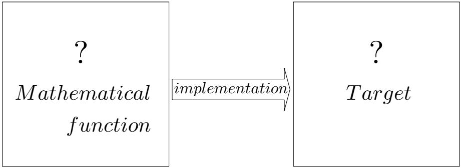
Fixed-Point implementation of Lattice Wave Digital Filter: comparison and error analysis

Anastasia Volkova, Thibault Hilaire

Sorbonne Universités, University Pierre and Marie Curie, LIP6, Paris, France

EUSIPCO 15
September 2, 2015

Motivation



Need to deal with

- Discretize functions and coefficients
  - parametric errors
  - computational errors
- Implementation under constraints
  - software implementation
  - hardware implementation

Motivation

Different filter structures:

- Direct Form I, Direct Form II
- State-space
- Wave, Lattice Wave, ...
- ρ-operator: ρDFIIt, ρModal, ρState-space...
- LGS, LCW, etc.

Problem:

They are equivalent in *infinite* precision but no more in *finite* precision. The finite precision degradation depends on the realization.

Motivation

Given transfer function and a target, we want:

- Represent various realizations
- Evaluate finite precision degradation
- Find an optimal realization
  Tradeoff:
  - Error
  - Quality
  - Power consumption
  - Speed
  - etc.

Motivation

Given transfer function and a target, we want:

- Represent various realizations (in an easy way)
- Evaluate finite precision degradation
- Find an optimal realization
  Tradeoff:
  - Error
  - Quality
  - Power consumption
  - Speed
  - etc.

Motivation

Given transfer function and a target, we want:

- Represent various realizations (in an easy way)
- Evaluate finite precision degradation (a priori/a posteriori)
- Find an optimal realization
  Tradeoff:
  - Error
  - Quality
  - Power consumption
  - Speed
  - etc.

Motivation

Given transfer function and a target, we want:

- Represent various realizations (in an easy way)
- Evaluate finite precision degradation (a priori/a posteriori)
- Find an optimal realization (need to compare realizations)
  Tradeoff:
  - Error
  - Quality
  - Power consumption
  - Speed
  - etc.

Motivation

Given transfer function and a target, we want:

- Represent various realizations (in an easy way)
- Evaluate finite precision degradation (a priori/a posteriori)
- Find an optimal realization (need to compare realizations)
  Tradeoff:
  - Error
  - Quality
  - Power consumption
  - Speed
  - etc.

Specialized Implicit Framework (SIF)

Outline

SIF

SIF is:

- Macroscopic description
- Based on state-space
- Explicit all the computations and their order
- Any DFG can be transformed to this form
- Analytical derivation of measures

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

Denote $\mathbf{Z}$ the matrix containing all the coefficients

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix}$$

SIF: measures

Measures

- *a priori* measures
  - transfer function sensitivity $\left(\text{based on } \frac{\partial H}{\partial \boldsymbol{Z}}\right)$
    $\longrightarrow$ stochastic measure, takes into account coefficient wordlengths
  - poles or zeros sensitivity $\left(\text{e.g based on } \frac{\partial|\lambda_i|}{\partial \boldsymbol{Z}} \text{ for a pole } \lambda_i\right)$
    $\longrightarrow$ stochastic measure, takes into account coefficient wordlengths
  - RNG, ...
- *a posteriori* measures
  - Signal to Quantization Noise Ratio
  - output error

SIF: measures

Measures

- *a priori* measures
  - transfer function sensitivity $\left(\text{based on } \frac{\partial H}{\partial \boldsymbol{Z}}\right)$
    $\longrightarrow$ stochastic measure, takes into account coefficient wordlengths
  - poles or zeros sensitivity $\left(\text{e.g based on } \frac{\partial |\lambda_i|}{\partial \boldsymbol{Z}} \text{ for a pole } \lambda_i\right)$
    $\longrightarrow$ stochastic measure, takes into account coefficient wordlengths
  - RNG, ...
- *a posteriori* measures
  - Signal to Quantization Noise Ratio
  - **output error**

SIF: the rigorous filter error bound

WCPG theorem

Let $\mathcal{H} = \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}\}$ be a BIBO stable MIMO state-space.
If $\forall k \ \boldsymbol{u}(k) \leqslant \bar{\boldsymbol{u}}$ component-wisely, then component-wisely

$$\forall k \quad \boldsymbol{y}(k) \leqslant \langle\langle H \rangle\rangle \, \bar{\boldsymbol{u}},$$

where $\langle\langle H \rangle\rangle$ is the Worst-Case Peak Gain matrix of the system and can be computed as

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

Note: we can compute $\langle\langle H \rangle\rangle$ in arbitrary precision.

