

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

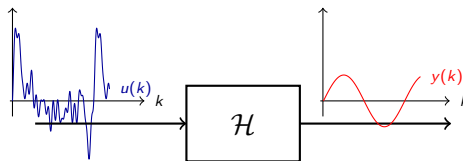
- 1 Signal processing and filters
 - Fundamentals, Purpose
 - FIR, IIR
 - State-Space representation
 - The SIF: a unified realization representation
- 2 Generating the architecture
 - The FloPoCo Framework: computing just right
 - Basic block: SOPC
 - Architecture generation
- 3 Computing the parameters
 - Computing the MSB
 - Computing the LSB
- 4 Future Work

Table of Contents

- 1 Signal processing and filters
 - Fundamentals, Purpose
 - FIR, IIR
 - State-Space representation
 - The SIF: a unified realization representation
- 2 Generating the architecture
 - The FloPoCo Framework: computing just right
 - Basic block: SOPC
 - Architecture generation
- 3 Computing the parameters
 - Computing the MSB
 - Computing the LSB
- 4 Future Work

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

- 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

The most simple LTI Filter: the FIR (Finite Impulse Response) filter

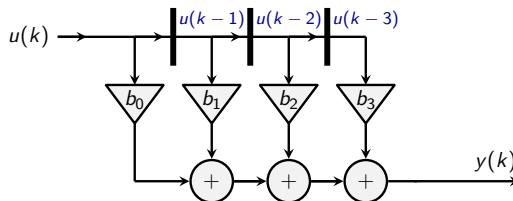
$$y(k) = \sum_{i=0}^n b_i u(k-i) \quad (1)$$

FIR, IIR

The most simple LTI Filter: the FIR (Finite Impulse Response) filter

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

Abstract architecture for the direct form realization of a FIR filter:

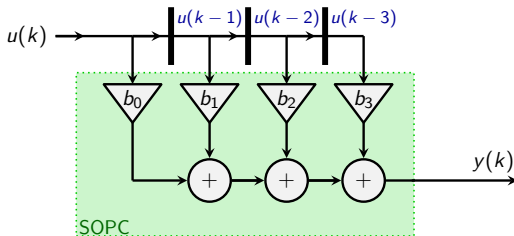


FIR, IIR

The most simple LTI Filter: the FIR (Finite Impulse Response) filter

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

Abstract architecture for the direct form realization of a FIR filter:



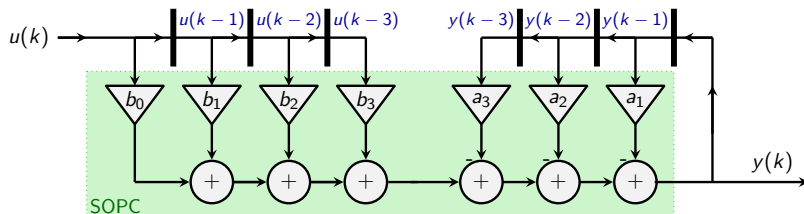
SOPC (Sum Of Products by Constants):
basic block of LTI filters implementation.

FIR, IIR

Full LTI complexity: the IIR (Infinite Impulse Response) filter:

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

Abstract architecture for the direct form realization of an IIR filter:



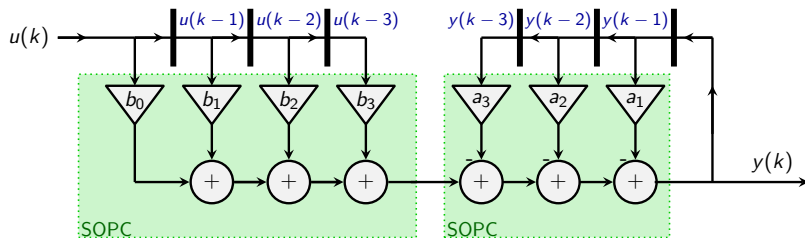
SOPC (Sum Of Products by Constants):
basic block of LTI filters implementation.

