties)
- Study various algorithms with respect to different metrics
  - LUTs, surface
  - accuracy

Perspectives: moyen terme



- Reliable implementation of digital filters
  - Integrate methods of the transfer function design
  - Relax the frequency-domain constraints
- Arithmetic tools
  - Computation of eigenvalues in varying precision
  - Solution of Lyapunov equations, etc.
- Optimization
  - Wrap-up the generator with optimization loops

Perspectives: long term



- Computer Arithmetic side
  - Reliable numerical tools for linear algebra
  - Evaluation of rational approximations
- Signal Processing side
  - Design the transfer functions with the best quantized coefficients
  - Consider non-linear filters, Kalman filters, etc.

Merci

Thank you

Спасибо

Дякую

List of publications

- A.V., C. Lauter., T. Hilaire and M. Mezzarobba Rigorous Determination of Recursive Filter Fixed-Point Implementation with Input Signal Frequency Specifications In *ASILOMAR'51*, Nov 2017
- T. Hilaire, A.V. Error analysis methods for the fixed-point implementation of linear systems. In *IEEE SiPS 2017*, Oct 2017
- F. Qureshi, A. V., J. Takala and T. Hilaire. Multiplierless Unified Architecture for Mixed Radix-2/3/4 FFTs. In *EUSIPCO'25*, Aug 2017
- A.V., C. Lauter and T. Hilaire. Reliable verification of digital implemented filters against frequency specifications. In *IEEE ARITH'24*, July 2017
- T. Hilaire, A.V. and M. Ravoson. Reliable fixed-point implementation of linear data-flows. In *IEEE SiPS 2016*, Oct 2016
- A.V., T. Hilaire and C. Lauter. Determining fixed-point formats for a digital filter implementation using the worst-case peak gain measure. In *ASILOMAR'49*, Nov 2015
- A.V. and T. Hilaire. Fixed-point implementation of lattice wave digital filter: Comparison and error analysis. In *EUSIPCO'23*, Sep 2015
- A.V., T. Hilaire, and C. Lauter. Reliable evaluation of the worst-case peak gain matrix in multiple precision. In *IEEE ARITH'22*, Jun 2015
- F. de Dinechin, T. Hilaire, M. Istoan, A.V. LTI Filters Computed Just Right Technical report

SIF

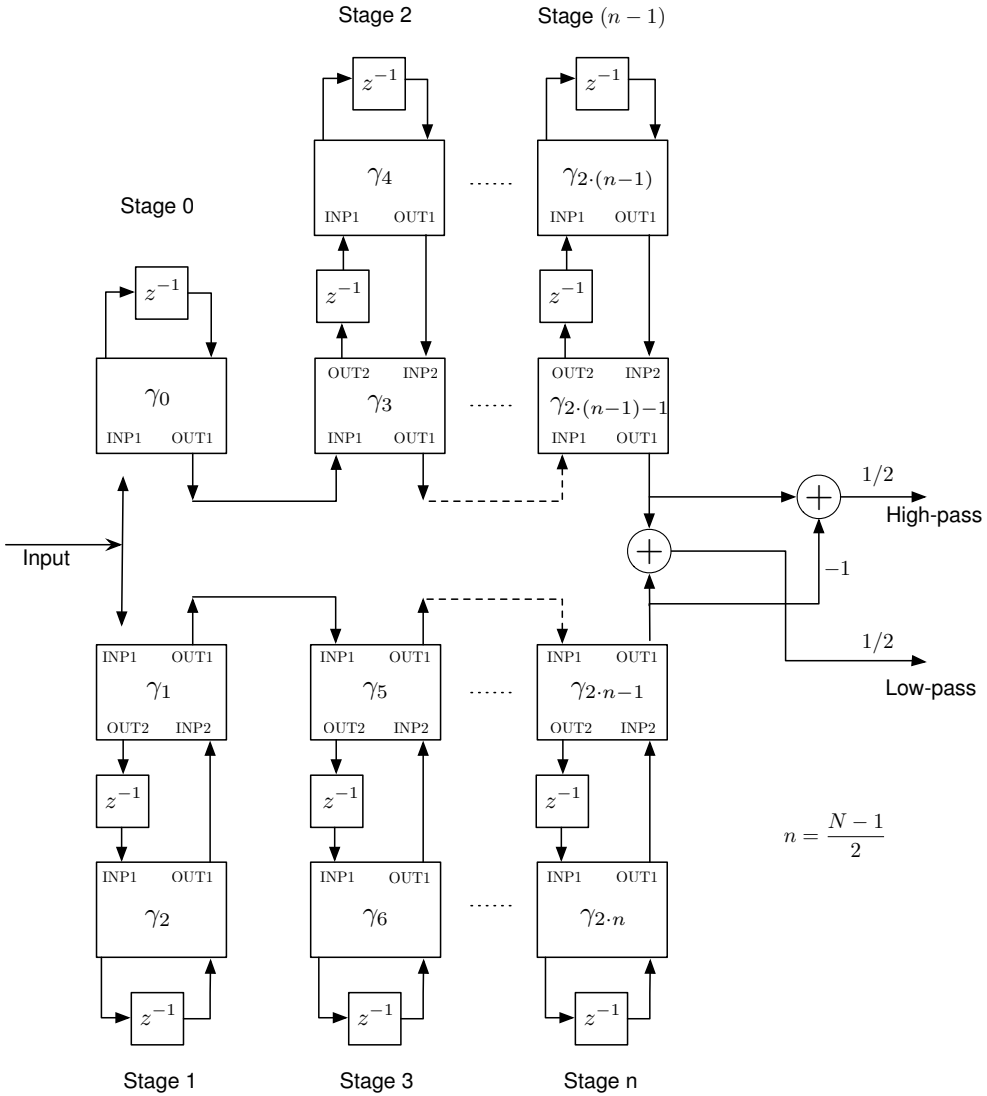
A MIMO LTI system in SIF representation is characterized by

