From controller/filter to code: the *optimal* implementation problem

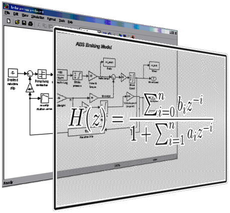
Thibault Hilaire (thibault.hilaire@lip6.fr)

Séminaire département SOC

1 Context and problematics

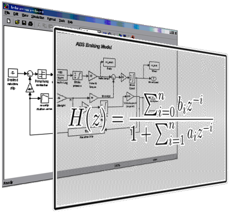
Outline

Context



Filters/Controllers
Algorithms

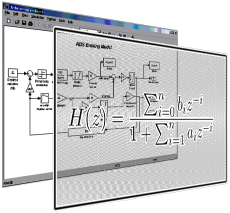
Context



Filters/Controllers
Algorithms

Targets

Context

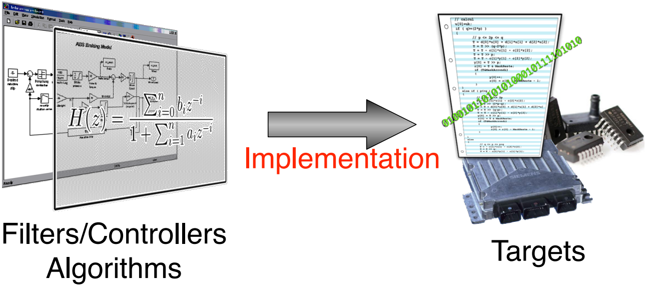


Filters/Controllers
Algorithms

Implementation

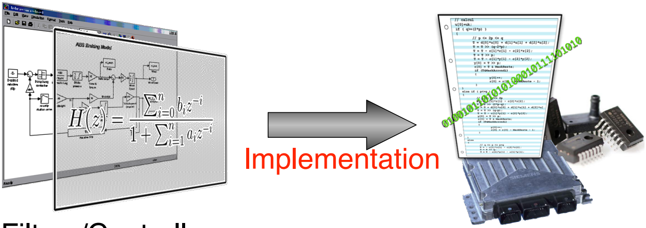
Targets

Context



- Finite precision implementation (fixed-point arithmetic)
- LTI systems
- hardware (FPGA, ASIC) or software (DSP, μC)

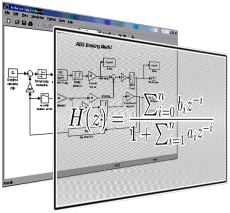
Context



- Finite precision implementation (fixed-point arithmetic)
- LTI systems
- hardware (FPGA, ASIC) or software (DSP, μC)

Methodology for the implementation of embedded algorithms (controllers/filters)

Context



Filters/Controllers
Algorithms

Implementation

Targets

- Finite precision implementation (fixed-point arithmetic)
- LTI systems
- hardware (FPGA, ASIC) or software (DSP, μC)

$\rightarrow$*implementation*: transformation of the mathematical object into finite precision operations to be performed on a specific target

Implementation under constraints

Embedded systems suffer from various constraints:

- Cost (short *Time to Market*, rapid renewal, etc...)
- Power consumption (energy-autonomous systems)
- Resources (Limited computing resources)

Implementation under constraints

Embedded systems suffer from various constraints:

- Cost (short *Time to Market*, rapid renewal, etc...)
- Power consumption (energy-autonomous systems)
- Resources (Limited computing resources: *no FPU ?*)

Implementation under constraints

Embedded systems suffer from various constraints:

- Cost (short *Time to Market*, rapid renewal, etc...)
- Power consumption (energy-autonomous systems)
- Resources (Limited computing resources: *no FPU ?*)

The implementation of a given algorithm is difficult:

- Deal with precision problems
  - Implementation ⇒ accuracy problems
  - Write fixed-point code/algorithm is not easy
- Area/power consumption constraints
  - multiple wordlength paradigm (ASIC and FPGA)
- Time-consuming work

Implementation under constraints

Embedded systems suffer from various constraints:

- Cost (short *Time to Market*, rapid renewal, etc...)
- Power consumption (energy-autonomous systems)
- Resources (Limited computing resources: *no FPU ?*)

The implementation of a given algorithm is difficult:

- Deal with precision problems
  - Implementation $\Rightarrow$ accuracy problems
  - Write fixed-point code/algorithm is not easy
- Area/power consumption constraints
  - multiple wordlength paradigm (ASIC and FPGA)
- Time-consuming work

We need tools to transform filters/controllers into code that deal with precision/performance/computational cost tradeoff

Objectives

The objectives of the presentation is to give an overview of:

- the interest *and* the difficulties of this problem
- the parts already (almost) done
- the various parts *in-progress*
- and to open the discussions with *your works!*