SIF: the rigorous filter error bound

Exact filter:

$$\mathcal{H}\left\{\begin{aligned} \boldsymbol{J}\boldsymbol{t}(k+1) &= \boldsymbol{M}\boldsymbol{x}(k)+\boldsymbol{N}u(k) \\ \boldsymbol{x}(k+1) &= \boldsymbol{K}\boldsymbol{t}(k+1)+\boldsymbol{P}\boldsymbol{x}(k)+\boldsymbol{Q}u(k) \\ \boldsymbol{y}(k) &= \boldsymbol{L}\boldsymbol{t}(k+1)+\boldsymbol{R}\boldsymbol{x}(k)+\boldsymbol{S}u(k) \end{aligned}\right.$$

SIF: the rigorous filter error bound

Implemented filter:

$$\mathcal{H}^{*}\left\{\begin{aligned} \boldsymbol{J}\boldsymbol{t}^{*}(k+1) &= \boldsymbol{M}\boldsymbol{x}^{*}(k)+\boldsymbol{N}u(k)+\boldsymbol{\varepsilon}_t(k)\\ \boldsymbol{x}^{*}(k+1) &= \boldsymbol{K}\boldsymbol{t}^{*}(k+1)+\boldsymbol{P}\boldsymbol{x}^{*}(k)+\boldsymbol{Q}u(k)+\boldsymbol{\varepsilon}_x(k)\\ \boldsymbol{y}^{*}(k) &= \boldsymbol{L}\boldsymbol{t}^{*}(k+1)+\boldsymbol{R}\boldsymbol{x}^{*}(k)+\boldsymbol{S}u(k)+\boldsymbol{\varepsilon}_y(k)\end{aligned}\right.$$

where $\boldsymbol{\varepsilon}_t(k)$, $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ are the computational errors.

SIF: the rigorous filter error bound

Implemented filter:

$$\mathcal{H}^* \begin{cases} \boldsymbol{J}\boldsymbol{t}^*(k+1) = \boldsymbol{M}\boldsymbol{x}^*(k) + \boldsymbol{N}u(k) + \boldsymbol{\varepsilon}_t(k) \\ \boldsymbol{x}^*(k+1) = \boldsymbol{K}\boldsymbol{t}^*(k+1) + \boldsymbol{P}\boldsymbol{x}^*(k) + \boldsymbol{Q}u(k) + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^*(k) = \boldsymbol{L}\boldsymbol{t}^*(k+1) + \boldsymbol{R}\boldsymbol{x}^*(k) + \boldsymbol{S}u(k) + \boldsymbol{\varepsilon}_y(k) \end{cases}$$

where $\boldsymbol{\varepsilon}_t(k)$, $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ are the computational errors.
The output error

$$\Delta\boldsymbol{y}(k) \triangleq \boldsymbol{y}^*(k) - \boldsymbol{y}(k)$$

can be seen as the output of a MIMO filter $\mathcal{H}_\varepsilon$.

SIF: the rigorous filter error bound

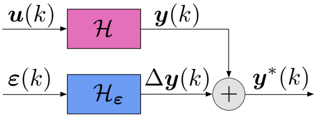
Implemented filter:

$$\mathcal{H}^* \begin{cases} \boldsymbol{J}\boldsymbol{t}^*(k+1) = \boldsymbol{M}\boldsymbol{x}^*(k) + \boldsymbol{N}u(k) + \boldsymbol{\varepsilon}_t(k) \\ \boldsymbol{x}^*(k+1) = \boldsymbol{K}\boldsymbol{t}^*(k+1) + \boldsymbol{P}\boldsymbol{x}^*(k) + \boldsymbol{Q}u(k) + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^*(k) = \boldsymbol{L}\boldsymbol{t}^*(k+1) + \boldsymbol{R}\boldsymbol{x}^*(k) + \boldsymbol{S}u(k) + \boldsymbol{\varepsilon}_y(k) \end{cases}$$

where $\boldsymbol{\varepsilon}_t(k)$, $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ are the computational errors.
The output error

$$\Delta\boldsymbol{y}(k) \triangleq \boldsymbol{y}^*(k) - \boldsymbol{y}(k)$$

can be seen as the output of a MIMO filter $\mathcal{H}_\varepsilon$.



