Signal processing and filters
OOOOOOOO

Generating the architecture
OOOOOOO

Computing the parameters
OOOO

Future Work

Architecture Synthesis
for
Linear Time-Invariant Filters

Antoine Martinet

CITI lab,
INRIA's SOCRATE Team,

Under the supervision of
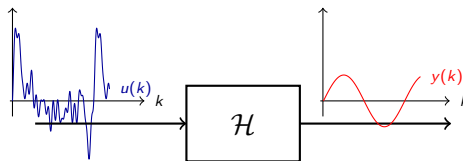Florent de Dinechin

2 February - 31 July, 2015

Table of Contents

Table of Contents

Introduction: LTI Filters

Signal: temporal variable $x(k)$ with $\{x(k)\}_{k \geq 0} \in \mathbb{R}$

LTI (Linear Time Invariant) filters are particular filters that are:

- Linear: outputs are linear combinations of inputs (allows to use linear algebra definitions)
- Time-Invariant: all coefficients are constant

Purpose of this work

What is the purpose of this work?

- Lopez's PhD thesis studies how to compute LTI filters in software:
    - scheduling issues
    - fixed size
- The goal here is to do such a work in hardware, where we have more flexibility:
    - full parallelism
    - arbitrary size

In this context, constraints become degrees of freedom.

FIR, IIR

FIR definition:

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) \qquad (1)$$
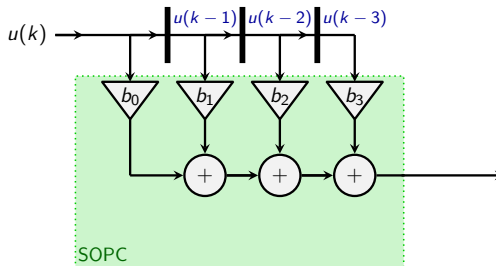
Abstract architecture for the direct form realization of an FIR filter

FIR, IIR

IIR definition:

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

Abstract architecture for the direct form realization of an IIR filter

FIR, IIR

IIR definition:

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

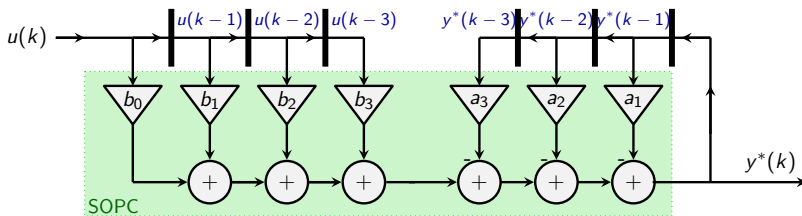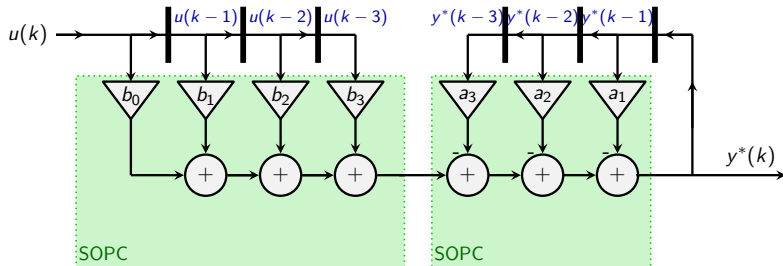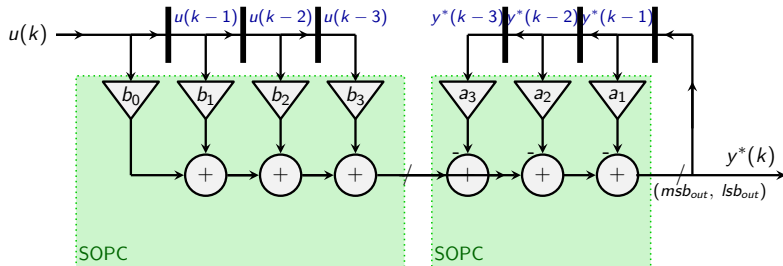Abstract architecture for the direct form realization of an IIR filter