FIR, IIR

Full LTI complexity: the IIR (Infinite Impulse Response) filter:

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

Abstract architecture for the direct form realization of an IIR filter:



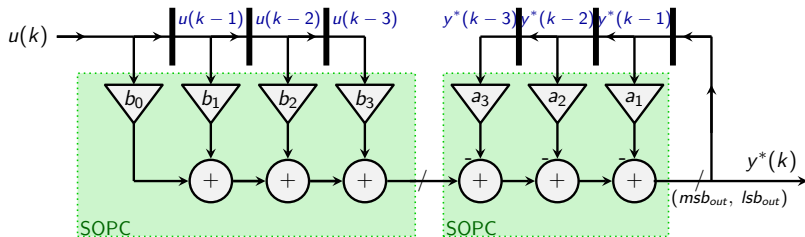
SOPC (Sum Of Products by Constants):
basic block of LTI filters implementation.

FIR, IIR

Full LTI complexity: the IIR (Infinite Impulse Response) filter:

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

Abstract architecture for the direct form realization of an IIR filter:



SOPC (Sum Of Products by Constants):
basic block of LTI filters implementation.

State-Space representation

The “ABCD” form

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

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

With:

$$\begin{aligned} \mathbf{A} &\in \mathbb{R}^{n_x \times n_x}, \quad \mathbf{B} \in \mathbb{R}^{n_x \times n_u}, \\ \mathbf{C} &\in \mathbb{R}^{n_y \times n_x}, \quad \mathbf{D} \in \mathbb{R}^{n_y \times n_u} \end{aligned}$$

Equivalent matrix formulation:

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

The SIF: a unified realization representation

Problems with ABCD:

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

The SIF (Specialized Implicit Form) generalizes the state-space.

Addition: $\mathbf{t}(k)$ describes explicit intermediate computations:

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

The SIF: a unified realization representation

The SIF as an algorithm

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

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

end

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

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

end

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

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

end

Algorithm 1: Computation of SIF outputs from inputs

A more common notation

A non-compact notation:

$$\begin{pmatrix} \mathbf{J} & \mathbf{0} & \mathbf{0} \\ -\mathbf{K} & \mathbf{I}_{n_x} & \mathbf{0} \\ -\mathbf{L} & \mathbf{0} & \mathbf{I}_{n_y} \end{pmatrix} \begin{pmatrix} \mathbf{t}(k+1) \\ \mathbf{x}(k+1) \\ \mathbf{y}(k) \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{M} & \mathbf{N} \\ \mathbf{0} & \mathbf{P} & \mathbf{Q} \\ \mathbf{0} & \mathbf{R} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{t}(k) \\ \mathbf{x}(k) \\ \mathbf{u}(k) \end{pmatrix} \quad (5)$$

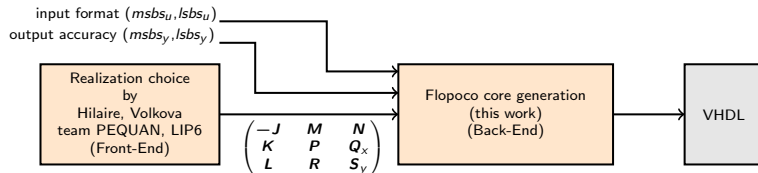
An easier way to communicate SIFs:

The Z matrix:

$$\mathbf{Z} = \begin{pmatrix} -\mathbf{J} & \mathbf{M} & \mathbf{N} \\ \mathbf{K} & \mathbf{P} & \mathbf{Q} \\ \mathbf{L} & \mathbf{R} & \mathbf{S} \end{pmatrix} \quad (6)$$

The SIF: a unified realization representation

Workflow



Workflow overview of tools usage

Table of Contents

- 1 Signal processing and filters
 - Fundamentals, Purpose
 - FIR, IIR
 - State-Space representation
 - The SIF: a unified realization representation
- 2 Generating the architecture
 - The FloPoCo Framework: computing just right
 - Basic block: SOPC
 - Architecture generation
- 3 Computing the parameters
 - Computing the MSB
 - Computing the LSB
- 4 Future Work

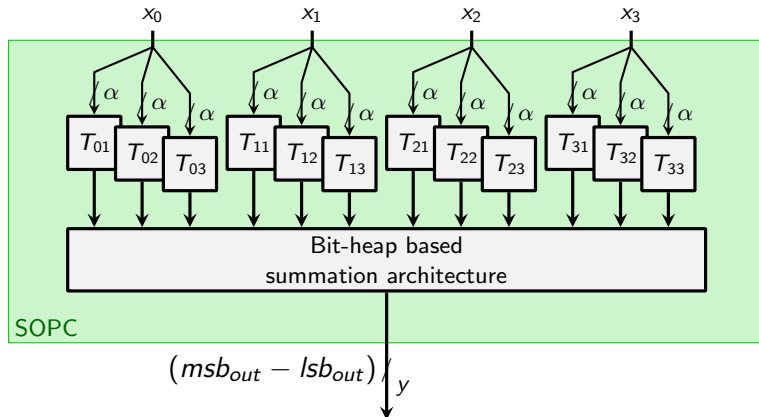
The FloPoCo Framework: computing just right