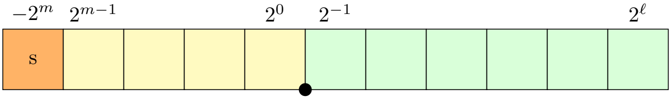
Fixed-point representation – 1

Fixed-point

- A real number x is *represented* by

$$X \cdot 2^{\ell} \qquad \begin{array}{ll} X & : \text{signed integer on } w \text{ bits (2's complement)} \\ \ell & : \text{fixed integer (implicit)} \end{array}$$

- The quantization step $q = 2^{\ell}$ is fixed, the dynamic is fixed (and often limited)



Fixed-point representation – 1

Fixed-point

- A real number x is *represented* by

$$X \cdot 2^{\ell} \qquad \begin{array}{ll} X & : \text{signed integer on } w \text{ bits (2's complement)} \\ \ell & : \text{fixed integer (implicit)} \end{array}$$

- The quantization step $q = 2^{\ell}$ is fixed, the dynamic is fixed (and often limited)

Example: the 1st bits of $\sqrt{2}$ are

$$01_{\Delta}0110101000001001111001 1\ldots$$

Fixed-point representation – 1

Fixed-point

- A real number x is *represented* by

$$X \cdot 2^{\ell} \qquad \begin{array}{ll} X & : \text{signed integer on } w \text{ bits (2's complement)} \\ \ell & : \text{fixed integer (implicit)} \end{array}$$

- The quantization step $q = 2^{\ell}$ is fixed, the dynamic is fixed (and often limited)

Example: the 1st bits of $\sqrt{2}$ are

$$\color{red}{01_{\triangle}011010}\color{black}{10000010011110011\ldots}$$

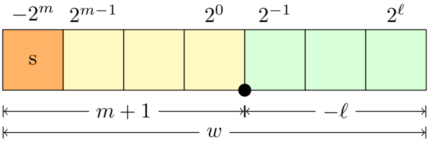
So, with $w = 8$, we choose $\ell = -6$

And then $\sqrt{2}$ is approached by 91.2^{-6}, and coded on 8 bits by the integer 91

Fixed-point representation – 2

A real number x is *represented* by a fixed-point number $\tilde{x}$:

$$\tilde{x} = \underbrace{-2^m x_\alpha}_{\text{sign bit}} + \sum_{i=\ell}^{m-1} x_i 2^i, \quad x_i \in \mathbb{B} \triangleq \{0,1\}$$



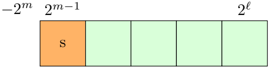
- m is the *most significant bit*, l is the *least significant bit*
- 2's complement or unsigned arithmetic
- x is approached with $w = m - \ell + 1$ bits

Fixed-point representation – 3

- Often, $m > 0$ and $\ell < 0$

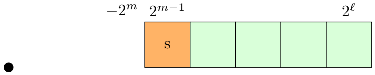
Fixed-point representation – 3

- Often, $m > 0$ and $\ell < 0$
- But m could be < 0 (no integer part, $x < 1$)
-

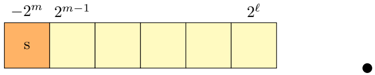


Fixed-point representation – 3

- Often, $m > 0$ and $\ell < 0$
- But m could be < 0 (no integer part, $x < 1$)



- ℓ could be > 0 (no fractional, quantization step $q > 1$)



Fixed-point arithmetic – 2

Depending on wordlength operators, certain rules can be set for the error-free fixed-point operations:

Operator	s_z	w_z	m_z
$+$	$s_x \vert s_y$	$m_z + \max(w_x - m_x, w_y - m_y)$	$\max(m_x + !s_x \& s_y, m_y + !s_y \& s_x) + 1$
-	$True$	$m_z + max(w_x - m_x, w_y - m_y)$	$\max(m_x + !s_x \& s_y, m_y + !s_y \& s_x) + 1$
$\times$	$s_x \vert s_y$	$w_x + w_y$	$m_x + m_y + 1$

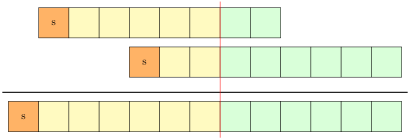
- The wordlength for the operations is defined by the wordlength of the operands
- One bit is added for the addition/substraction to prevent the overflow

Fixed-point arithmetic – 2

Depending on wordlength operators, certain rules can be set for the error-free fixed-point operations:

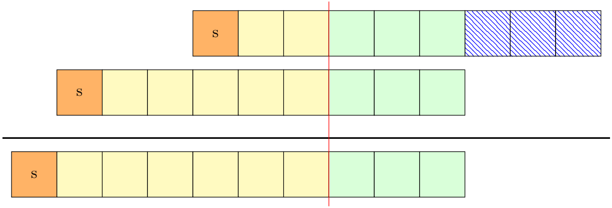
Operator	s_z	w_z	m_z
$+$	$s_x \| s_y$	$m_z + \max(w_x - m_x, w_y - m_y)$	$\max(m_x + !s_x \& s_y, m_y + !s_y \& s_x) + 1$
-	*True*	$m_z + max(w_x - m_x, w_y - m_y)$	$\max(m_x + !s_x \& s_y, m_y + !s_y \& s_x) + 1$
$\times$	$s_x \| s_y$	$w_x + w_y$	$m_x + m_y + 1$

- The wordlength for the operations is defined by the wordlength of the operands
- One bit is added for the addition/substraction to prevent the overflow
- To perform the operations on the associated *integers*, the binary point positions must be aligned



Fixed-point arithmetic – 2

But in certain cases, it is necessary to perform the operations on lower wordlength operators



Obviously, the operations cannot be exact anymore

Finite precision degradations

The implementation $\Rightarrow$ degradations (performances, characteristics, etc.)

Finite precision degradations

The implementation $\Rightarrow$ degradations (performances, characteristics, etc.)

Origins of the degradations

Two sources of deteriorations are considered:

- Roundoff errors into the computations
  $\rightarrow$ *roundoff noise*

Finite precision degradations

The implementation $\Rightarrow$ degradations (performances, characteristics, etc.)

Origins of the degradations

Two sources of deteriorations are considered:

- Roundoff errors into the computations
  $\rightarrow$ *roundoff noise*
- Quantization of the coefficients
  $\rightarrow$ *parametric errors*

Finite precision degradations

The implementation $\Rightarrow$ degradations (performances, characteristics, etc.)

Origins of the degradations

Two sources of deteriorations are considered:

- Roundoff errors into the computations
  $\rightarrow$ *roundoff noise*
- Quantization of the coefficients
  $\rightarrow$ *parametric errors*

The degradations depends on

- the algorithmic relationship used to compute output(s) from input(s)
- the computations themselves (wordlengths, roundoff mode, binary point position, additions order, etc.)

Linear filters

LTI: Linear system with known and constant coefficients

- FIR (Finite Impulse Response) :

$$H(z) = \sum_{i=0}^{n} b_i z^{-i}, \quad \Rightarrow y(k) = \sum_{i=0}^{n} b_i u(k-i)$$

- IIR (Infinite Impulse Response) :

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}, \quad \Rightarrow y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$$

▸ Filters and signal processing

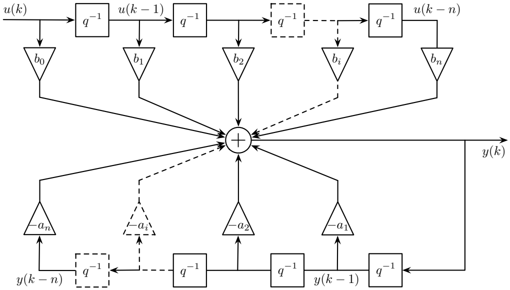
Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$

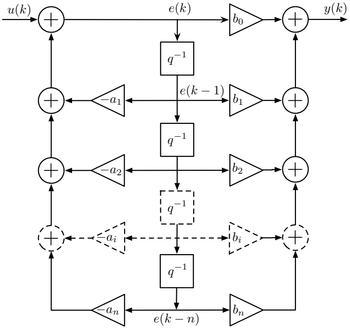
$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$$



Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)
- Direct Form II (transposed or not) ($2n+1$ coefs)



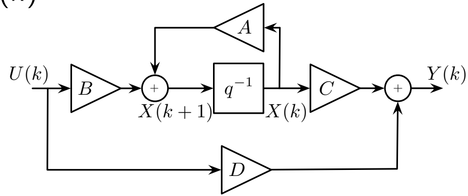
Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)
- Direct Form II (transposed or not) ($2n+1$ coefs)
- State-space (depend on the basis) ($(n+1)^2$ coefs)

$$\left\{\begin{array}{rcl} X(k+1) & = & AX(k)+Bu(k) \\ y(k) & = & CX(k)+Du(k) \end{array}\right.$$

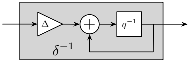
$$X(k) \rightarrow T.X(k)$$

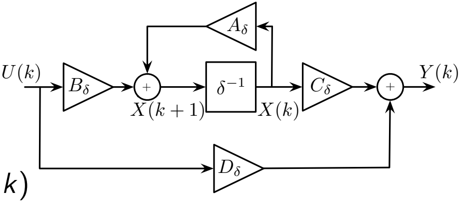


Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n + 1$ coefs)
- Direct Form II (transposed or not) ($2n + 1$ coefs)
- State-space (depend on the basis) ($(n+1)^2$ coefs)
- δ-operator $\qquad \delta \triangleq \frac{q-1}{\Delta}$