FIR, IIR

IIR definition:

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

Abstract architecture for the direct form realization of an IIR filter

State-Space representation
The "ABCD" form

Let's define $\boldsymbol{x}(k)$ a state vector (hardware register)

$$\begin{cases} \boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k+1) = \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{2}$$

With:

$$\boldsymbol{A} \in \mathbb{R}^{n_x \times n_x} \ , \ \boldsymbol{B} \in \mathbb{R}^{n_x \times n_u} \ ,$$
$$\boldsymbol{C} \in \mathbb{R}^{n_y \times n_x} \ , \ \boldsymbol{D} \in \mathbb{R}^{n_y \times n_u}$$

Equivalent matrix formulation:

$$\begin{pmatrix} \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k+1) \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \tag{3}$$

The SIF: a unified realization representation

Problems with ABCD:

- the whole analysis on precisions has to be rebuilt for each new filter.
- this doesn't gives an explicit order in the operations

The SIF generalizes the state-space.

Addition: $t(k)$ describes the operations order:

$$
\begin{pmatrix} J & 0 & 0 \\ -K & I_{n_x} & 0 \\ -L & 0 & I_{n_y} \end{pmatrix} \begin{pmatrix} t(k+1) \\ x(k+1) \\ y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} t(k) \\ x(k) \\ u(k) \end{pmatrix} \quad (4)
$$

The SIF as an algorithm

for *int $i = 0$; $i \leq n_t$; $i{+}{+}$* **do**

$\quad \left| \quad \boldsymbol{t}_i(k+1) \leftarrow - \sum_{j<i} \boldsymbol{J}_{ij}\boldsymbol{t}_j(k+1) + \sum_{j=1}^{n_x} \boldsymbol{M}_{ij}\boldsymbol{x}_j(k) + \sum_{j=1}^{n_u} \boldsymbol{N}_{ij}\boldsymbol{u}_j(k) \right.$

end

for *int $i = 0$; $i \leq n_x$; $i{+}{+}$* **do**

$\quad \left| \quad \boldsymbol{x}_i(k+1) \leftarrow \sum_{j=1}^{n_t} \boldsymbol{K}_{ij}\boldsymbol{t}_j(k+1) + \sum_{j=1}^{n_x} \boldsymbol{P}_{ij}\boldsymbol{x}_j(k) + \sum_{j=1}^{n_u} \boldsymbol{Q}_{ij}\boldsymbol{u}_j(k) \right.$

end

for *int $i = 0$; $i \leq ny$; $i{+}{+}$* **do**

$\quad \left| \quad \boldsymbol{y}_i(k) \leftarrow \sum_{j=1}^{n_t} \boldsymbol{L}_{ij}\boldsymbol{t}_j(k+1) + \sum_{j=1}^{n_x} \boldsymbol{R}_{ij}\boldsymbol{x}_j(k) + \sum_{j=1}^{n_u} \boldsymbol{S}_{ij}\boldsymbol{u}_j(k) \right.$

end

Algorithm 1: Computation of SIF outputs from inputs

Signal processing and filters · OOOOOOOO · Generating the architecture · OOOOOOO · Computing the parameters · OOOO · Future Work