FloPoCo:

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

Basic block: SOPC

Basic block: SOPC



SOPC architecture

Architecture generation algorithm

Assuming all the MSBs and LSBs are known.

```

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

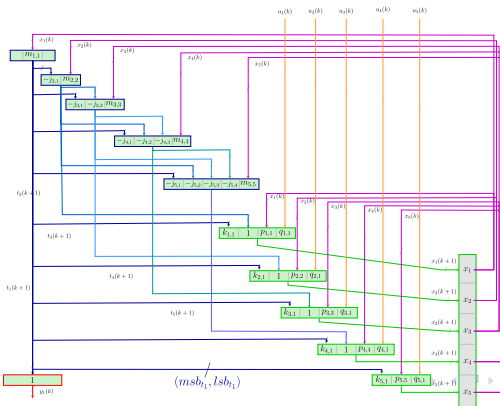
```

Algorithm 2: Architecture Generation Algorithm

Example

$$Z =$$

1	0	0	0	0	$m_{1,1}$	0	0	0	0	0
$p_{2,1}$	1	0	0	0	0	$m_{2,2}$	0	0	0	0
$p_{3,1}$	$p_{3,2}$	1	0	0	0	0	$m_{3,3}$	0	0	0
$p_{4,1}$	$p_{4,2}$	$p_{4,3}$	1	0	0	0	0	$m_{4,4}$	0	0
$p_{5,1}$	$p_{5,2}$	$p_{5,3}$	$p_{5,4}$	1	0	0	0	0	$m_{5,5}$	0
$k_{1,1}$	1	0	0	0	0	$p_{1,1}$	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



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

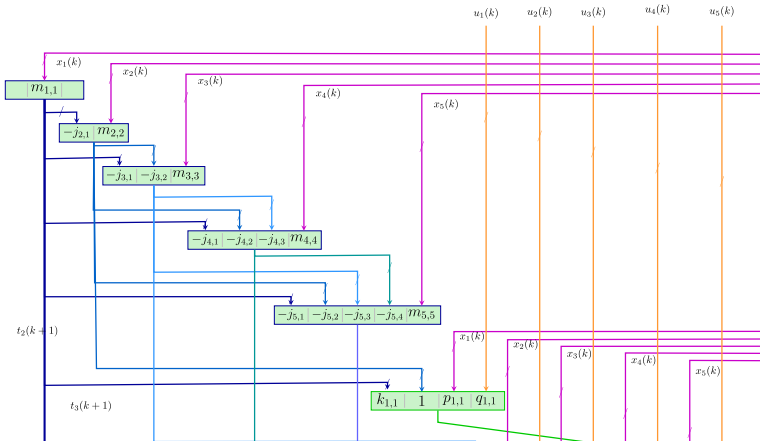
oooooooo

oooooooo

oooo

Architecture generation

Example



Example

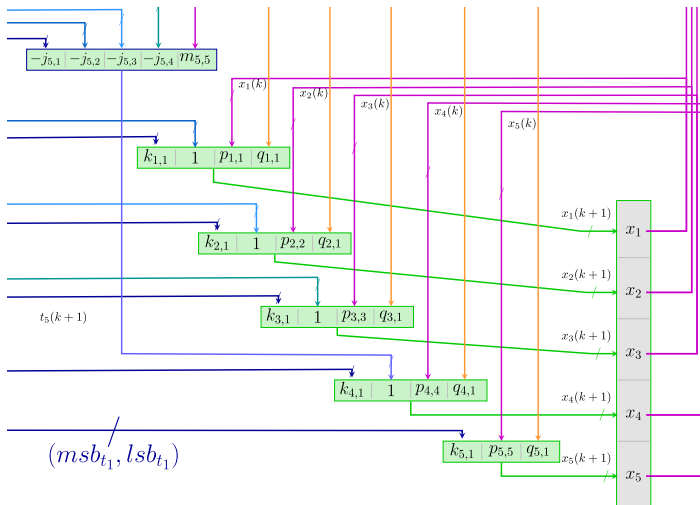
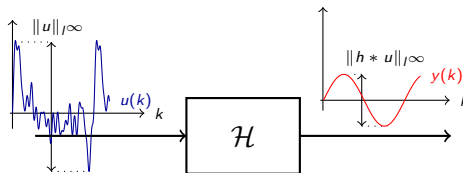


