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

Mathematical function $\rightarrow$ implementation $\rightarrow$ Target
Motivation

Different filter structures:
- Direct Form I, Direct Form II
- State-space
- Wave, Lattice Wave, ...
- $\rho$-operator: $\rho$DFIIIt, $\rho$Modal, $\rho$State-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

1. Motivation
2. Specialized Implicit Framework
3. Lattice Wave Digital Filters
4. LWDF-to-SIF conversion
5. Example and comparison
6. Summary
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

\[
\begin{align*}
J_t(k+1) &= Mx(k) + Nu(k) \\
\mathcal{H} \begin{cases} 
    x(k+1) = K_t(k+1) + Px(k) + Qu(k) \\
    y(k) = Lt(k+1) + Rx(k) + Su(k)
\end{cases}
\end{align*}
\]

Denote $Z$ the matrix containing all the coefficients

\[
Z \triangleq \begin{pmatrix} -J & M & N \\
K & P & Q \\
L & R & S \end{pmatrix}
\]
SIF: measures

Measures

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

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

\[
\forall k \quad y(k) \leq \langle H \rangle \bar{u},
\]

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

\[
\langle H \rangle := |D| + \sum_{k=0}^{\infty} \left| CA^k B \right|.
\]

**Note:** we can compute \( \langle H \rangle \) in arbitrary precision.
SIF: the rigorous filter error bound

Exact filter:

\[
\mathcal{H} \begin{cases} 
J_t(k+1) = Mx(k) + Nu(k) \\
x(k+1) = Kt(k+1) + Px(k) + Qu(k) \\
y(k) = Lt(k+1) + Rx(k) + Su(k)
\end{cases}
\]

where \( \varepsilon_t(k) \), \( \varepsilon_x(k) \) and \( \varepsilon_y(k) \) are the computational errors.

The output error \( \Delta y(k) \) can be seen as the output of a MIMO filter \( H_{\varepsilon} \).

WCPG theorem on \( H_{\varepsilon} \) gives the output error interval:

\( \Delta y(k) \leq \langle \langle H_{\varepsilon} \rangle \rangle \bar{\varepsilon} \)
SIF: the rigorous filter error bound

Implemented filter:

\[
\mathcal{H}^* \begin{cases} 
Jt^*(k+1) = Mx^*(k) + Nu(k) + \varepsilon_t(k) \\
x^*(k+1) = Kt^*(k+1) + Px^*(k) + Qu(k) + \varepsilon_x(k) \\
y^*(k) = Lt^*(k+1) + Rx^*(k) + Su(k) + \varepsilon_y(k)
\end{cases}
\]

where \(\varepsilon_t(k)\), \(\varepsilon_x(k)\) and \(\varepsilon_y(k)\) are the computational errors.
SIF: the rigorous filter error bound

Implemented filter:

$$\mathcal{H}^* \begin{aligned} Jt^*(k + 1) &= Mx^*(k) + Nu(k) + \varepsilon_t(k) \\ x^*(k + 1) &= Kt^*(k + 1) + Px^*(k) + Qu(k) + \varepsilon_x(k) \\ y^*(k) &= Lt^*(k + 1) + Rx^*(k) + Su(k) + \varepsilon_y(k) \end{aligned}$$

where $\varepsilon_t(k)$, $\varepsilon_x(k)$ and $\varepsilon_y(k)$ are the computational errors. The output error

$$\Delta y(k) \triangleq y^*(k) - y(k)$$

can be seen as the output of a MIMO filter $\mathcal{H}_\varepsilon$. 
SIF: the rigorous filter error bound

Implemented filter:

\[
\begin{align*}
\mathcal{H}^* & \quad \left\{ \begin{array}{l}
J^*(k+1) = Mx^*(k) + Nu(k) + \varepsilon_t(k) \\
\dot{x}^*(k+1) = Kt^*(k+1) + Px^*(k) + Qu(k) + \varepsilon_x(k) \\
y^*(k) = Lt^*(k+1) + Rx^*(k) + Su(k) + \varepsilon_y(k)
\end{array} \right.
\end{align*}
\]

where \(\varepsilon_t(k), \varepsilon_x(k)\) and \(\varepsilon_y(k)\) are the computational errors. The output error

\[
\Delta y(k) \triangleq y^*(k) - y(k)
\]

can be seen as the output of a MIMO filter \(\mathcal{H}_\varepsilon\).
SIF: the rigorous filter error bound

Implemented filter:

\[
\mathcal{H}^* \begin{cases} 
J^*(k+1) = M^*(k) + N^*(k) + \varepsilon_t(k) \\
X^*(k+1) = K^*(k+1) + P^*(k) + Q^*(k) + \varepsilon_x(k) \\
Y^*(k) = L^*(k+1) + R^*(k) + S^*(k) + \varepsilon_y(k)
\end{cases}
\]

where \( \varepsilon_t(k) \), \( \varepsilon_x(k) \) and \( \varepsilon_y(k) \) are the computational errors.