$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_n & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix}$$

By rewriting we obtain

$$\begin{cases} \boldsymbol{J}\boldsymbol{t}(k+1) = \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) = \boldsymbol{K}\boldsymbol{t}(k+1) + \boldsymbol{P}\boldsymbol{x}(k) + \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) = \boldsymbol{L}\boldsymbol{t}(k+1) + \boldsymbol{R}\boldsymbol{x}(k) + \boldsymbol{S}\boldsymbol{u}(k) \end{cases}$$

Lattice Wave Digital Filters



The error filter $\mathcal{H}_\Delta$

Denote

$$\boldsymbol{\varepsilon}(k) = \begin{pmatrix} \boldsymbol{\varepsilon}_x(k) \\ \varepsilon_y(k) \end{pmatrix}$$

$$\boldsymbol{\Delta}_x(k) = \boldsymbol{x}^\Diamond(k) - \boldsymbol{x}(k)$$

$$\boldsymbol{\Delta}_y(k) = \boldsymbol{y}^\Diamond(k) - \boldsymbol{y}(k)$$

Then, the filter $\mathcal{H}_\Delta = \mathcal{H} - \mathcal{H}^\Diamond$ is

$$\mathcal{H}_\Delta \begin{cases} \boldsymbol{\Delta}_x(k+1) &= \boldsymbol{A}\boldsymbol{\Delta}_x(k) &+ \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \boldsymbol{\varepsilon}(k) \\ \boldsymbol{\Delta}_y(k) &= \boldsymbol{C}\boldsymbol{\Delta}_x(k) &+ \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{\varepsilon}(k) \end{cases}$$

Worst-Case Peak Gain

Sensitive real-life filter

Input: A highly sensitive 5^{th} order transfer function from industrial application[1].

Problem: given inputs from interval $[-1.125; 1.125]$, determine the interval for output variables.