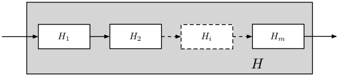


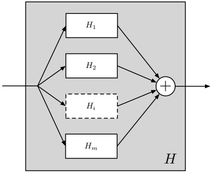
$$\begin{cases} \delta[X(k)] & = & A_\delta X(k) + B_\delta u(k) \\ y(k) & = & C_\delta X(k) + D_\delta u(k) \end{cases}$$

Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)
- Direct Form II (transposed or not) ($2n+1$ coefs)
- State-space (depend on the basis) ($(n+1)^2$ coefs)
- δ-operator $\delta \triangleq \frac{q-1}{\Delta}$
- Cascad and/or parallel decompositions

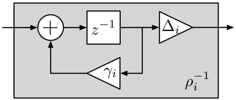


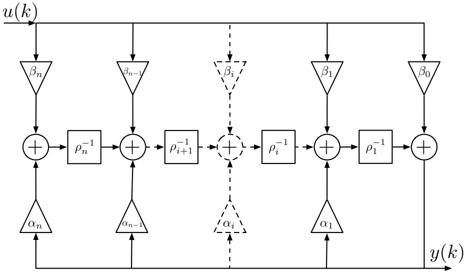


Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)
- Direct Form II (transposed or not) ($2n+1$ coefs)
- State-space (depend on the basis) ($(n+1)^2$ coefs)
- δ-operator $\qquad \delta \triangleq \frac{q-1}{\Delta}$
- Cascad and/or parallel decompositions
- ρDirect Form II transposed ($5n+1$ ou $4n+1$ coefs) ▸ ρDFIIt





Equivalent realizations

For a given LTI controller, it exist various equivalent realizations

- Direct Form I ($2n+1$ coefs)
- Direct Form II (transposed or not) ($2n+1$ coefs)
- State-space (depend on the basis) ($(n+1)^2$ coefs)
- δ-operator $\delta \triangleq \frac{q-1}{\Delta}$
- Cascad and/or parallel decompositions
- ρDirect Form II transposed ($5n+1$ ou $4n+1$ coefs) ρDFIIt
- lattice, ρ-modale, LGC et LCW-structures, sparse state-space, etc.

Each of these *realizations* uses different coefficients.
They are equivalent in infinite precision, but no more in finite precision. The finite precision degradation (parametric errors and roundoff noises) depends on the realization.

Comparison with polynomials

It is interesting to *compare* filter realizations and polynomial evaluation:

- The shift operator $\rightarrow\boxed{z^{-1}}\rightarrow$
- The multiplication by x

Comparison with polynomials

It is interesting to *compare* filter realizations and polynomial evaluation:

- The shift operator $\to \boxed{z^{-1}} \to$
- Direct forms

- The multiplication by x
- Horner's method

Comparison with polynomials

It is interesting to *compare* filter realizations and polynomial evaluation:

- The shift operator $\to \boxed{z^{-1}} \to$
- Direct forms
- ρ-operator

$$\varrho_i(z) = \prod_{j=i+1}^{n} \frac{z - \gamma_j}{\Delta_j}$$

- The multiplication by x
- Horner's method
- Newton's basis

$$P_i = \Pi_{k=1}^{i}(X - x_k)$$

Comparison with polynomials

It is interesting to *compare* filter realizations and polynomial evaluation:

- The shift operator $\rightarrow\boxed{z^{-1}}\rightarrow$
- Direct forms
- ρ-operator

$$\varrho_i(z) = \prod_{j=i+1}^{n} \frac{z-\gamma_j}{\Delta_j}$$

- ρDFIIt

- The multiplication by x
- Horner's method
- Newton's basis

$$P_i = \Pi_{k=1}^{i}(X - x_k)$$

- Horner's method with Newton's basis

HW or SW implementation

In addition to the choice of the realization, a lot of other options:

- Fixed-point evaluation schemes
  - order of the operations
  - fixed-point alignments
  - quantization modes, etc.

HW or SW implementation

In addition to the choice of the realization, a lot of other options:

- Fixed-point evaluation schemes
  - order of the operations
  - fixed-point alignments
  - quantization modes, etc.
- Software implementation (μC, DSP, etc.):
  - Parallelization
  - Ressource allocations

HW or SW implementation

In addition to the choice of the realization, a lot of other options:

- Fixed-point evaluation schemes
  - order of the operations
  - fixed-point alignments
  - quantization modes, etc.
- Software implementation (μC, DSP, etc.):
  - Parallelization
  - Ressource allocations
- Hardware implementation (FPGA, ASIC)
  - Multiple word-length paradigm
  - Operators optimizations (multiplications by constant, etc.)
  - Hardware-oriented arithmetic
  - Operators regroupment
  - ...