SIF: the rigorous filter error bound

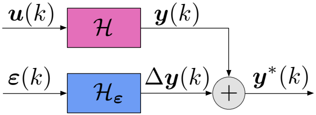
Implemented filter:

$$\mathcal{H}^* \begin{cases} \boldsymbol{J}\boldsymbol{t}^*(k+1) = \boldsymbol{M}\boldsymbol{x}^*(k) + \boldsymbol{N}u(k) + \boldsymbol{\varepsilon}_t(k) \\ \boldsymbol{x}^*(k+1) = \boldsymbol{K}\boldsymbol{t}^*(k+1) + \boldsymbol{P}\boldsymbol{x}^*(k) + \boldsymbol{Q}u(k) + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{y}^*(k) = \boldsymbol{L}\boldsymbol{t}^*(k+1) + \boldsymbol{R}\boldsymbol{x}^*(k) + \boldsymbol{S}u(k) + \boldsymbol{\varepsilon}_y(k) \end{cases}$$

where $\boldsymbol{\varepsilon}_t(k)$, $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ are the computational errors.
The output error

$$\Delta\boldsymbol{y}(k) \triangleq \boldsymbol{y}^*(k) - \boldsymbol{y}(k)$$

can be seen as the output of a MIMO filter $\mathcal{H}_\varepsilon$.



WCPG theorem on $\mathcal{H}_\varepsilon$ gives the output error interval:

$$\Delta\boldsymbol{y}(k) \leqslant \langle\!\langle H_{\boldsymbol{\varepsilon}} \rangle\!\rangle\, \bar{\boldsymbol{\varepsilon}}$$

SIF: code generation

WCPG theorem gives a *rigorous* way to compute Most Significant Bit:

$$\boldsymbol{m}_y = \left\lfloor \log_2 \left(\langle\langle \boldsymbol{H} \rangle\rangle \, \bar{\boldsymbol{u}} \right) \right\rfloor + 1$$

Equivalent technique: WCPG-scaling, it guarantees that no overflows occur.

SIF: code generation

WCPG theorem gives a *rigorous* way to compute Most Significant Bit:

$$\boldsymbol{m}_y = \left\lfloor \log_2 \left(\langle\!\langle \boldsymbol{H} \rangle\!\rangle \, \bar{\boldsymbol{u}} \right) \right\rfloor + 1$$

Equivalent technique: WCPG-scaling, it guarantees that no overflows occur.

Fixed Point Code Generator (FiPoGen)

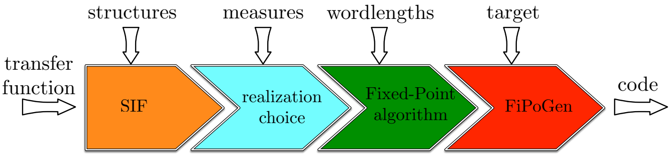
Given wordlength and evaluation scheme

- Generates bit-accurate fixed-point algorithms

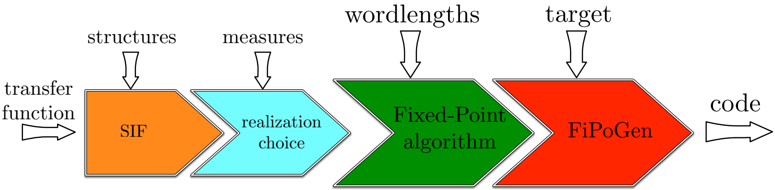
Given only evaluation scheme

- Optimizes the wordlength under certain criteria (e.g. area)
- Generates bit-accurate fixed-point algorithms