Result:

- Naive WCPG, i.e. summing 1000 terms in double precision: $\bar{y} = 1.6238497 \times 1.125 = 0.8693\ldots$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 1.9997191 \times 1.125 = 1.1697\ldots$

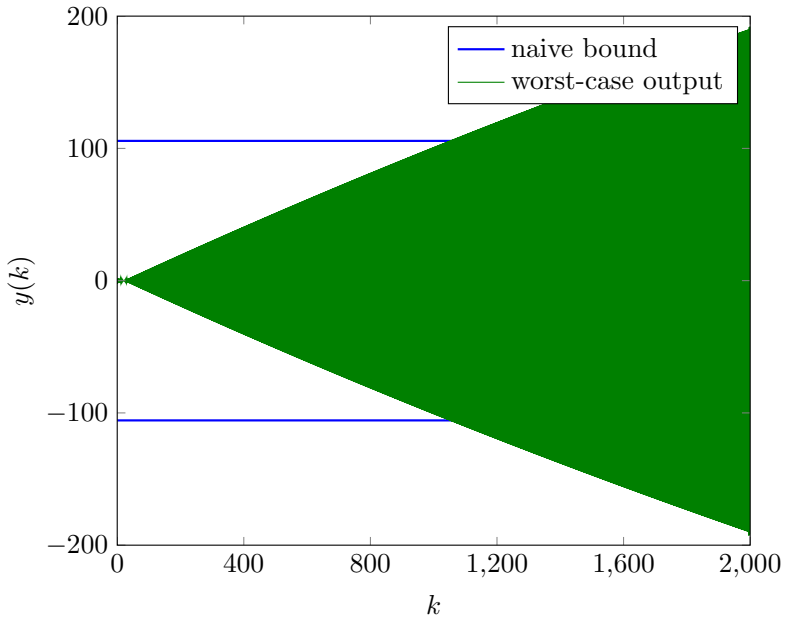
[1] Obtained from Xilinx

Worst-Case Peak Gain

Random filter

Input: Consider a stable 5^{th} order random SISO filter
Problem: given inputs from interval $[-1; 1)$, determine the interval for output variables.
Result:

- Naive WCPG (1000 terms in double precision) $\bar{y} = 105.66...$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 772.48...$

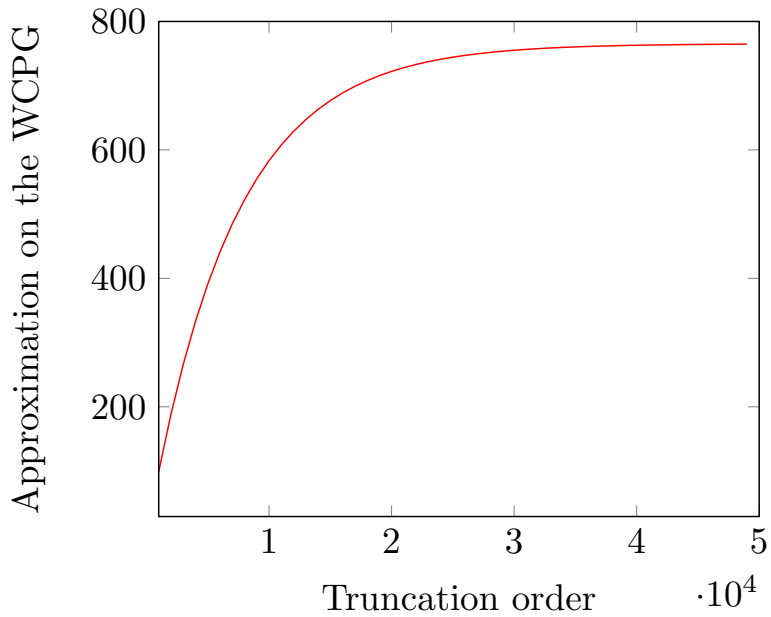


Worst-Case Peak Gain

Random filter

Input: Consider a stable 5^{th} order random SISO filter
Problem: given inputs from interval $[-1; 1)$, determine the interval for output variables.
Result:

- Naive WCPG (1000 terms in double precision) $\bar{y} = 105.66...$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 772.48...$



Worst-Case Peak Gain

Examples

Example 1: comes from Control Theory, describes a controller of vehicle longitudinal oscillation

Example 2: 12th-order Butterworth filter

	Example 1			Example 2		
sizes n, p and q	$n=10,\quad p=11,\quad q=1$			$n=12,\quad p=1,\quad q=25$		
$1-\rho(\boldsymbol{A})$	1.39×10^{-2}			8.65×10^{-3}		
$\max(\boldsymbol{S}_N)$	3.88×10^{1}			5.50×10^{9}		
$\min(\boldsymbol{S}_N)$	1.29×10^{0}			1.0×10^{0}		
ε	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}
N	220	2153	29182	308	4141	47811
Inversion iterations	0	2	4	2	3	5
overall max precision (bits)	212	293	1401	254	355	1459
$\boldsymbol{V}^{-1}$ max precision (bits)	106	173	727	148	204	756
$\boldsymbol{P}_N$ max precision (bits)	64	84	639	64	86	640
$\boldsymbol{S}_N$ max precision (bits)	64	79	630	64	107	658
Overall execution time (sec)	0.11	1.53	60.06	0.85	11.54	473.20