HW or SW implementation

In addition to the choice of the realization, a lot of other options:

- Fixed-point evaluation schemes
  - order of the operations
  - fixed-point alignments
  - quantization modes, etc.
- Software implementation (μC, DSP, etc.):
  - Parallelization
  - Ressource allocations
- Hardware implementation (FPGA, ASIC)
  - Multiple word-length paradigm
  - Operators optimizations (multiplications by constant, etc.)
  - Hardware-oriented arithmetic
  - Operators regroupment
  - ...

Different levels of optimizations !

2 Our approach

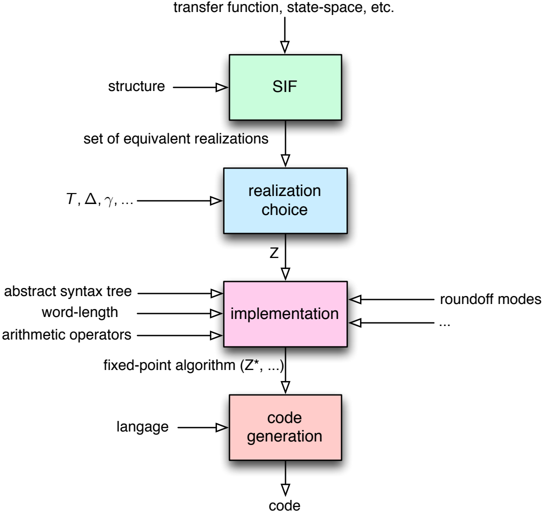
Outline

Objectives

Given a controller and a target, find the *optimal* implementation.

- Model all the equivalent realizations
  - SIF (*Specialized Implicit Form*)
- Model the fixed-point algorithms
  - depends on evaluation schemes for sum-of-products
- Model the hardware resources
  - computational units, etc.
- Evaluate the degradation
- Find one/some *optimal* realizations
  Tradeoff between:
  - Performance / Accuracy (multi-criteria)
  - Computational cost
  - Area, power consumption
  - Computation time, reusability, off-line efforts, etc.

From controller to code



A Unifying Framework

We have shown that it was possible to describe all these realizations in a *descriptor framework* called Specialized Implicit Form (SIF).

- *Macroscopic* description of the computations
- More general than state-space
- Explicit all the operations and their linking

$$\begin{pmatrix} J & 0 & 0 \\ -K & I & 0 \\ -L & 0 & I \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

A Unifying Framework

We have shown that it was possible to describe all these realizations in a *descriptor framework* called Specialized Implicit Form (SIF).

- *Macroscopic* description of the computations
- More general than state-space
- Explicit all the operations and their linking

$$\begin{pmatrix} J & 0 & 0 \\ -K & I & 0 \\ -L & 0 & I \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

$\rightarrow$ it is able to represent all the graph flows with delay, additions and constant multiplications

▸ SIF

A Unifying Framework

We have shown that it was possible to describe all these realizations in a *descriptor framework* called Specialized Implicit Form (SIF).

- *Macroscopic* description of the computations
- More general than state-space
- Explicit all the operations and their linking

$$\begin{pmatrix} J & 0 & 0 \\ -K & I & 0 \\ -L & 0 & I \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

We denote Z the matrix containing all the coefficients (and the structure of the realization)

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

▸ SIF

Different criteria

We have three kinds of criteria to evaluate the finite-precision degradation:

- Performance:
  - transfer function error $\left\| H - H^{\dagger} \right\|$
  - poles or zeros errors $\left\| |\lambda| - |\lambda^{\dagger}| \right\|$
  - weighted errors

Different criteria

We have three kinds of criteria to evaluate the finite-precision degradation:

- Performance:
  - transfer function error $\left\| H - H^{\dagger} \right\|$
  - poles or zeros errors $\left\| |\lambda| - |\lambda^{\dagger}| \right\|$
  - weighted errors
- Accuracy:
  - Signal to Quantization Noise Ratio

Different criteria

We have three kinds of criteria to evaluate the finite-precision degradation:

- Performance:
  - transfer function error $\left\|H - H^\dagger\right\|$
  - poles or zeros errors $\left\||\lambda| - |\lambda^\dagger|\right\|$
  - weighted errors
- Accuracy:
  - Signal to Quantization Noise Ratio
- Computational cost:
  - number of coefficients, number of operations
  - area, power consumption (FPGA, ASIC)
  - WCET
  - complexity, etc.

Different criteria

We have three kinds of criteria to evaluate the finite-precision degradation:

- Performance:
  - transfer function error $\left\|H - H^\dagger\right\|$
  - poles or zeros errors $\left\||\lambda| - |\lambda^\dagger|\right\|$
  - weighted errors
- Accuracy:
  - Signal to Quantization Noise Ratio
- Computational cost:
  - number of coefficients, number of operations
  - area, power consumption (FPGA, ASIC)
  - WCET
  - complexity, etc.