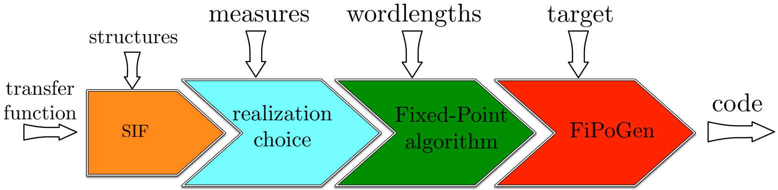
SIF: from transfer function to Fixed-Point code



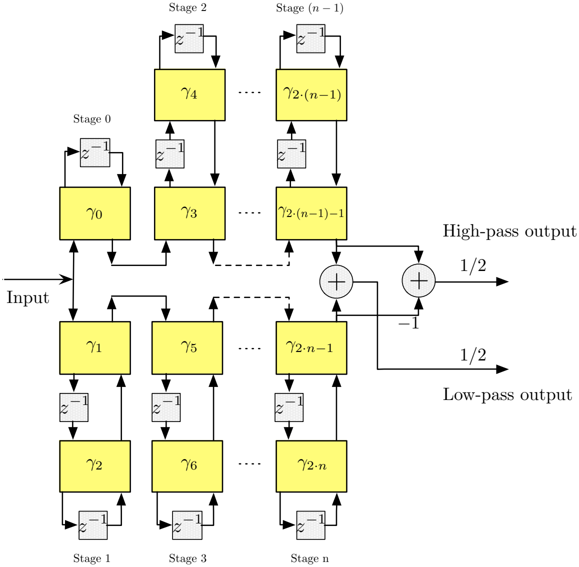
SIF: from transfer function to Fixed-Point code



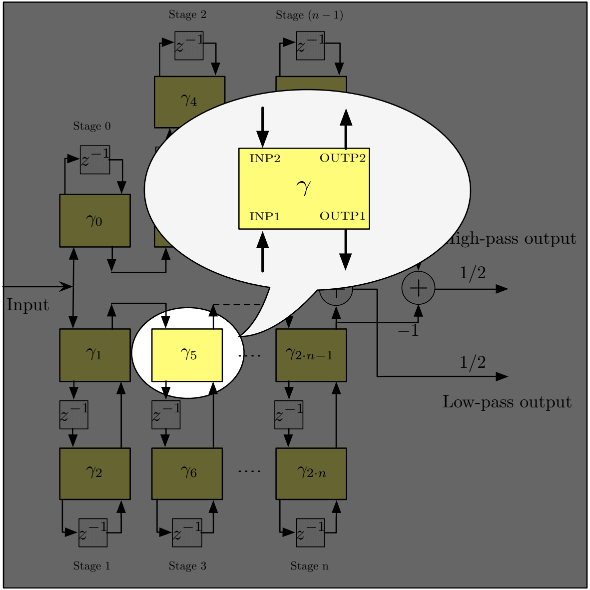
SIF: from transfer function to Fixed-Point code