Transfer Function of a State-Space

Transfer function of a single-input single-output state-space $\mathcal{S}$:

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d$$

Using the eigendecomposition $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1}$:

$$H(z) = \frac{P(z)}{Q(z)} + d$$

$$P(z) = \sum_{i=1}^{n} (\boldsymbol{c}\boldsymbol{X})_i (\boldsymbol{X}^{-1}\boldsymbol{b})_i \prod_{j \neq i} (z - \lambda_j)$$

$$Q(z) = \prod_{j=1}^{n} (z - \lambda_j)$$

We compute an approximation $\widehat{H}(z)$ in Multiple Precision arithmetic with `mpmath`.

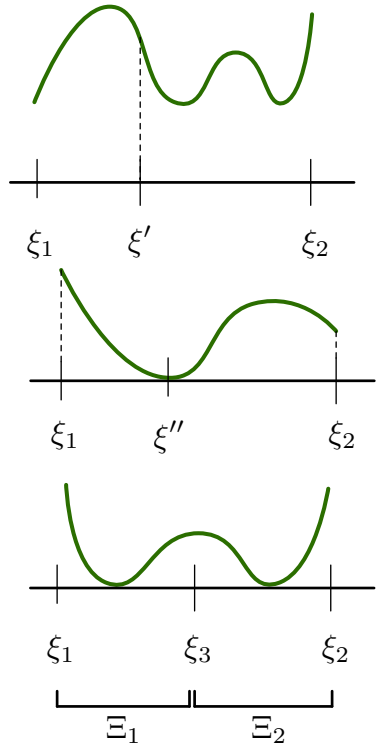
Verification of non-negativity

To verify $f(\xi) \geq 0$, $\forall \xi \in \Xi = [\xi_1, \xi_2] \subseteq [0, 1]$ we check if

(i) $f(\xi)$ has no zeros
$f(\xi') > 0$ for some $\xi' \in [\xi_1, \xi_2]$

(ii) $f(\xi)$ has one zero
$f(\xi_1) > 0$ and $f(\xi_2) > 0$

(iii) interval Ξ can be split into subintervals s.t. (i) or (ii) are satisfied for every subinterval



We use Sollya tool for the implementation

- Number of zeros: Sturm's theorem
- Evaluations: interval multiple precision arithmetic

Verification of specifications

Sturm's technique

Sturm's sequence is a sequence of polynomials $p_0(x), \ldots, p_m(x)$:

$$
\begin{aligned}
p_0(x) &= p(x) \\
p_1(x) &= p'(x) \\
p_2(x) &= -rem(p_0, p_1) = p_1(x)q_0(x) - p_0(x), \\
p_3(x) &= -rem(p_1, p_2) = p_2(x)q_1(x) - p_1(x), \\
&\ldots \\
0 &= -rem(p_{m-1}, p_m)
\end{aligned}
$$

Verification of specifications

Numerical examples

Input: four realizations of the same filter

Problem: verify realizations after coefficient quantization to 32/16/8 bits

Results:

	wordlength	32	16	8
DFIIt	margin	✓	unstable	unstable
	time	12.49s	-	-
ρ DFIIt	margin	✓	✓	4.68e-3 dB
	time	13.12s	4.19s	104.01s
State-Space Balanced	margin	6.16e-10 dB	✓	6.71e-1 dB
	time	12.27s	18.18s	92.05s
Lattice Wave	margin	3.80e-10 dB	✓	1.73e-2 dB
	time	920.88s	4.58s	200.83s

Verification of specifications

Numerical examples

Input: four simple frequency specifications

Problem: Verify and compare transfer function design methods.