$\rightarrow$ These criteria require to know the *exact* implementation.

Different criteria

We have three kinds of criteria to evaluate the finite-precision degradation:

- Performance:
    - transfer function error $\left\|H - H^\dagger\right\|$
    - poles or zeros errors $\left\||\lambda| - |\lambda^\dagger|\right\|$
    - weighted errors
- Accuracy:
    - Signal to Quantization Noise Ratio
- Computational cost:
    - number of coefficients, number of operations
    - area, power consumption (FPGA, ASIC)
    - WCET
    - complexity, etc.

$\rightarrow$ These criteria require to know the *exact* implementation.
So we extend them in a priori criteria (that do not depend on implementation) and a posteriori criteria.

a priori criteria – 1

a priori measures

These criteria come from *control*:

a priori measures

- sensitivity with respect to the coefficients Z
  - transfer function sensitivity : $\frac{\partial H}{\partial Z}$

a priori criteria – 1

a priori measures

These criteria come from *control*:

a priori measures

- sensitivity with respect to the coefficients Z
  - transfer function sensitivity : $\frac{\partial H}{\partial Z}$
  - poles or zeros sensitivity : $\frac{\partial |\lambda_k|}{\partial Z}$

a priori criteria – 1

a priori measures

These criteria come from *control*:

a priori measures

- sensitivity with respect to the coefficients Z
  - transfer function sensitivity : $\frac{\partial H}{\partial Z}$
  - poles or zeros sensitivity : $\frac{\partial |\lambda_k|}{\partial Z}$
- Roundoff Noise Gain

  we suppose here that each quantization error leads to the addition of a white independent uniform noise

  $\rightarrow$ Gain from these noises to the output

a priori criteria – 2

a priori measures

The idea is to approximate how much the transfer function is modified by the coefficients' quantization.
Denote

$$(\delta_Z)_{i,j} = \begin{cases} 0 & \text{if } Z_{i,j} \text{ could be } \textit{exactly} \text{ implemented} \in \{2^p | p \in \mathbb{Z}\} \\ 1 & \text{else} \end{cases}$$

Then, the transfer function error is approached by the L_2-sensitivity measure

L_2-sensitivity measure

$$M_{L_2} = \left\| \frac{\partial H}{\partial Z} \times \delta_Z \right\|_2^2$$

a priori criteria – 3

a priori measures

In a same way, the pole error is approached by a pole-sensitivity measured.

pole-sensitivity

$$\Psi \triangleq \sum_{k=1}^{n} \omega_k \left\| \frac{\partial |\lambda_k|}{\partial Z} \times \delta_Z \right\|_F^2$$

where λ_k are the poles, and ω_k some wieghting parameters (usually $\omega_k = \frac{1}{1-|\lambda_k|}$)

a priori criteria – 3

a priori measures

In a same way, the pole error is approached by a pole-sensitivity measured.

pole-sensitivity

$$\Psi \triangleq \sum_{k=1}^{n} \omega_k \left\| \frac{\partial |\lambda_k|}{\partial Z} \times \delta_Z \right\|_F^2$$

where λ_k are the poles, and ω_k some wieghting parameters (usually $\omega_k = \frac{1}{1-|\lambda_k|}$)

$\rightarrow$In closed-loop, Ψ is related to the distance to instability.

a priori
measures

a priori criteria – 3

In a same way, the pole error is approached by a pole-sensitivity measured.

pole-sensitivity

$$\Psi \triangleq \sum_{k=1}^{n} \omega_k \left\| \frac{\partial |\lambda_k|}{\partial Z} \times \delta_Z \right\|_F^2$$

where λ_k are the poles, and ω_k some wieghting parameters (usually $\omega_k = \frac{1}{1-|\lambda_k|}$)

→In closed-loop, Ψ is related to the distance to instability.

These measures have been developed for the SIF:

- MIMO case
- open and closed-loop cases
- analytical expressions

And extended with more accurate fixed-point considerations $\sigma^2_{\delta H}$

a posteriori measure

a posteriori measures

The main measure based on exact implementation is the Signal to Quantization Noise Ratio ▸ SQNR:

a posteriori measure

It measures the noise on the output produced by the implementation

- bit-accurate measure
- depends on HW/SW
  - wordlength
  - evaluation scheme (order, roundoff mode, etc.)
- an optimization on the word-length can be done

a posteriori measure

a posteriori measures

The main measure based on exact implementation is the Signal to Quantization Noise Ratio SQNR:

a posteriori measure

It measures the noise on the output produced by the implementation

- bit-accurate measure
- depends on HW/SW
  - wordlength
  - evaluation scheme (order, roundoff mode, etc.)
- an optimization on the word-length can be done

A similar approach can be used for interval-based output errors. Output error

→ For that purpose, we need to generate bit-accurate fixed-point algorithms

Sum-of-products evaluation scheme – 1

fixed-point evaluation schemes

The only operations needed are sum-of-products:

$$S = \sum_{i=1}^{n} c_i \cdot x_i$$

where c_i are known constants and x_i variables (inputs, state or intermediate variables).