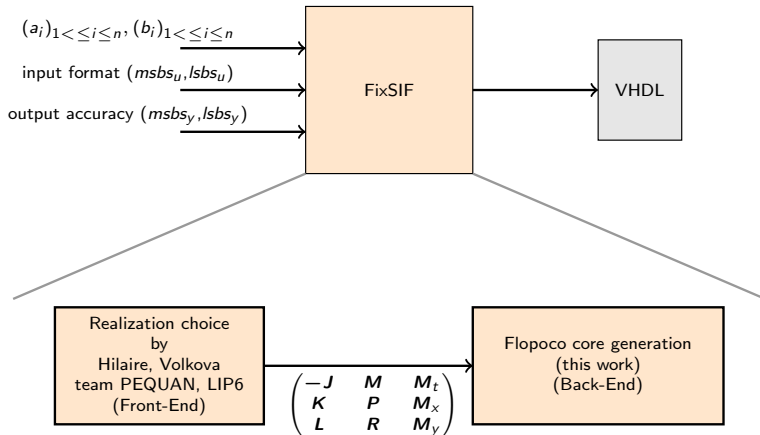The SIF: a unified realization representation

A more common notation

An easier way to communicate SIFs:
The Z matrix:

$$Z = \begin{pmatrix} -J & M & N \\ K & P & Q \\ L & R & S \end{pmatrix} \qquad (5)$$

Workflow

Workflow overview of tools usage

Table of Contents

Signal processing and filters   Generating the architecture   Computing the parameters   Future Work
○○○○○○○○○   ●○○○○○○○   ○○○○
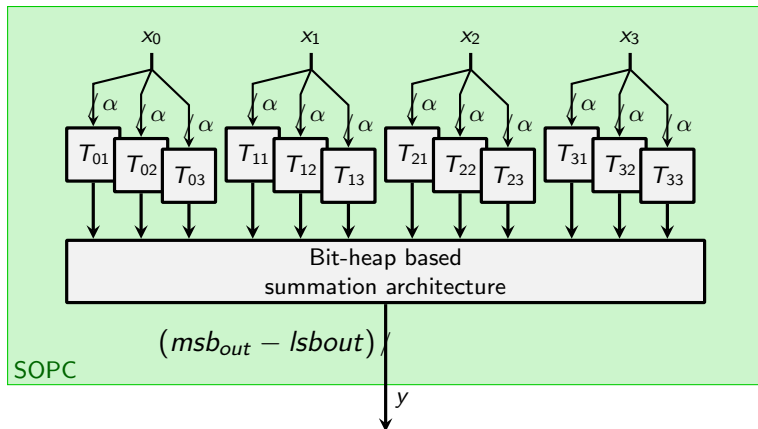The FloPoCo Framework: computing just right

The FloPoCo Framework: computing just right

FloPoCo:

- C++ framework
- Target: FPGAs
- Work: generating arithmetical cores in VHDL computing just right
- Reference: http://flopoco.gforge.inria.fr/

Basic block: SOPC

SOPC architecture

14/36

Architecture generation algorithm

Assuming all the MSBs and LSBs are known. **for** *i=1; i=Z.size(); i++* **do**

> row[ ]= Z[i][ ] //pick first row of Z
> **for** *j=1; j=1; j=Z.size() j++* **do**
> > assign(SOPC[i], row[j], "T","X","U",[msbs,lsbs][i][j])
>
> **end**
> Second pass for wiring.