Table of Contents

- 1 Signal processing and filters
 - Fundamentals, Purpose
 - FIR, IIR
 - State-Space representation
 - The SIF: a unified realization representation
- 2 Generating the architecture
 - The FloPoCo Framework: computing just right
 - Basic block: SOPC
 - Architecture generation
- 3 Computing the parameters
 - Computing the MSB
 - Computing the LSB
- 4 Future Work

Computing the MSB

MSB should allow the full range of the signal.
Worst-Case Peak Gain (WCPG):



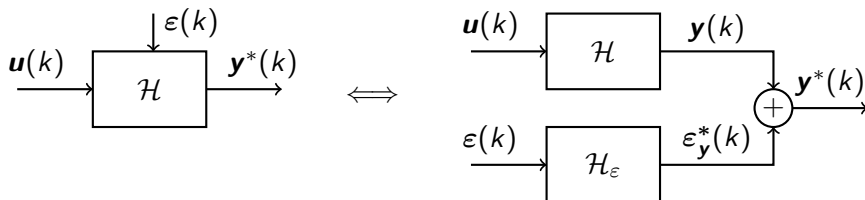
$$\|\mathcal{H}\|_{wcpg} = \sup_{u \neq 0} \frac{\|h * u\|_\infty}{\|u\|_\infty} \quad (7)$$

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)

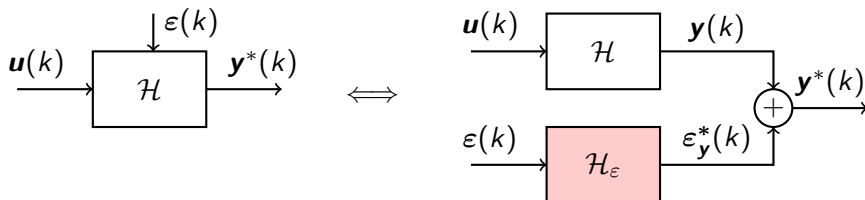


Error propagation with respect to the ideal filter

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)



Error propagation with respect to the ideal filter

Computing the LSB: Impact of errors

```

for int i = 0 ; i ≤ nt; i++ do
    |  $\mathbf{t}_i(k+1) \leftarrow$ 
    |  $-\sum_{j<i} \mathbf{J}_{ij} \mathbf{t}_j(k+1) + \sum_{j=1}^{n_x} \mathbf{M}_{ij} \mathbf{x}_j(k) + \sum_{j=1}^{n_u} \mathbf{N}_{ij} \mathbf{u}_j(k) + \epsilon_{t_i}(k)$ 
end
for int i = 0 ; i ≤ nx; i++ do
    |  $\mathbf{x}_i(k+1) \leftarrow \sum_{j=1}^{n_t} \mathbf{K}_{ij} \mathbf{t}_j(k+1) + \sum_{j=1}^{n_x} \mathbf{P}_{ij} \mathbf{x}_j(k) + \sum_{j=1}^{n_u} \mathbf{Q}_{ij} \mathbf{u}_j(k) + \epsilon_{x_i}(k)$ 
end
for int i = 0 ; i ≤ ny; i++ do
    |  $\mathbf{y}_i(k) \leftarrow \sum_{j=1}^{n_t} \mathbf{L}_{ij} \mathbf{t}_j(k+1) + \sum_{j=1}^{n_x} \mathbf{R}_{ij} \mathbf{x}_j(k) + \sum_{j=1}^{n_u} \mathbf{S}_{ij} \mathbf{u}_j(k) + \epsilon_{y_i}(k)$ 
end

```

Algorithm 3: Computation of SIF outputs from inputs

Computing the LSB: current solution

Let's define:

$$\mathbf{v}' = \begin{pmatrix} \mathbf{t}(k+1) \\ \mathbf{x}(k+1) \\ \mathbf{y}(k) \end{pmatrix} \quad (8)$$

Main constraint:

$$\varepsilon_{\mathbf{v}'} < 2^{-lsb_{\mathbf{v}'}} \quad (9)$$

Transformed into the following constraint:

$$|\langle \langle \mathcal{H}_\varepsilon \rangle \rangle| \cdot 2^{lsb_{\mathbf{v}'}+1} < 2^{-lsb_{y_i}} \quad (10)$$

Lopez gives a simple solution that matches the fixed size constraint.