Lattice Wave Digital Filters

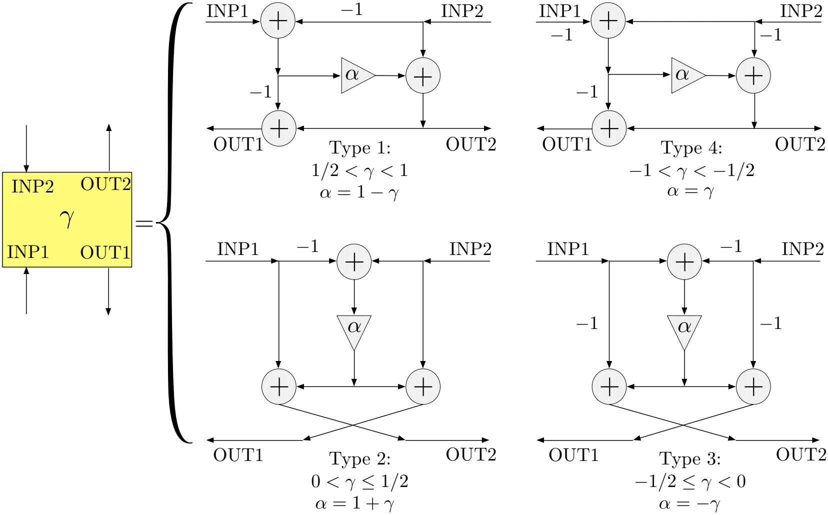


Lattice Wave Digital Filters



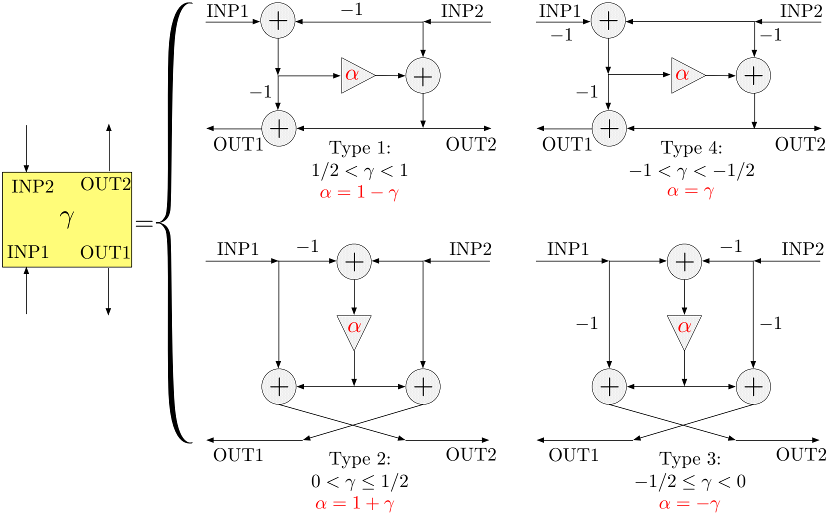
Lattice Wave Digital Filters

Two-port adaptor: Richard's structures



Lattice Wave Digital Filters

Two-port adaptor: Richard's structures



Lattice Wave Digital Filters

Positive sides

- parallelizable
- modular, convenient for VLSI
- often referred to as *stable*

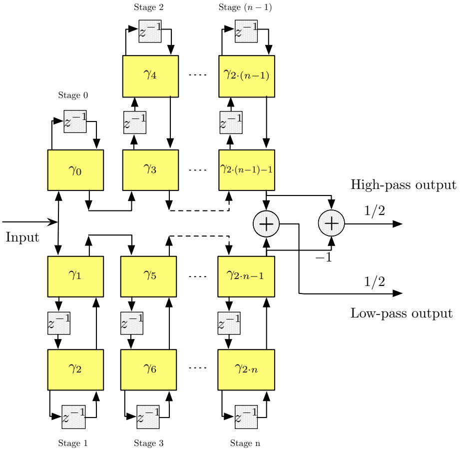
Drawbacks

- Studies of Fixed-Point implementation include complicated infinite-precision optimization
- Comparison is difficult

Objectives

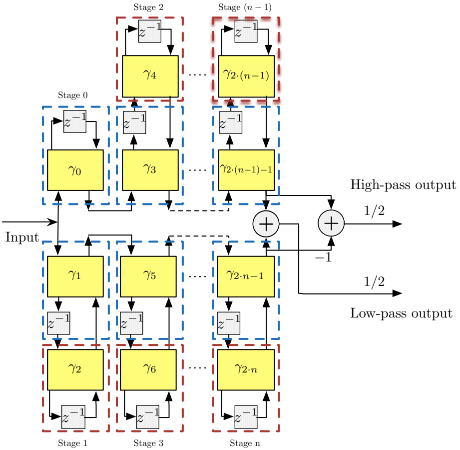
- Represent LWDF in terms of SIF
- Perform *rigorous* error analysis
- Instantly compare with other structures