end

Algorithm 2: Architecture Generation Algorithm

Signal processing and filters
○○○○○○○○

Generating the architecture
○○○●○○○○

Computing the parameters
○○○○

Future Work

Architecture generation

Example

Example
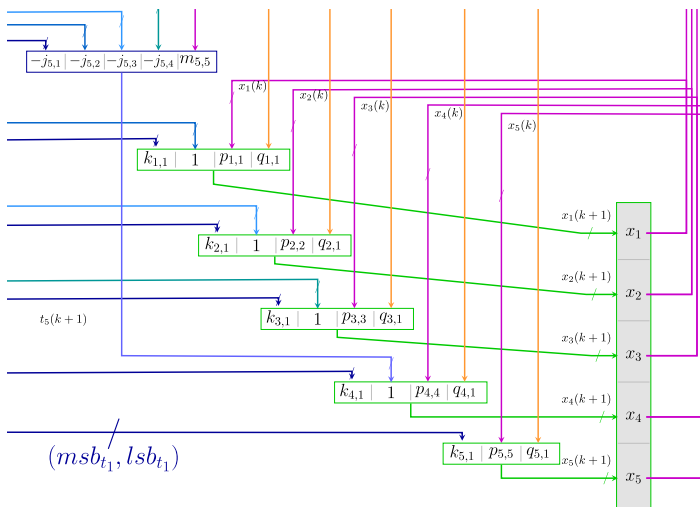
$$
Z = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & m_{1,1} & 0 & 0 & 0 & 0 & 0 \\
j_{2,1} & 1 & 0 & 0 & 0 & 0 & m_{2,2} & 0 & 0 & 0 & 0 \\
j_{3,1} & j_{3,2} & 1 & 0 & 0 & 0 & 0 & m_{3,3} & 0 & 0 & 0 \\
j_{4,1} & j_{4,2} & j_{4,3} & 1 & 0 & 0 & 0 & 0 & m_{4,4} & 0 & 0 \\
j_{5,1} & j_{5,2} & j_{5,3} & j_{5,4} & 1 & 0 & 0 & 0 & 0 & m_{5,5} & 0 \\
k_{1,1} & 1 & 0 & 0 & 0 & p_{1,1} & 0 & 0 & 0 & 0 & q_{1,1} \\
k_{2,1} & 0 & 1 & 0 & 0 & 0 & p_{2,2} & 0 & 0 & 0 & q_{2,1} \\
k_{3,1} & 0 & 0 & 1 & 0 & 0 & 0 & p_{3,3} & 0 & 0 & q_{3,1} \\
k_{4,1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & p_{4,4} & 0 & q_{4,1} \\
k_{5,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{5,5} & q_{5,1} \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Each line will be a SOPC

Signal processing and filters
○○○○○○○○○

Generating the architecture
○○○○○○●○

Computing the parameters
○○○○

Future Work
○○○○

Architecture generation

Example

Signal processing and filters
○○○○○○○○

Generating the architecture
○○○○○○○●

Computing the parameters
○○○○

Future Work
○○○○

Architecture generation

Example

Table of Contents

Signal processing and filters | Generating the architecture | Computing the parameters | Future Work
○○○○○○○○ | ○○○○○○○ | ●○○○ |

Computing the MSB
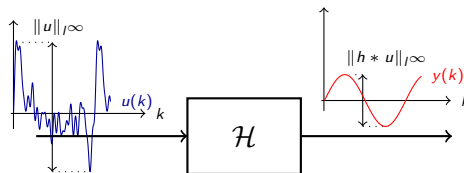
Computing the MSB

$$\|\mathcal{H}\|_{wcpg} = \sup_{u \neq 0} \frac{\|h * u\|_{l\infty}}{\|u\|_{l\infty}} \tag{6}$$

This computation is done by colleagues in LIP6.

Computing the LSB: Point of view about error

ε is the error introduced by the SOPC (closely related to 2^{lsb_i}).
ε^* is the total error (taking the loopback into account)
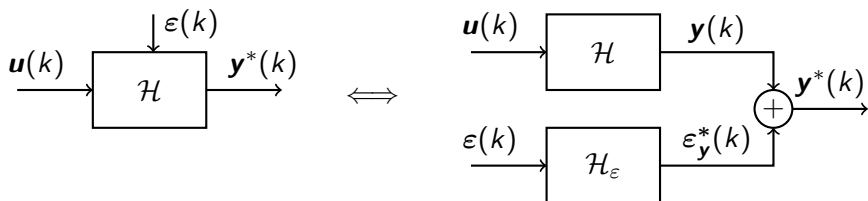
A signal view of the error propagation with respect to the ideal filter

Computing the LSB: Point of view about error

ε is the error introduced by the SOPC (closely related to 2^{lsb_i}).
ε^* is the total error (taking the loopback into account)