Results: comparison of SciPy in Python and Matlab

		Butterworth	Chebyshev	Elliptic
		margin (dB)	margin (dB)	margin (dB)
lowpass	Matlab	1.29e-17	7.93e-17	✓
	SciPy	2.14e-15	4.48e-2	4.48e-2
highpass	Matlab	2.77e-16	6.94e-17	4.48e-2
	SciPy	3.02e-15	2.29e-16	4.48e-2
bandpass	Matlab	3.04e-17	✓	✓
	SciPy	✓	4.48e-2	4.48e-2
bandstop	Matlab	4.59e-16	3.09e-15	✓
	SciPy	✓	6.36e-15	7.02e-6

Verification of specifications

Numerical examples

Filter implementation: 14^{th} order bandpass filter
Specifications:

$$\begin{cases} 0\text{dB} \leq & |H(e^{i\omega})| \leq & -80\text{dB} & \forall\omega \in [0, 17\text{kHz}] & \text{(stopband)} \\ & |H(e^{i\omega})| \leq & 1 - 10^{-4}\text{dB} & \forall\omega \in [21\text{kHz}, 25\text{kHz}] & \text{(passband)} \\ & |H(e^{i\omega})| \leq & -80\text{dB} & \forall\omega \in [27\text{kHz}, 30\text{kHz}] & \text{(stopband)} \end{cases}$$

Verification result: implemented filter *does not* pass the verification against frequency constraints
Verification time: 53 s

Verification of specifications

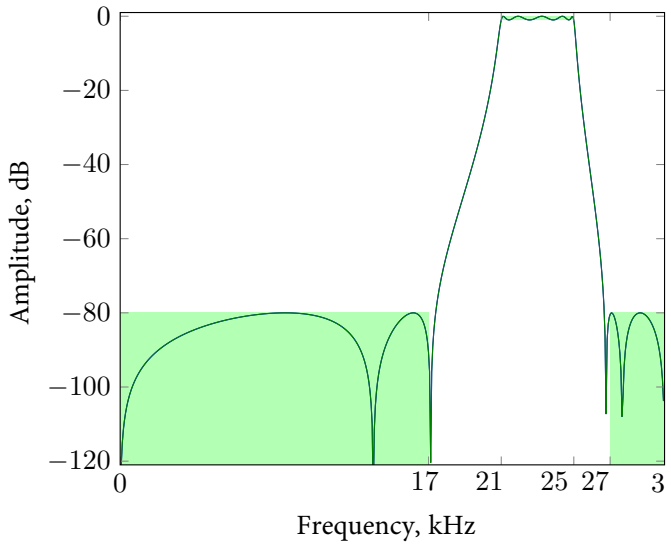
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Verification of specifications

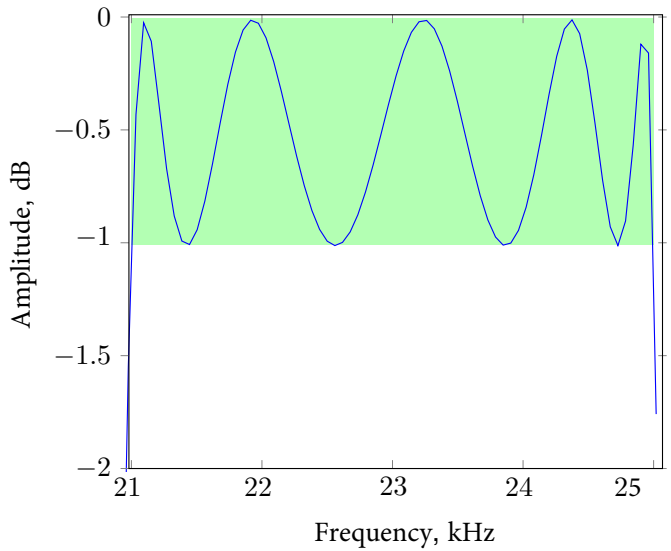
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Verification of specifications