We have developed technics to transform real sum-of-products into fixed-point algorithms:

- we consider the $\Pi_{i=1}^{n}(2i-1)$ possible ordered-sum-of-products (assuming the fixed-point addition is commutative but not associative)
- given the various word-lengths (operators, constants), we propagate fixed-point rules among the abstract syntax tree
- we consider various scheme (RBM/RAM, no-shifts, etc.)

Sum-of-products evaluation scheme – 2

$$S = 1.524 \cdot X_0 + 4.89765 \cdot X_1 - 42.3246 \cdot X_2 + 58.35498 \cdot X_3 + 2.857 \cdot X_4 - 49.3246 \cdot X_5 + 24.3468 \cdot X_6$$

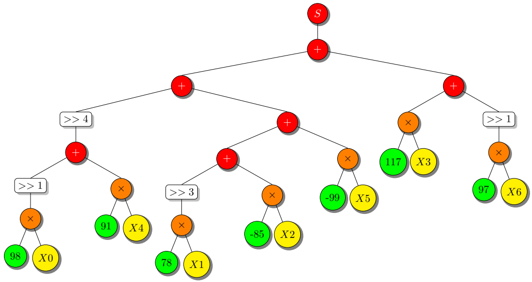
- 8-bit coefficients, 16-bit additions
- Q6.2 for x_i and s

Sum-of-products evaluation scheme – 2

fixed-point evaluation schemes

$$S = 1.524 \cdot X_0 + 4.89765 \cdot X_1 - 42.3246 \cdot X_2 + 58.35498 \cdot X_3 + 2.857 \cdot X_4 - 49.3246 \cdot X_5 + 24.3468 \cdot X_6$$

- 8-bit coefficients, 16-bit additions
- Q6.2 for x_i and s



Roundoff After Multiplication

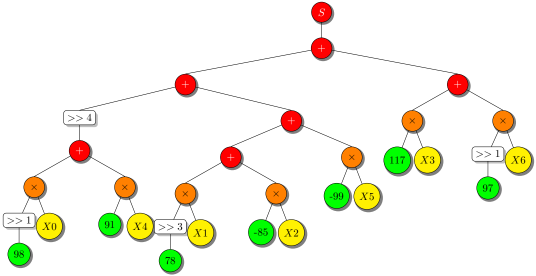
→The roundoff noise is then deduced

Sum-of-products evaluation scheme – 2

fixed-point evaluation schemes

$$S = 1.524{\cdot}X_0 + 4.89765{\cdot}X_1 - 42.3246{\cdot}X_2 + 58.35498{\cdot}X_3 + 2.857{\cdot}X_4 - 49.3246{\cdot}X_5 + 24.3468{\cdot}X_6$$

- 8-bit coefficients, 16-bit additions
- Q6.2 for x_i and s



Roundoff Before Multiplication

→The roundoff noise is then deduced

State-space example

$$\left\{\begin{array}{rcl} X(k+1) & = & \begin{pmatrix} 0.58399 & -0.42019 \\ 0.42019 & 0.1638 \end{pmatrix} X(k) + \begin{pmatrix} 0.64635 \\ -0.23982 \end{pmatrix} \\ Y(k) & = & \begin{pmatrix} 0.64635 & 0.23982 \end{pmatrix} X(k) + 0.13111 U(k) \end{array}\right.$$

State-space example

$$\left\{\begin{array}{rcl} X(k+1) & = & \begin{pmatrix} 0.58399 & -0.42019 \\ 0.42019 & 0.1638 \end{pmatrix} X(k) + \begin{pmatrix} 0.64635 \\ -0.23982 \end{pmatrix} \\ Y(k) & = & \begin{pmatrix} 0.64635 & 0.23982 \end{pmatrix} X(k) + 0.13111\, U(k) \end{array}\right.$$

16-bit input, output, states
16-bit coefficients
32-bit accumulators
$\overset{\max}{U} = 10$

State-space example

$$\begin{cases} X(k+1) &= \begin{pmatrix} 0.58399 & -0.42019 \\ 0.42019 & 0.1638 \end{pmatrix} X(k) + \begin{pmatrix} 0.64635 \\ -0.23982 \end{pmatrix} U(k) \\ Y(k) &= \begin{pmatrix} 0.64635 & 0.23982 \end{pmatrix} X(k) + 0.13111 U(k) \end{cases}$$