A signal view of the error propagation with respect to the ideal filter

Computing the LSB: Impact of errors

for *int i = 0 ; i ≤ n_t; i++* **do**

$\quad$ $\boldsymbol{t}_i(k+1) \leftarrow$

$\quad$ $-\sum\limits_{j<i} \boldsymbol{J}_{ij}\boldsymbol{t}_j(k+1) + \sum\limits_{j=1}^{n_x} \boldsymbol{M}_{ij}\boldsymbol{x}_j(k) + \sum\limits_{j=1}^{n_u} \boldsymbol{N}_{ij}\boldsymbol{u}_j(k) + \varepsilon_{t_i}(k)$

end

for *int i = 0 ; i ≤ n_x; i++* **do**

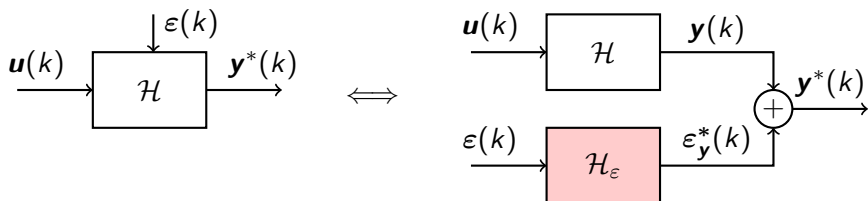$\quad$ $\boldsymbol{x}_i(k+1) \leftarrow \sum\limits_{j=1}^{n_t} \boldsymbol{K}_{ij}\boldsymbol{t}_j(k+1) + \sum\limits_{j=1}^{n_x} \boldsymbol{P}_{ij}\boldsymbol{x}_j(k) + \sum\limits_{j=1}^{n_u} \boldsymbol{Q}_{ij}\boldsymbol{u}_j(k) + \varepsilon_{x_i}(k)$

end

for *int i = 0 ; i ≤ ny; i++* **do**

$\quad$ $\boldsymbol{y}_i(k) \leftarrow \sum\limits_{j=1}^{n_t} \boldsymbol{L}_{ij}\boldsymbol{t}_j(k+1) + \sum\limits_{j=1}^{n_x} \boldsymbol{R}_{ij}\boldsymbol{x}_j(k) + \sum\limits_{j=1}^{n_u} \boldsymbol{S}_{ij}\boldsymbol{u}_j(k) + \varepsilon_{y_i}(k)$

end

Algorithm 3: Computation of SIF outputs from inputs

Computing the LSB: current solution

Let's define:

$$\mathbf{v}' = \begin{pmatrix} \mathbf{t}(k+1) \\ \mathbf{x}(k+1) \\ \mathbf{y}(k) \end{pmatrix} \tag{7}$$

Main constraint:

$$\varepsilon_v < 2^{-lsb_v} \tag{8}$$

Transformed into the following constraint:

$$|\langle\langle \mathcal{H}_\varepsilon \rangle\rangle| \cdot 2^{lsb_{v'}+1} < 2^{-lsb_{y_i}} \tag{9}$$

Lopez gives a simple solution that matches the fixed size constraint. The solution matching the hardware constraint (arbitrary precision) is work in progress.

Table of Contents

Signal processing and filters
00000000

Generating the architecture
0000000

Computing the parameters
0000

Future Work

Future Work

- Removal of power of two: adapt KCM and SOPC FloPoCo cores
- Sub-filter detection: open question for either Front-End and Back-End
- Precision calculations improvement
- File format re-specification

Conclusion

To conclude

- Adapting LTI Filters generation from software to hardware
- Reuse of Lopez's calculations in our context
- Implementation for FPGAs in a parametric view

Signal processing and filters
○○○○○○○○

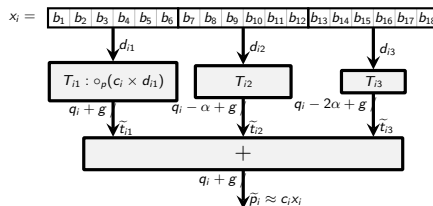Generating the architecture
○○○○○○○

Computing the parameters
○○○○

Future Work

Any question?

