RELIABLE FIXED-POINT IMPLEMENTATION OF LINEAR DATA-FLOWS

Thibault HILAIRE, Anastasia VOLKOVA and Maminionja RAVOSON

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

Motivation

Implementation of Linear Time-Invariant systems in finite precision:

- Coefficient quantization, computational errors and their propagation
- Software/Hardware implementation under constraints
- Large variety of possible structures (e.g. Direct Forms, Lattices, etc.) described by different means (graphical, analytical)
- Most error analysis are based on statistical models ⇒ no guarantee on finite-precision implementation

Contributions

Automatic Fixed-Point code Generator for filters for which we:

- Use unified *analytical* representation of linear data-flows
- Can describe any analytical and graphical representation with our framework
- Adopted numerous classical and developed new quality measures
- Provide **fully rigorous** and **reliable** implementation in Fixed-Point arithmetic
- Generate C and VHDL code (for ASICs and FPGAs)

Perspectives

We plan to:

- Perform a reliable verification of specification-to-implementation correspondance
- Optimize Software/Hardware implementation for various constraints using our rigorous approach
- Wrap the generator with optimization routines to create a complete and efficient filter-to-code tool

SIF

Specialized **I**mplicit **F**orm is an analytical matrix-based representation of input/output relationship:

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

- ⋄ $\boldsymbol{u}(k)$ - inputs ⋄ $\boldsymbol{t}(k)$ - temp. variables
- ⋄ $\boldsymbol{y}(k)$ - outputs ⋄ $\boldsymbol{x}(k)$ - states

Some properties of SIF:

- Easy algebraic computations
- Can describe any linear data flow
- Unifies analysis and implementation

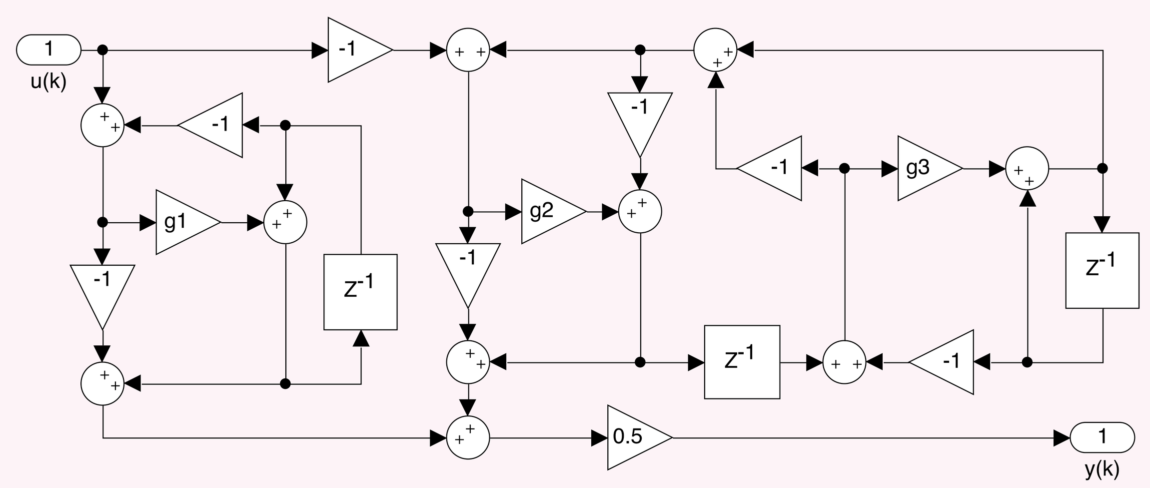
Important: order of computations is preserved in the lower-triangular matrix $\boldsymbol{J}$. For example,

$$\boldsymbol{y} = \boldsymbol{M}_2 (\boldsymbol{M}_1 \boldsymbol{x})$$

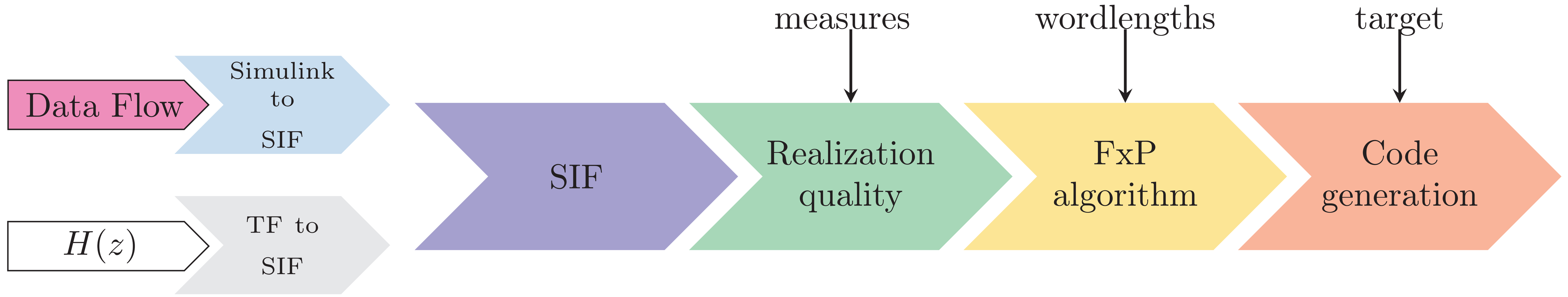
is described using temporary variable $\boldsymbol{t}$ with

$$\begin{pmatrix} \boldsymbol{I} & 0 \\ -\boldsymbol{M}_2 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{t} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_1 \\ 0 \end{pmatrix} \boldsymbol{x}$$