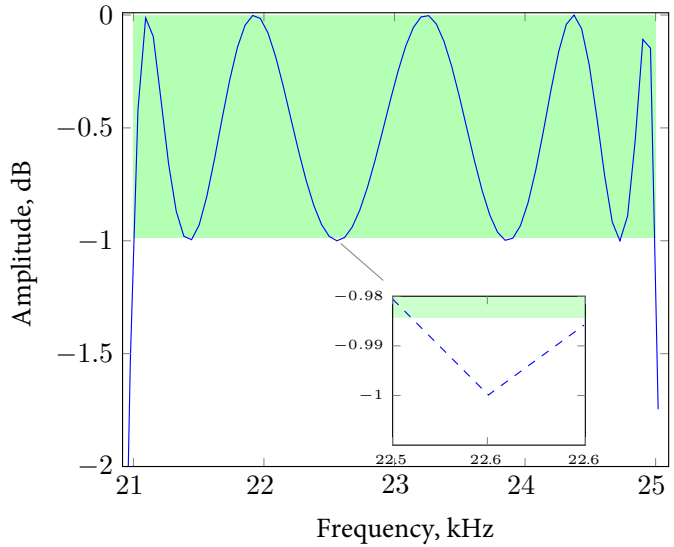
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Binding with FloPoCo

FloPoCo:

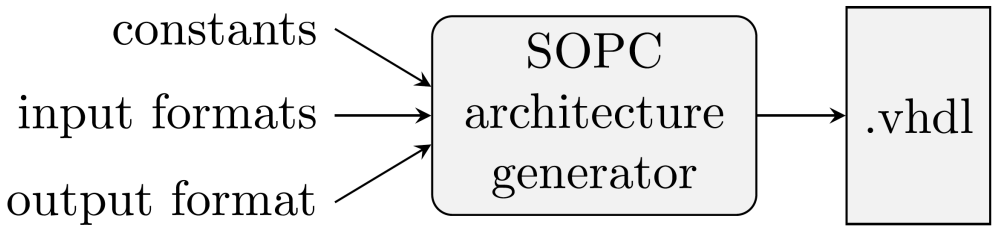


Figure: Interface to a Sum-Of-Product-by-Constant generator

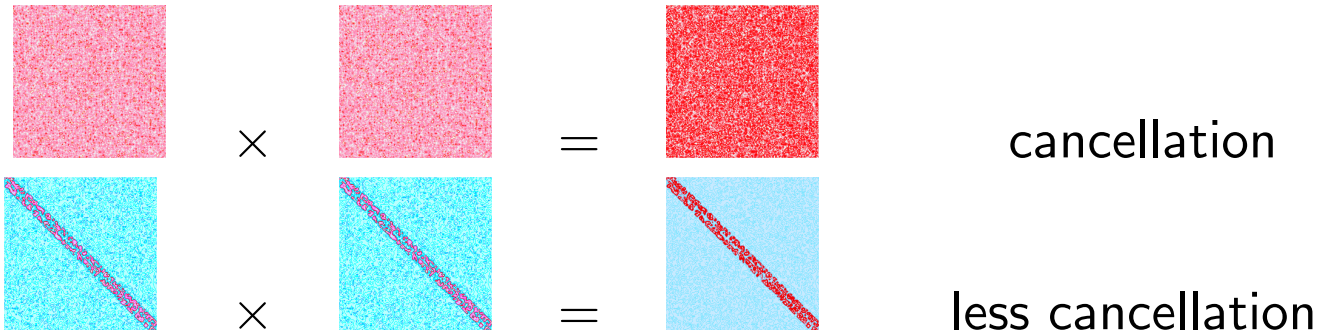
Our tool: we deduce a lower bound on the error of computation of each Sum-of-Product s.t. the error-bound *on the filter's output* is respected.

- Push-button
- Can implement any structure
- No need to quantize coefficients
- Reliable

Worst-Case Peak Gain

Powering matrix A

$$\sum_{k=0}^{N} \left| C A^k B \right|$$



$$A = XEX^{-1} \qquad V \approx X \text{ and } T \approx E$$

$$T \approx V^{-1} \times A \times V$$

$$A^k \approx V \times T^k \times V^{-1}$$

Worst-Case Peak Gain

Step 3

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\,\boldsymbol{B} \right| \quad - \quad \sum_{k=0}^{N} \left| \boldsymbol{C'}\,\boldsymbol{T}^k\,\boldsymbol{B'} \right| \right| \leq \varepsilon_3$$