Full definition of the SIF

$$
\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_{n_x} & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_{n_y} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \quad (10)
$$

With:

$$
\boldsymbol{J} \in \mathbb{R}^{n_t \times n_t}, \boldsymbol{M} \in \mathbb{R}^{n_t \times n_x}, \boldsymbol{N} \in \mathbb{R}^{n_t \times n_u},
$$
$$
\boldsymbol{K} \in \mathbb{R}^{n_x \times n_t}, \boldsymbol{P} \in \mathbb{R}^{n_x \times n_x}, \boldsymbol{Q} \in \mathbb{R}^{n_x \times n_u}, \quad (11)
$$
$$
\boldsymbol{L} \in \mathbb{R}^{n_y \times n_t}, \boldsymbol{R} \in \mathbb{R}^{n_y \times n_x}, \boldsymbol{S} \in \mathbb{R}^{n_y \times n_u},
$$

$$(12)$$

KCM multiplier

The FixRealKCM method when x_i is split in 3 chunks

Mathematical definition of a filter $\mathcal{H}$

Definition of a filter: $\boldsymbol{y} = \mathcal{H}(\boldsymbol{u})$ With $dim(\boldsymbol{y}) = n_y$ and, $dim(\boldsymbol{u}) = n_u$
Linearity:

$$\mathcal{H}(\alpha \cdot \boldsymbol{u}_1 + \beta \cdot \boldsymbol{u}_2) = \alpha \cdot \mathcal{H}(\boldsymbol{u}_1) + \beta \cdot \mathcal{H}(\boldsymbol{u}_2)$$

Time invariance:

$$\{\mathcal{H}(\boldsymbol{u})(k - k_0)\}_{k \geq 0} = \mathcal{H}(\{\boldsymbol{u}(k - k_0)\}_{k \geq 0})$$

Impulse response

$$u = \sum_{i \geq 0} u(l)\delta_l$$

Where δ_l is a Dirac impulsion centered in l:

$$\delta_l(k) = \begin{cases} 1 & \text{when } k = l \\ 0 & \text{else} \end{cases} \tag{13}$$

Time invariance gives: $\mathcal{H}(\delta_l)(k) = h(k - l)$

Computation of the outputs:

$$y_i(k) = \sum_{j=1}^{n_u} \sum_{l=0}^{k} u_j(l) h_{i,j}(k - l), \ \forall 1 \leq i \leq n_y$$

From:

$$Z = \begin{pmatrix} -J & M & N \\ K & P & Q \\ L & R & S \end{pmatrix} \tag{14}$$

The ABCD form is deducible from the SIF:

$$A_Z = KJ^{-1}M + P, \ \ B_Z = KJ^{-1}N + Q,$$
$$C_Z = LJ^{-1}M + R, \ \ D_Z = LJ^{-1}N + S,$$

$$\tag{15}$$

with:

$$A_Z \in \mathbb{R}^{n_x \times n_x}, B_Z \in \mathbb{R}^{n_x \times n_u},$$
$$C_Z \in \mathbb{R}^{n_y \times n_x}, D_Z \in \mathbb{R}^{n_y \times n_u}, \tag{16}$$

Definition of the error filter:

$$Z_\varepsilon = \begin{pmatrix} -J & M & M_t \\ K & P & M_x \\ L & R & M_y \end{pmatrix} \qquad (17)$$

with:

$$M_t = (I_{n_t} \quad 0_{n_t \times n_x} \quad 0_{n_t \times n_y}), \qquad (18)$$

$$M_x = (0_{n_x \times n_t} \quad I_{n_x} \quad 0_{n_x \times n_y}), \qquad (19)$$

$$M_y = (0_{n_y \times n_t} \quad 0_{n_y \times n_x} \quad I_{n_y}), \qquad (20)$$

Interface specification:

X | c
x_1_1 x_1_2 ... x_1_c
x_2_1 x_2_2 ... x_2_c
.
.
.
x_l_1 x_l_2 ... x_l_c

Where:

- X = name of the matrix
- $X \in \{$ J, K, L, M, N, P, Q, R, S, T $\}$
- x_i_j = the coefficient
- $i \in [1, l]$
- $j \in [1, c]$
- l = the number of lines
- c = the numer of columns.

