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

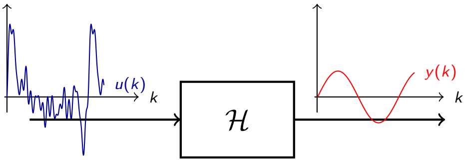
Table of Contents

Table of Contents

Introduction: LTI Filters

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?

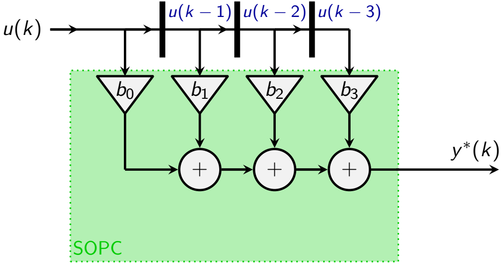
- Lopez's PhD thesis states how to compute LTI filters in software:
  - scheduling issues
  - fixed size
- The goal here is to do such a work in hardware, where we have more flexibility:
  - full parallelism
  - arbitrary size

In this context, constraints become degrees of freedom.

FIR, IIR

FIR definition:

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

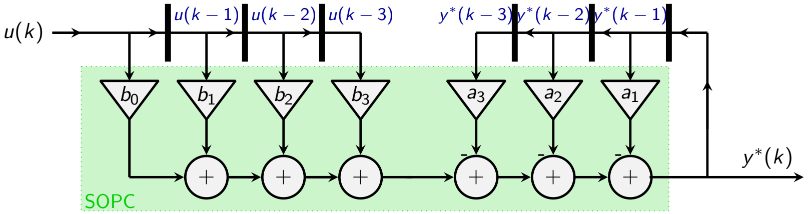


Abstract architecture for the direct form realization of an FIR filter

FIR, IIR

IIR definition:

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

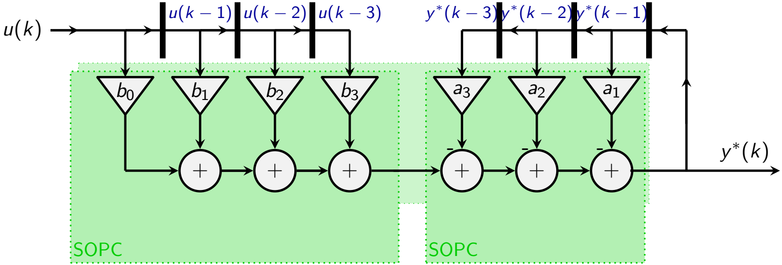


Abstract architecture for the direct form realization of an IIR filter

FIR, IIR

IIR definition:

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

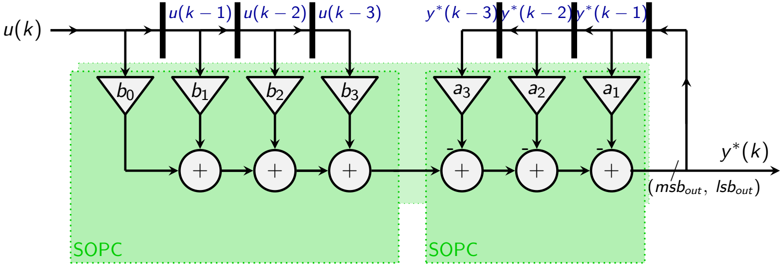


Abstract architecture for the direct form realization of an IIR filter

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

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:

$$\begin{array}{c} \boldsymbol{A} \in \mathbb{R}^{n_x \times n_x} \;, \; \boldsymbol{B} \in \mathbb{R}^{n_x \times n_u} \;, \\ \boldsymbol{C} \in \mathbb{R}^{n_y \times n_x} \;, \; \boldsymbol{D} \in \mathbb{R}^{n_y \times n_u} \end{array}$$

Equivalent matrix formulation:

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

Definition of the SIF

Problems with ABCD:

- *lsb* and *msb* computations have to be rebuilt for each new filter in this form.
- this doesn't gives an explicit order in the operations

The SIF generalizes the state-space.
Addition: $\boldsymbol{t}(k)$ describes the operations order:

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

The SIF as an algorithm

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

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

end

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

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

end

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

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

end

Algorithm 1: Computation of SIF outputs from inputs

Workflow

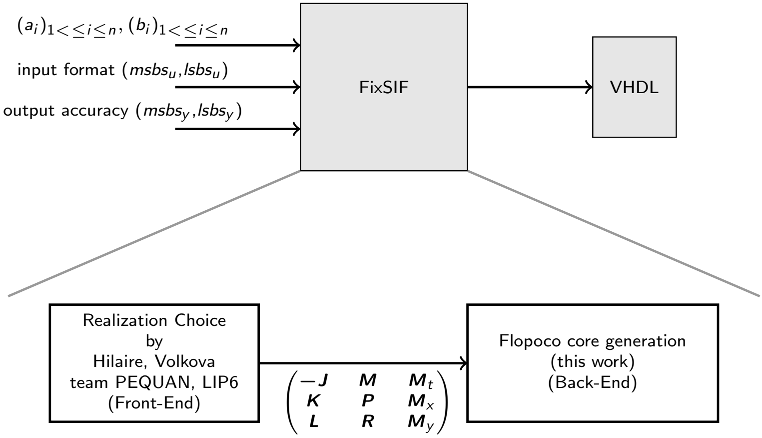


Figure: Workflow overview of tools usage

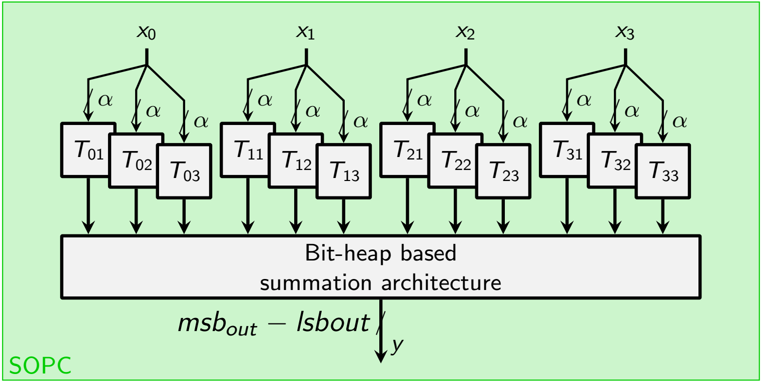
Table of Contents

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: based on the KCM architecture, split in 3 chunks