Step 3 Compute the products $\boldsymbol{C}\,\boldsymbol{V}$ and $\boldsymbol{V}^{-1}\boldsymbol{B}$ such that the propagated error of matrix multiplications is bounded by ε_3.

$$\sum_{k=0}^{\infty} \left| \boldsymbol{C}\,\boldsymbol{A}^k\,\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\,\boldsymbol{A}^k\,\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\,\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C'}\,\boldsymbol{T}^k\,\boldsymbol{B'} \right|$$

Worst-Case Peak Gain

Step 4

$$\left| \sum_{k=0}^{N} |C' T^k B'| \quad - \quad \sum_{k=0}^{N} |C' P_k B'| \right| \leq \varepsilon_4$$

$$P_0 := I$$
$$P_k := T \otimes P_{k-1}$$

$$\sum_{k=0}^{\infty} |C A^k B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C A^k B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C V T^k V^{-1} B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C' T^k B'|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C' P_k B'|$$

Step 4 Compute the powers P_k of matrix T such that the propagated error of matrix multiplications is bounded by ε_4.

Worst-Case Peak Gain

Step 5

$$\left| \sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}| \quad - \quad \sum_{k=0}^{N} |\boldsymbol{L}_k| \right| \leq \varepsilon_5$$

$$\boldsymbol{L}_k := \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$$

Step 5 Compute on each step the matrix product $\boldsymbol{C'T}^k\boldsymbol{B'}$ such the overall error of these multiplications on each step is bounded by ε_5.

$$\sum_{k=0}^{\infty} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CVT}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{C'T}^k\boldsymbol{B'}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}|$$
$$\downarrow$$

Worst-Case Peak Gain

Step 5

$$\left| \sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}| \quad - \quad \sum_{k=0}^{N} |\boldsymbol{L}_k| \right| \leq \varepsilon_5$$

$$\boldsymbol{L}_k := \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$$

Step 5 Compute on each step the matrix product $\boldsymbol{C'T}^k\boldsymbol{B'}$ such the overall error of these multiplications on each step is bounded by ε_5.

$$\sum_{k=0}^{\infty} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CVT}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{C'T}^k\boldsymbol{B'}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{L}_k|$$

Worst-Case Peak Gain

Step 6

$$\left| \sum_{k=0}^{N} |\boldsymbol{L}_k| \quad - \quad \boldsymbol{S}_N \right| \leq \varepsilon_6$$

$$\boldsymbol{S}_k := \boldsymbol{S}_{k-1} \oplus |\boldsymbol{L}_k|$$

Step 6 Compute the absolute value of matrix and accumulate it in the result such that the error is bounded by ε_6.

$$\sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{B}' \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C}'\boldsymbol{P}_k\boldsymbol{B}'|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{L}_k|$$
$$\downarrow$$
$$\boldsymbol{S}_N$$

Worst-Case Peak Gain

Algorithm

Taking $\varepsilon_i = \frac{1}{6}\varepsilon$ we obtain that $\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_6 \leq \varepsilon$ hence the overall error bound is satisfied.

A floating-point evaluation of the WCPG:

Step 1: Compute N

Step 2: Compute $\boldsymbol{V}$

$\boldsymbol{T} \leftarrow \mathsf{inv}(\boldsymbol{V}) \otimes (\boldsymbol{A} \otimes \boldsymbol{V})$

Step 3: $\boldsymbol{B'} \leftarrow \mathsf{inv}(\boldsymbol{V}) \otimes \boldsymbol{B}$

$\boldsymbol{C'} \leftarrow \boldsymbol{C} \otimes \boldsymbol{V}$

$\boldsymbol{S}_{-1} \leftarrow |\boldsymbol{D}|$, $\boldsymbol{P}_{-1} \leftarrow \boldsymbol{I}_n$

for k **from** 0 **to** N **do:**

Step 4: $\quad \boldsymbol{P}_k \leftarrow \boldsymbol{T} \otimes \boldsymbol{P}_{k-1}$

Step 5: $\quad \boldsymbol{L}_k \leftarrow \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$

Step 6: $\quad \boldsymbol{S}_k \leftarrow \boldsymbol{S}_{k-1} \oplus \mathsf{abs}(\boldsymbol{L}_k)$

end for

Worst-Case Peak Gain

Basic bricks

- `multiplyAndAdd`$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \delta)$: for $\boldsymbol{A} \in \mathbb{C}^{p\times n}$, $\boldsymbol{B} \in \mathbb{C}^{n\times q}$, $\boldsymbol{C} \in \mathbb{C}^{p\times q}$, computes a matrix $\boldsymbol{D} \in \mathbb{C}^{p\times q}$ such that

$$\boldsymbol{D} = \boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} + \boldsymbol{\Delta},$$

  where the error-matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

The algorithm performs an error-free scalar multiplication and uses a modified software-implemented Kulisch-like accumulator.