The solution matching the hardware context (arbitrary precision) is work in progress.

Table of Contents

- 1 Signal processing and filters
 - Fundamentals, Purpose
 - FIR, IIR
 - State-Space representation
 - The SIF: a unified realization representation
- 2 Generating the architecture
 - The FloPoCo Framework: computing just right
 - Basic block: SOPC
 - Architecture generation
- 3 Computing the parameters
 - Computing the MSB
 - Computing the LSB
- 4 Future Work

Future Work

- Finish implementation of LSB computation (still waiting for the WCPG code)
- Removal of power of two: adapt KCM and SOPC FloPoCo cores
- Sub-filter detection: open question for either Front-End and Back-End
- Better interface, using matlab matrix syntax

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

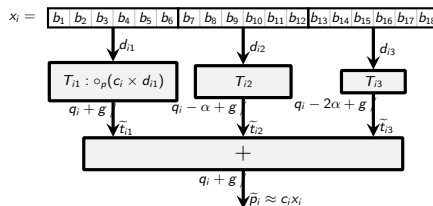
$$\begin{pmatrix} \mathbf{J} & \mathbf{0} & \mathbf{0} \\ -\mathbf{K} & \mathbf{I}_{n_x} & \mathbf{0} \\ -\mathbf{L} & \mathbf{0} & \mathbf{I}_{n_y} \end{pmatrix} \begin{pmatrix} \mathbf{t}(k+1) \\ \mathbf{x}(k+1) \\ \mathbf{y}(k) \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{M} & \mathbf{N} \\ \mathbf{0} & \mathbf{P} & \mathbf{Q} \\ \mathbf{0} & \mathbf{R} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{t}(k) \\ \mathbf{x}(k) \\ \mathbf{u}(k) \end{pmatrix} \quad (11)$$

With:

$$\begin{aligned} \mathbf{J} &\in \mathbb{R}^{n_t \times n_t}, \mathbf{M} \in \mathbb{R}^{n_t \times n_x}, \mathbf{N} \in \mathbb{R}^{n_t \times n_u}, \\ \mathbf{K} &\in \mathbb{R}^{n_x \times n_t}, \mathbf{P} \in \mathbb{R}^{n_x \times n_x}, \mathbf{Q} \in \mathbb{R}^{n_x \times n_u}, \end{aligned} \quad (12)$$

$$\mathbf{L} \in \mathbb{R}^{n_y \times n_t}, \mathbf{R} \in \mathbb{R}^{n_y \times n_x}, \mathbf{S} \in \mathbb{R}^{n_y \times n_u}, \quad (13)$$

KCM multiplier



The FixRealKCM method when x_i is split in 3 chunks

Mathematical definition of a filter \mathcal{H}

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

Linearity:

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

Time invariance:

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

Impulse response

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

Where δ_l is a Dirac impulsions centered in l :

$$\delta_l(k) = \begin{cases} 1 & \text{when } k = l \\ 0 & \text{else} \end{cases} \quad (14)$$

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

Computation of the outputs:

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

From:

$$\mathbf{Z} = \begin{pmatrix} -J & M & N \\ K & P & Q \\ L & R & S \end{pmatrix} \quad (15)$$

The ABCD form is deducible from the SIF:

$$\begin{aligned} \mathbf{A}_Z &= \mathbf{KJ}^{-1}\mathbf{M} + \mathbf{P}, & \mathbf{B}_Z &= \mathbf{KJ}^{-1}\mathbf{N} + \mathbf{Q}, \\ \mathbf{C}_Z &= \mathbf{LJ}^{-1}\mathbf{M} + \mathbf{R}, & \mathbf{D}_Z &= \mathbf{LJ}^{-1}\mathbf{N} + \mathbf{S}, \end{aligned} \quad (16)$$

with:

$$\begin{aligned} \mathbf{A}_Z &\in \mathbb{R}^{n_x \times n_x}, & \mathbf{B}_Z &\in \mathbb{R}^{n_x \times n_u}, \\ \mathbf{C}_Z &\in \mathbb{R}^{n_y \times n_x}, & \mathbf{D}_Z &\in \mathbb{R}^{n_y \times n_u}, \end{aligned} \quad (17)$$