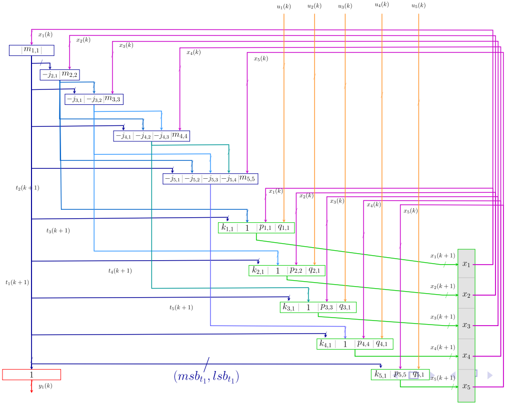
Architecture generation algorithm

```
computeMSBSLSBS([msbs, lsbs][][]) //get the matrix of msbs lsbs.
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

Example

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

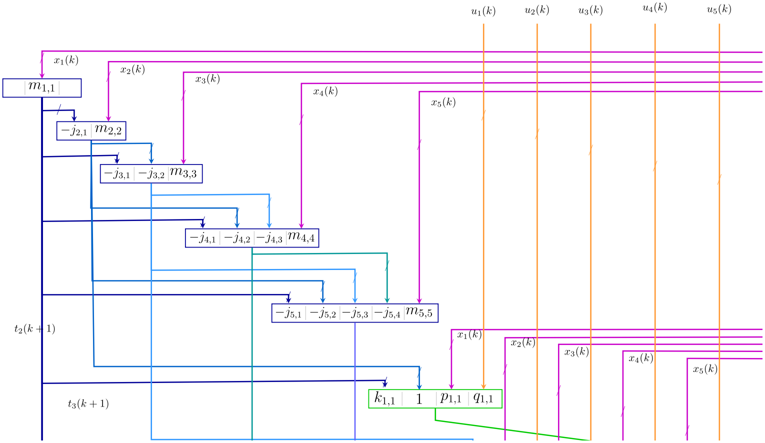


Example

Each line will be a SOPC

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

Example


$u_1(k)$
$u_2(k)$
$u_3(k)$
$u_4(k)$
$u_5(k)$
$x_1(k)$
$x_2(k)$
$x_3(k)$
$x_4(k)$
$x_5(k)$
$m_{1,1}$
$-j_{2,1}$ $m_{2,2}$
$-j_{3,1}$ $-j_{3,2}$ $m_{3,3}$
$-j_{4,1}$ $-j_{4,2}$ $-j_{4,3}$ $m_{4,4}$
$-j_{5,1}$ $-j_{5,2}$ $-j_{5,3}$ $-j_{5,4}$ $m_{5,5}$
$t_2(k+1)$
$x_1(k)$
$x_2(k)$
$x_3(k)$
$x_4(k)$
$x_5(k)$
$k_{1,1}$ 1 $p_{1,1}$ $q_{1,1}$
$t_3(k+1)$

Example

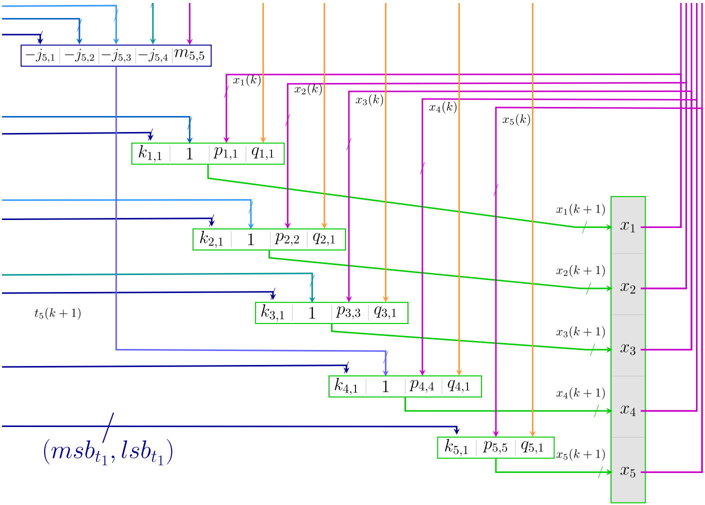
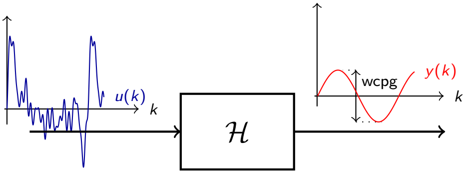

$-j_{5,1}$ $-j_{5,2}$ $-j_{5,3}$ $-j_{5,4}$ $m_{5,5}$
$x_1(k)$
$x_2(k)$
$x_3(k)$
$x_4(k)$
$x_5(k)$
$k_{1,1}$ 1 $p_{1,1}$ $q_{1,1}$
$k_{2,1}$ 1 $p_{2,2}$ $q_{2,1}$
$k_{3,1}$ 1 $p_{3,3}$ $q_{3,1}$
$k_{4,1}$ 1 $p_{4,4}$ $q_{4,1}$
$k_{5,1}$ $p_{5,5}$ $q_{5,1}$
$t_5(k+1)$
(msb_{t_1}, lsb_{t_1})
$x_1(k+1)$
$x_2(k+1)$
$x_3(k+1)$
$x_4(k+1)$
$x_5(k+1)$
x_1
x_2
x_3
x_4
x_5

Table of Contents

Computing the MSB



$$\|\mathcal{H}\|_{wcpg} = \sup_{u \neq 0} \frac{\|h * u\|_{l^\infty}}{\|u\|_{l^\infty}} \tag{5}$$

This computation is done by colleagues in LIP6.

Computing the LSB: Error definition

Let's define:

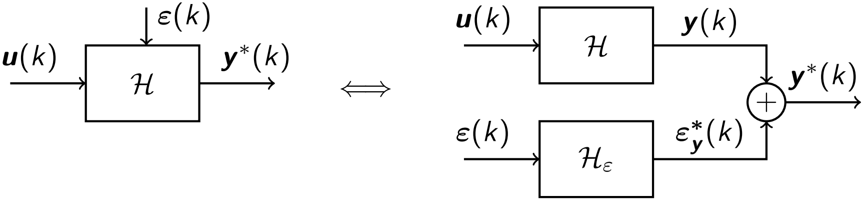
$$\boldsymbol{v'} = \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} \qquad (6)$$

Errors introduced by SOPCs:

$$\boldsymbol{\varepsilon}_{v'}(k) = \begin{pmatrix} \boldsymbol{\varepsilon}_t(k) \\ \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\varepsilon}_y(k) \end{pmatrix} = \begin{pmatrix} \varepsilon_{t_1}(k) \\ \varepsilon_{t_2}(k) \\ \vdots \\ \varepsilon_{t_{n_t}}(k) \\ \varepsilon_{x_1}(k) \\ \varepsilon_{x_2}(k) \\ \vdots \\ \varepsilon_{x_{n_x}}(k) \\ \varepsilon_{y_1}(k) \\ \varepsilon_{y_2}(k) \\ \vdots \\ \varepsilon_{y_{n_y}}(k) \end{pmatrix} \qquad (7)$$