LWDF-to-SIF convertion



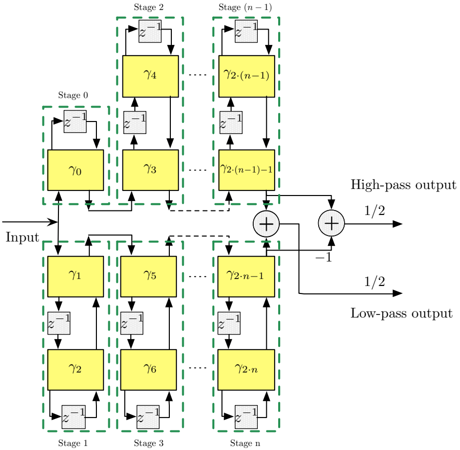
LWDF-to-SIF convertion

1. SIF representation for subsystems of Type A and Type B



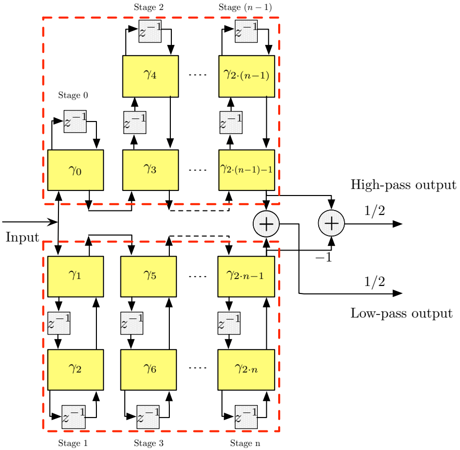
LWDF-to-SIF convertion

1. SIF representation for subsystems of Type A and Type B
2. Cascade subsystems into stages



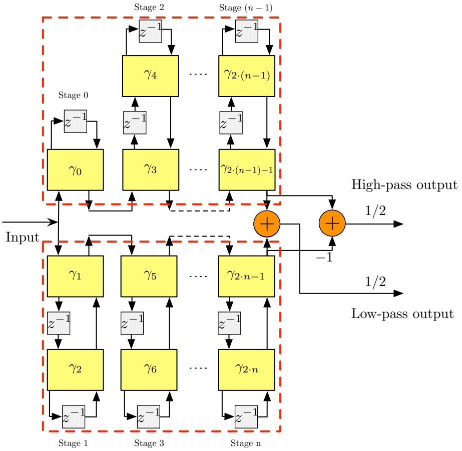
LWDF-to-SIF convertion

1. SIF representation for subsystems of Type A and Type B
2. Cascade subsystems into stages
3. Cascade stages into branches



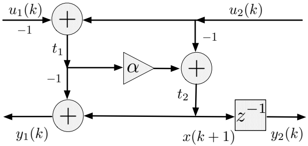
LWDF-to-SIF convertion

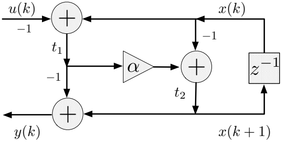
1. SIF representation for subsystems of Type A and Type B
2. Cascade subsystems into stages
3. Cascade stages into branches
4. Cascade branches into low/high pass filter



LWDF-to-SIF convertion: example

Convert DFGs of two adaptors into SIFs:



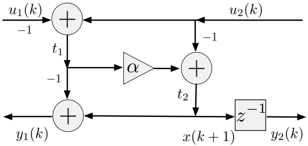


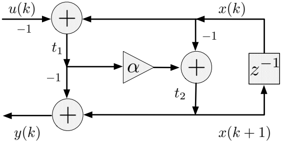
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix}$$

LWDF-to-SIF convertion: example

Convert DFGs of two adaptors into SIFs:





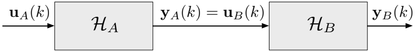
$$\boldsymbol{Z}_A \triangleq \left(\begin{array}{c:c:c} -\boldsymbol{J}_A & \boldsymbol{M}_A & \boldsymbol{N}_A \\ \hdashline \boldsymbol{K}_A & \boldsymbol{P}_A & \boldsymbol{Q}_A \\ \hdashline \boldsymbol{L}_A & \boldsymbol{R}_A & \boldsymbol{S}_A \end{array}\right) = \left(\begin{array}{cc:c:cc} -1 & 0 & 0 & -1 & 1 \\ \alpha & -1 & 0 & 0 & -1 \\ \hdashline 0 & -1 & 0 & 0 & 0 \\ \hdashline -1 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 \end{array}\right)$$

$$\boldsymbol{Z}_B \triangleq \left(\begin{array}{c:c:c} -\boldsymbol{J}_B & \boldsymbol{M}_B & \boldsymbol{N}_B \\ \hdashline \boldsymbol{K}_B & \boldsymbol{P}_B & \boldsymbol{Q}_B \\ \hdashline \boldsymbol{L}_B & \boldsymbol{R}_B & \boldsymbol{S}_B \end{array}\right) = \left(\begin{array}{cc:c:c} -1 & 0 & 1 & -1 \\ \alpha & -1 & 1 & 0 \\ \hdashline 0 & -1 & 0 & 0 \\ \hdashline -1 & 1 & 0 & 0 \end{array}\right)$$