Worst-Case Peak Gain

Basic bricks

- $\texttt{sumAbs}(\boldsymbol{A}, \boldsymbol{B}, \delta)$: for $\boldsymbol{A} \in \mathbb{R}^{p\times n}$, $\boldsymbol{B} \in \mathbb{C}^{p\times n}$, computes a matrix $\boldsymbol{C} \in \mathbb{R}^{p\times n}$ such that

$$\boldsymbol{C} = \boldsymbol{A} + |\boldsymbol{B}| + \boldsymbol{\Delta},$$

where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

Worst-Case Peak Gain

Basic bricks

- $\texttt{inv}(V, \delta)$: for a complex square matrix $\boldsymbol{V} \in \mathbb{C}^{n \times n}$, computes a matrix $\boldsymbol{U} \in \mathbb{C}^{n \times n}$ such that

$$\boldsymbol{U} = \boldsymbol{V}^{-1} + \boldsymbol{\Delta},$$

  where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

The algorithm is based on Newton-Raphson matrix iteration, requires a seed matrix in argument and works on certain conditions, easily verified in our case.

Worst-Case Peak Gain

Basic bricks

- `frobeniusNormUpperBound`$(\boldsymbol{A}, \delta)$: for $\boldsymbol{A} \in \mathbb{C}^{p \times n}$ computes f an upper bound on the Frobenius norm of $\boldsymbol{A}$ such that

$$f = \|\boldsymbol{A}\|_F + \gamma$$

where $0 \leq \gamma < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

Interval Worst-Case Peak Gain

Given a state-space system $\mathcal{H} = ([\boldsymbol{A}], [\boldsymbol{B}], [\boldsymbol{C}], [\boldsymbol{D}])$, compute an approximation $\boldsymbol{S}$ on the WCPG

$$\langle\langle \mathcal{H} \rangle\rangle = |[\boldsymbol{D}]| + \sum_{k=0}^{\infty} \left| [\boldsymbol{C}][\boldsymbol{A}]^k [\boldsymbol{B}] \right|$$

such that two properties are ensured:

- bound property: $\langle\langle \mathcal{H} \rangle\rangle \leq \boldsymbol{S}$ element-by-element;
- if coefficients' radii $\to 0$ and precision $\to \infty$ then the exact $\langle\langle \mathcal{H} \rangle\rangle$ is contained in an ε neighborhood of the approximation $\boldsymbol{S}$ for an a priori given small $\varepsilon > 0$.

Interval Worst-Case Peak Gain

Computing the eigensystem of interval matrix

Eigenvalues of interval matrix

Compute enclosures $\boldsymbol{\lambda}^{\mathcal{I}}$ such that $\forall \boldsymbol{A} \in \boldsymbol{A}^{\mathcal{I}}, \boldsymbol{\lambda}(\boldsymbol{A}) \in \boldsymbol{\lambda}^{\mathcal{I}}$

Approach

Following the works of Xu and Rachid (1996) and Rohn(1998), use the Generalized Gershgorin's Circles theorem.

Eigenvectors of interval matrix

Given the enclosures on eigenvalues $\boldsymbol{\lambda}^{\mathcal{I}}$, compute enclosures $\boldsymbol{V}^{\mathcal{I}}$ such that $\forall \boldsymbol{\lambda} \in \boldsymbol{\lambda}^{\mathcal{I}}, \forall \boldsymbol{A} \in \boldsymbol{A}^{\mathcal{I}}$ if $\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{A}\,\boldsymbol{V}$, then $\boldsymbol{V} \in \boldsymbol{V}^{\mathcal{I}}$.

Approach

Use Rump's theory of Verified Inclusions.