Computing the LSB: Error definition

Let's define:

$$\boldsymbol{v'} = \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} \tag{21}$$

Errors introduced by SOPCs:

$$\boldsymbol{\varepsilon}_{v'}(k) = \begin{pmatrix} \boldsymbol{\varepsilon}_t(k) \\ \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\varepsilon}_y(k) \end{pmatrix} = \begin{pmatrix} \varepsilon_{t_1}(k) \\ \varepsilon_{t_2}(k) \\ \vdots \\ \varepsilon_{t_{n_t}}(k) \\ \varepsilon_{x_1}(k) \\ \varepsilon_{x_2}(k) \\ \vdots \\ \varepsilon_{x_{n_x}}(k) \\ \varepsilon_{y_1}(k) \\ \varepsilon_{y_2}(k) \\ \vdots \\ \varepsilon_{y_{n_y}}(k) \end{pmatrix} \tag{22}$$

$\boldsymbol{\varepsilon}_{v'}^*(k)$ will represent the total error and is defined similarly to $\boldsymbol{\varepsilon}_{v'}(k)$

References I

Small bibliography

K.D. Chapman.
Fast integer multipliers fit in FPGAs (EDN 1993 design idea winner).
EDN magazine, 39(10):80, May 1993.

S. Chevillard, M. Joldeş, and C. Lauter.
Sollya: An environment for the development of numerical codes.
In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 28–31, Heidelberg, Germany, September 2010. Springer.

Florent de Dinechin, Matei Istoan, and Abdelbassat Massouri.
Sum-of-product architectures computing just right.
In *Application-Specific Systems, Architectures and Processors (ASAP)*. IEEE, 2014.

Florent de Dinechin and Bogdan Pasca.
Designing custom arithmetic data paths with FloPoCo.
IEEE Design & Test of Computers, 28(4):18–27, July 2011.

Thibaut Hilaire.
Analyse et synthèse de l'implémentation de lois de contrôle-commande en précision finie.
PhD thesis, Université de Nantes, 2006.

References II

P.Chawdhry I.Njabeleke, R.Pannett and C.Burrows.
Design of h-infiny loop-shaping controllers for fluid power systems.
In *IEEE Colloquium Robust Control - Theory, Software and Applications*, 1997.

J.F.Whidborne and R.H.Istepanian.
Reduction of controller fragility by pole sensivity minimization.
In *Int. J.Control*, 2000.

G.Li J.Wu, S.Chen and J.Chu.
Constructing sparse realizations of finite precision digital controllers based on a closed-loop stability related
measure.
In *IEEE Proc. Control Theory and Applications*, 2003.

Benoit Lopez.
Implémentation optimale de filtres linéaires en arithmétique virgule fixe.
PhD thesis, Université Paris VI, 2014.

P.Chevel T.Hilaire and J.F.Whidbornz.
A unifying framework for finite wordlength realizations.
In *IEEE*, 2007.

P.Chevel T.Hilaire and Y.Trinquet.
Implicit state-space representation: a unifying framework for FWL implementation of LTI systems.
In *Proc. of the 16th IFAC World Congress.*, 2005.

References III

A. Volkova, T. Hilaire, and C. Lauter.

Reliable evaluation of the Worst-Case Peak Gain matrix in multiple precision.
In *IEEE Symposium on Computer Arithmetic*, 2015.