Definition of the error filter:

$$\mathbf{Z}_\varepsilon = \begin{pmatrix} -\mathbf{J} & \mathbf{M} & \mathbf{M}_t \\ \mathbf{K} & \mathbf{P} & \mathbf{M}_x \\ \mathbf{L} & \mathbf{R} & \mathbf{M}_y \end{pmatrix} \quad (18)$$

with:

$$\mathbf{M}_t = (\mathbf{I}_{n_t} \quad \mathbf{0}_{n_t \times n_x} \quad \mathbf{0}_{n_t \times n_y}), \quad (19)$$

$$\mathbf{M}_x = (\mathbf{0}_{n_x \times n_t} \quad \mathbf{I}_{n_x} \quad \mathbf{0}_{n_x \times n_y}), \quad (20)$$

$$\mathbf{M}_y = (\mathbf{0}_{n_y \times n_t} \quad \mathbf{0}_{n_y \times n_x} \quad \mathbf{I}_{n_y}), \quad (21)$$

Interface specification:

$X \mid c$

$x_{1_1} x_{1_2} \dots x_{1_c}$

$x_{2_1} x_{2_2} \dots x_{2_c}$

.

.

.

$x_{l_1} x_{l_2} \dots x_{l_c}$

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 number of columns.

Computing the LSB: Error definition

Errors introduced by SOPCs:

$$\epsilon_{v'}(k) = \begin{pmatrix} \epsilon_t(k) \\ \epsilon_x(k) \\ \epsilon_y(k) \end{pmatrix} = \begin{pmatrix} \epsilon_{t_1}(k) \\ \epsilon_{t_2}(k) \\ \vdots \\ \epsilon_{t_{n_t}}(k) \\ \epsilon_{x_1}(k) \\ \epsilon_{x_2}(k) \\ \vdots \\ \epsilon_{x_{n_x}}(k) \\ \epsilon_{y_1}(k) \\ \epsilon_{y_2}(k) \\ \vdots \\ \epsilon_{y_{n_y}}(k) \end{pmatrix} \quad (22)$$

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

LSB computation I

$$|\langle\langle\mathcal{H}_\epsilon\rangle\rangle| \cdot 2^{lsb_{v'}+1} < 2^{-lsb_{y_i}} \quad (23)$$

Expanded line:

$$\sum_{j=1}^{n_t} |\langle\langle\mathcal{H}_\epsilon\rangle\rangle_{i,j}| \cdot 2^{lsb_{t_i}+1} + \sum_{j=n_t}^{n_t+n_x} |\langle\langle\mathcal{H}_\epsilon\rangle\rangle_{i,j}| \cdot 2^{lsb_{x_i}+1} + \sum_{j=n_t+n_x}^{n_t+n_x+n_u} |\langle\langle\mathcal{H}_\epsilon\rangle\rangle_{i,j}| \cdot 2^{lsb_{u_i}+1} < 2^{-lsb_{y_i}} \quad (24)$$

LSB computation II

Splitting error in $n' = nt + nx + nu$ equal chunks:

$$|\langle\langle\mathcal{H}_\varepsilon\rangle\rangle_{i,j}| \cdot 2^{lsb_{v'}+1} < \frac{2^{-lsb_{y_i}}}{n'} \quad (25)$$

\Leftrightarrow

$$2^{lsb_{t_j}+1} < \frac{2^{-lsb_{y_i}}}{|\langle\langle\mathcal{H}_\varepsilon\rangle\rangle_{i,t_j}| \cdot n'} \quad (26)$$

$$2^{lsb_{x_j}+1} < \frac{2^{-lsb_{y_i}}}{|\langle\langle\mathcal{H}_\varepsilon\rangle\rangle_{i,x_j}| \cdot n'} \quad (27)$$

Then:

$$lsb_{x_j} < lsb_{y_i} - \log(n' \cdot \|\langle\langle\mathcal{H}_\varepsilon\rangle\rangle\|_{i,x_j}) - 1 \quad (28)$$

$$lsb_{t_j} < lsb_{y_i} - \log(n' \cdot \|\langle\langle\mathcal{H}_\varepsilon\rangle\rangle\|_{i,t_j}) - 1 \quad (29)$$

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-infinity 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 sensitivity 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.