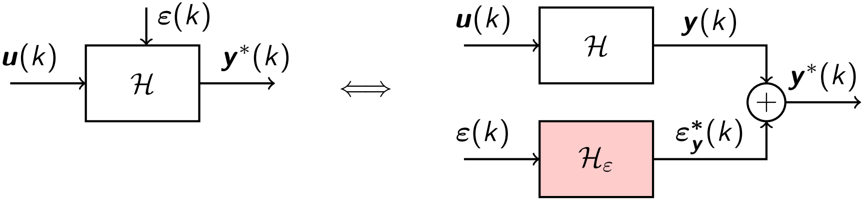
$\boldsymbol{\varepsilon}^*_{v'}(k)$ will represent the total error and is defined similarly to $\boldsymbol{\varepsilon}_{v'}(k)$

Computing the LSB: Point of view about error



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

Computing the LSB: Point of view about error



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**

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

end

for *int i = 0 ; i ≤ n_x; i++* **do**

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

end

for *int i = 0 ; i ≤ ny; i++* **do**

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

end

Algorithm 3: Computation of SIF outputs from inputs

Computing the LSB: current solution

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.
In the present case, a better solution is achievable at the end of this work.

Table of Contents

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

Any question?

Full definition of the SIF

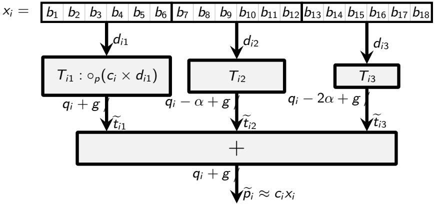
$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_{n_x} & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_{n_y} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix}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

With:

$$\begin{aligned} \boldsymbol{J} \in \mathbb{R}^{n_t \times n_t}, \boldsymbol{M} \in \mathbb{R}^{n_t \times n_x}, \boldsymbol{N} \in \mathbb{R}^{n_t \times n_u}, \\ \boldsymbol{K} \in \mathbb{R}^{n_x \times n_t}, \boldsymbol{P} \in \mathbb{R}^{n_x \times n_x}, \boldsymbol{Q} \in \mathbb{R}^{n_x \times n_u}, \\ \boldsymbol{L} \in \mathbb{R}^{n_y \times n_t}, \boldsymbol{R} \in \mathbb{R}^{n_y \times n_x}, \boldsymbol{S} \in \mathbb{R}^{n_y \times n_u}, \end{aligned} \tag{11}$$

$$\tag{12}$$

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:

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

Computation of the outputs:

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. Joldeș, 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.