Cascading SIFs:

$\boldsymbol{Z}_1 = \{\boldsymbol{J}_1, \boldsymbol{K}_1, \ldots, \boldsymbol{S}_1\}$

$\boldsymbol{Z}_2 = \{\boldsymbol{J}_2, \boldsymbol{K}_2, \ldots, \boldsymbol{S}_2\}$



Then,

$$\boldsymbol{Z} = \left(\begin{array}{ccc|cc|c} -\boldsymbol{J}_1 & \mathbf{0} & \mathbf{0} & \boldsymbol{M}_1 & \mathbf{0} & \boldsymbol{N}_1 \\ \boldsymbol{L}_1 & -\boldsymbol{I} & \mathbf{0} & \boldsymbol{R}_1 & \mathbf{0} & \boldsymbol{S}_1 \\ \mathbf{0} & \boldsymbol{N}_2 & -\boldsymbol{J}_2 & \mathbf{0} & \boldsymbol{M}_2 & \mathbf{0} \\ \hline \boldsymbol{K}_1 & \mathbf{0} & \mathbf{0} & \boldsymbol{P}_1 & \mathbf{0} & \boldsymbol{Q}_1 \\ \mathbf{0} & \boldsymbol{Q}_2 & \boldsymbol{K}_2 & \mathbf{0} & \boldsymbol{P}_2 & \mathbf{0} \\ \hline \mathbf{0} & \boldsymbol{S}_2 & \boldsymbol{L}_2 & \mathbf{0} & \boldsymbol{R}_2 & \mathbf{0} \end{array}\right)$$

Note: matrix Z is extremely sparse.

Example and comparison

Reference filter: low-pass 5^{th} order Butterworth filter with cutoff frequency 0.1.

Structures for the comparison:

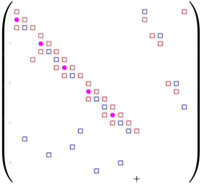
- LWDF
- state-space
- ρ-Direct Form II transposed
- Direct Form I

Normalized (*i.e.* all coefficients have the same wordlength) measures:

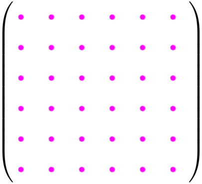
- transfer function error: $\bar{\sigma}^2_{\Delta H}$
- pole error: $\bar{\sigma}^2_{\Delta|\boldsymbol{\lambda}|}$
- output error: $\overline{\Delta_y}$

Example and comparison

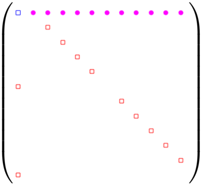
LWDF, Z is 22×22

$\mathbf{Z} =$ 

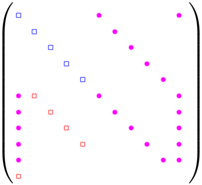
State-space, Z is 12×12

$\mathbf{Z} =$ 

DFI, Z is 12×12

$\mathbf{Z} =$ 

ρDFIIt, Z is 12×12

$\mathbf{Z} =$ 

Example and comparison

Realization	size(Z)	coeff.	$\bar{\sigma}^2_{\Delta H}$	$\bar{\sigma}^2_{\Delta\lvert\boldsymbol{\lambda}\rvert}$	$\overline{\Delta_y}$
LWDF	22×22	5	0. 3151	0.56	122.9
state-space	6× 6	36	1.15	5.75	23.33
ρDFIIt	11×11	11	0.09	0.45	94.3
DFI	12×12	11	1.42e+6	-	7.961

Conclusion and perspectives

Conclusion:

- LWDF converted to SIF
- Normalized sensitivity and output error measures applied
- Comparison with several popular structures presented

Perspectives:

- Use VHDL code generator (FloPoCo) to compare hardware implementations
- Apply ρ-operator to LWDF

Thank you!
Questions?