The output error

\[
\Delta y(k) \triangleq y^*(k) - y(k)
\]

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

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

\[
\Delta y(k) \leq \left\langle \mathcal{H}_\varepsilon \right\rangle \bar{\varepsilon}
\]
SIF: code generation

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

\[ m_y = \left\lfloor \log_2 (\langle \langle H \rangle \rangle \bar{u}) \right\rfloor + 1 \]

**Equivalent technique:** WCPG-scaling, it guarantees that no overflows occur.
SIF: code generation

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

\[ m_y = \lfloor \log_2 (\langle \langle H \rangle \rangle \bar{u}) \rfloor + 1 \]

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

Stage 0
Stage 2
Stage (n – 1)

Stage 1
Stage 3
Stage n

Input

High-pass output

Low-pass output

1/2
1/2
Lattice Wave Digital Filters
Two-port adaptor: Richard’s structures

Type 1:
\[ \frac{1}{2} < \gamma < 1 \]
\[ \alpha = 1 - \gamma \]

Type 2:
\[ 0 < \gamma \leq \frac{1}{2} \]
\[ \alpha = 1 + \gamma \]

Type 3:
\[ -\frac{1}{2} \leq \gamma < 0 \]
\[ \alpha = -\gamma \]

Type 4:
\[ -1 < \gamma < -\frac{1}{2} \]
\[ \alpha = \gamma \]
Lattice Wave Digital Filters

Two-port adaptor: Richard’s structures

Type 1:
\[ \frac{1}{2} < \gamma < 1 \]
\[ \alpha = 1 - \gamma \]

Type 2:
\[ 0 < \gamma \leq \frac{1}{2} \]
\[ \alpha = 1 + \gamma \]

Type 3:
\[ -\frac{1}{2} \leq \gamma < 0 \]
\[ \alpha = -\gamma \]

Type 4:
\[ -1 < \gamma < -\frac{1}{2} \]
\[ \alpha = \gamma \]
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 conversion

SIF representation for subsystems of Type A and Type B

Cascade subsystems into stages
Cascade stages into branches
Cascade branches into low/high pass filter

\[ z_n = \frac{1}{2} \]

Input

\[ \frac{1}{2} \] +

\[ 1 \]

Low-pass output

\[ 1 \]

High-pass output

\[ 1 \]

Stage 0

Stage 1

Stage 2

Stage (n−1)

Stage n
LWDF-to-SIF conversion

- SIF representation for subsystems of Type A and Type B
LWDF-to-SIF conversion

- SIF representation for subsystems of Type A and Type B
- Cascade subsystems into stages
LWDF-to-SIF conversion

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

- SIF representation for subsystems of Type A and Type B
- Cascade subsystems into stages
- Cascade stages into branches

![Diagram of LWDF-to-SIF conversion](image-url)
LWDF-to-SIF conversion

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 conversion: example

Convert DFGs of two adaptors into SIFs:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
-\alpha & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
1 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
t(k + 1) \\ x(k + 1) \\ y(k)
\end{pmatrix}
= 
\begin{pmatrix}
0 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
t(k) \\ x(k) \\ u(k)
\end{pmatrix}
\]
LWDF-to-SIF conversion: example

Convert DFGs of two adaptors into SIFs:

\[
Z_A \triangleq \begin{pmatrix}
-J_A & M_A & N_A \\
K_A & P_A & Q_A \\
L_A & R_A & S_A
\end{pmatrix} = \begin{pmatrix}
-1 & 0 & 0 & -1 & 1 \\
\alpha & -1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]

\[
Z_B \triangleq \begin{pmatrix}
-J_B & M_B & N_B \\
K_B & P_B & Q_B \\
L_B & R_B & S_B
\end{pmatrix} = \begin{pmatrix}
-1 & 0 & 1 & -1 \\
\alpha & -1 & 1 & 0 \\
0 & -1 & 0 & 0 \\
-1 & 1 & 0 & 0
\end{pmatrix}
\]
Cascading SIFs:
\[ Z_1 = \{J_1, K_1, \ldots, S_1\} \]
\[ Z_2 = \{J_2, K_2, \ldots, S_2\} \]

Then,
\[
Z = \begin{pmatrix}
-J_1 & 0 & 0 & M_1 & 0 & N_1 \\
L_1 & -I & 0 & R_1 & 0 & S_1 \\
0 & N_2 & -J_2 & 0 & M_2 & 0 \\
K_1 & 0 & 0 & P_1 & 0 & Q_1 \\
0 & Q_2 & K_2 & 0 & P_2 & 0 \\
0 & S_2 & L_2 & 0 & R_2 & 0
\end{pmatrix}
\]

Note: matrix \( Z \) is extremely sparse.
Example and comparison

Reference filter: low-pass 5\textsuperscript{th} order Butterworth filter with cutoff frequency 0.1.
Structures for the comparison:

- LWDF
- state-space
- $\rho$-Direct Form II transposed
- Direct Form I

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

- transfer function error: $\bar{\sigma}_{\Delta H}^2$
- pole error: $\bar{\sigma}_{\Delta |\lambda|}^2$
- output error: $\Delta y$
Example and comparison

LWDF, $Z$ is $22 \times 22$

\[
Z = \begin{pmatrix}
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
+ & + & + \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\end{pmatrix}
\]

State-space, $Z$ is $12 \times 12$

\[
Z = \begin{pmatrix}
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\end{pmatrix}
\]

DFI, $Z$ is $12 \times 12$

\[
Z = \begin{pmatrix}
\vdots & \vdots \\
\vdots & \vdots \\
\vdots & \vdots \\
\vdots & \vdots \\
\vdots & \vdots \\
\vdots & \vdots \\
\vdots & \vdots \\
\end{pmatrix}
\]

$\rho$DFIIt, $Z$ is $12 \times 12$

\[
Z = \begin{pmatrix}
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\end{pmatrix}
\]
## Example and comparison

| Realization | size(Z) | coeff. | $\sigma^2_{\Delta H}$ | $\sigma^2_{\Delta|\lambda|}$ | $\overline{\Delta_y}$ |
|-------------|---------|--------|----------------------|-----------------------------|------------------------|
| LWDF        | 22×22   | 5      | 0.3151              | 0.56                        | 122.9                  |
| state-space | 6×6     | 36     | 1.15                | 5.75                        | 23.33                  |
| $\rho_{DFIIt}$ | 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 $\rho$-operator to LWDF
Thank you!

Questions?