Linear Data Flow



Example of a linear data flow: Lattice Wave Digital Filter



Simulink-to-SIF

Step 1: Label variables

- states ⟵ delays
- SIF coefficients ⟵ gain elements
- temporary variables ⟵ outputs of gain and sum operators

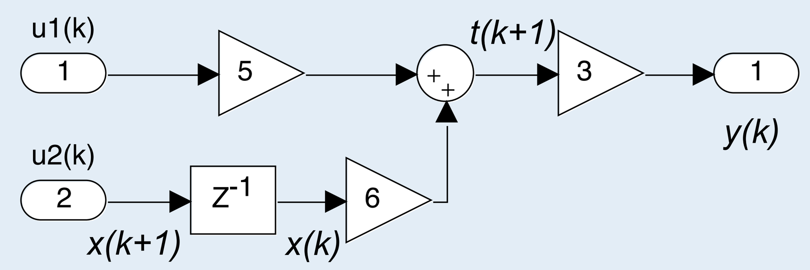
Step 2: Form the Sums-of-Products

- merge all directly cascading blocks
- regroup successive sums
- flatten design if subsystem is present

Step 3: Preserve the order of computations

- topological sort of temporary variables
- remove unnecessary variables

For example, a data-flow



Corresponds to SIF equations:

$$\begin{cases} t(k+1) &= 6 \cdot x(k) + 5 \cdot u_1(k) \\ x(k+1) &= 1 \cdot u_2(k) \\ y &= 3 \cdot t(k+1) \end{cases}$$

Quality measures

Classical measures

- Based on sensitivity wrt. the coefficients: → transfer function and pole/zero sensitivities
- Signal to Quantization Noise Ratio → errors modeled as noises ($\|\mathcal{H}_\Delta\|_2$)

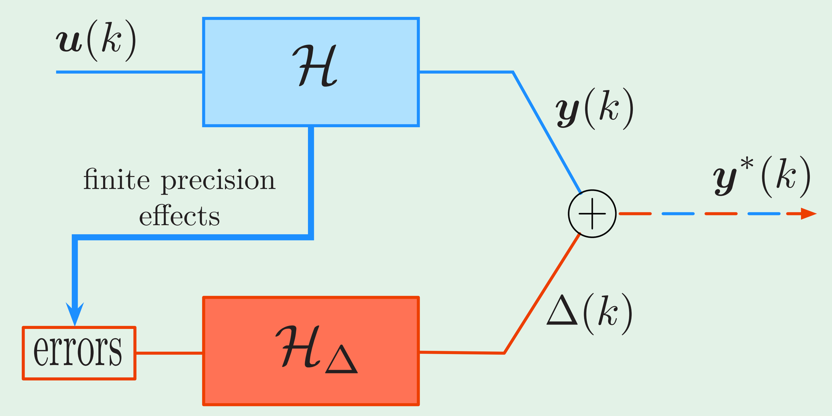
New measures: rigorous approach

Theorem 1. *For a stable LTI system $\mathcal{H}$, if $\forall k\ |\boldsymbol{u}(k)| \leq \bar{\boldsymbol{u}}$, then the output is bounded*

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

where $\langle\langle \mathcal{H} \rangle\rangle$ is the Worst-Case Peak Gain (WCPG) matrix of the system.

Theorem 2. *The finite precision output $\boldsymbol{y}^*(k)$ is bounded by the outputs of the exact and special error filters:* $|\boldsymbol{y}^*(k)| \leq |\boldsymbol{y}(k)| + |\boldsymbol{\Delta}(k)|$



$\mathcal{H}_\Delta$ shows the propagation of finite precision effects.

Fixed-Point Implementation

Most Significant Bit determination

Reliable approach based on mathematical proofs:

Step 1: Determine output interval using our rigorous evaluation of the WCPG in arbitrary precision.

Step 2: Deduce the Fixed-Point implementation parameters while taking into account the propagation of computational errors through the filter.

Important: we prove that no overflow occurs and MSB positions are overestimated at most by one.

Sum-of-Products

The computations involves sums of products by constants. They can be performed with **faithful rounding** using $\lceil \log_2 N \rceil$ guard bits.

Least Significant Bit determination

The output error is analytically determined from the word-lengths $\boldsymbol{w}$

$$\bar{\boldsymbol{\Delta}} = 2 \langle\langle \mathcal{H}_\Delta \rangle\rangle \left(\left\lceil \langle\langle \mathcal{H} \rangle\rangle \bar{\boldsymbol{u}} \right\rceil_2 \times 2^{-\boldsymbol{w}} \right)$$

The word-length optimization problem can be solved with a Mixed-Integer Non Linear Programming solver.

Code Generation

Our tool can then generate code:

- C code for μCs and DSPs
- VHDL for ASICs (using Stratus[a])
- VHDL for FPGAs (using FloPoCo[b])

[a] https://soc-extras.lip6.fr/en/coriolis/
[b] Flopoco: http://flopoco.gforge.inria.fr/