16-bit input, output, states
16-bit coefficients
32-bit accumulators

$\overset{\max}{U} = 10 \quad \Rightarrow \quad \gamma_U = 11$

$$\gamma_{ADD} = \begin{pmatrix} 26 \\ 27 \\ 26 \end{pmatrix}$$

$$\gamma_Z = \begin{pmatrix} 15 & 16 & 15 \\ 16 & 17 & 17 \\ 15 & 17 & 17 \end{pmatrix}$$

$$\gamma_X = \begin{pmatrix} 11 \\ 11 \end{pmatrix}$$

Roundoff After Multiplication

Intermediate variables

$Acc \leftarrow (xn(1) * 19136);$
$Acc \leftarrow Acc + (xn(2) * -27537) >> 1;$
$Acc \leftarrow Acc + (u * 21179);$
$xnp(1) \leftarrow Acc >> 15;$
$Acc \leftarrow (xn(1) * 27537);$
$Acc \leftarrow Acc + (xn(2) * 21470) >> 1;$
$Acc \leftarrow Acc + (u * -31433) >> 1;$
$xnp(2) \leftarrow Acc >> 16;$

Outputs

$Acc \leftarrow (xn(1) * 21179);$
$Acc \leftarrow Acc + (xn(2) * 31433) >> 2;$
$Acc \leftarrow Acc + (u * 17184) >> 2;$
$y \leftarrow Acc >> 15;$

Permutations

$xn \leftarrow xnp;$

Optimal realization

realization optimization

A 1st possible optimization concerns the realization

Optimal realization

realization optimization

A 1st possible optimization concerns the realization

Let H be a given controller, we denote $\mathscr{R}_H$ the set of equivalent realizations (with H as transfer function).
Let $\mathcal{J}$ be an *a priori* criterion (sensibility, RNG, $\sigma^2_{\Delta H}$, ...).

The optimal realization problem

The optimal realization problem consists in finding a realization that minimizes $\mathcal{J}$

$$\mathcal{R}^{\text{opt}} = \underset{\mathcal{R}\in\mathscr{R}_H}{\arg\min}\ \mathcal{J}(\mathcal{R})$$

Optimal realization

realization optimization

A 1st possible optimization concerns the realization

Let H be a given controller, we denote $\mathscr{R}_H$ the set of equivalent realizations (with H as transfer function).
Let $\mathcal{J}$ be an *a priori* criterion (sensibility, RNG, $\sigma^2_{\Delta H}$, ...).

The optimal realization problem

The optimal realization problem consists in finding a realization that minimizes $\mathcal{J}$

$$\mathcal{R}^{\text{opt}} = \underset{\mathcal{R} \in \mathscr{R}_H}{\arg\min}\ \mathcal{J}(\mathcal{R})$$

$\mathscr{R}_H$ is too large, so pratically we only considered structured realizations (state-space, ρ-forms, etc.)

Example – 1

We consider the following transfer function:

$$H : z \mapsto \frac{38252z^3 - 101878z^2 + 91135z - 27230}{z^4 - 2.3166z^3 + 2.1662z^2 - 0.96455z + 0.17565}$$

- Closed-loop L_2-sensitivity $\bar{M}_{L_2}$
- Closed-loop pole-sensitivity : $\bar{\Psi}$
- Roundoff Noise Gain (RNG) : $\bar{G}$

Example – 2

	$\bar{M}_{L_2}^W$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e+7$	$3.3562e+7$	$1.186e+6$		$7 + 8\times$

Example – 2

	$\bar{M}_{L_2}^{W}$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e{+}7$	$3.3562e{+}7$	$1.186e{+}6$	$3.6764e{+}8$	$7 + 8\times$
Z_2	$3.6427e{+}5$	$6.5007e{+}5$	$3.6582e{+}2$		$19 + 24\times$

Z_1: canonical form

Z_2: balanced state-space

Example – 2

	$\bar{M}_{L_2}^{W}$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e{+}7$	$3.3562e{+}7$	$1.186e{+}6$	$3.6764e{+}8$	$7 + 8\times$
Z_2	$3.6427e{+}5$	$6.5007e{+}5$	$3.6582e{+}2$		$19 + 24\times$
Z_3	$1.5267e{+}3$	$1.6689e{+}4$	$1.7455e{+}2$		$19 + 24\times$

Z_3: $\bar{M}_{L_2}$-optimal state-space

Example – 2

	$\bar{M}_{L_2}^{W}$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e{+}7$	$3.3562e{+}7$	$1.186e{+}6$	$3.6764e{+}8$	$7+8\times$
Z_2	$3.6427e{+}5$	$6.5007e{+}5$	$3.6582e{+}2$		$19+24\times$
Z_3	$1.5267e{+}3$	$1.6689e{+}4$	$1.7455e{+}2$		$19+24\times$
Z_4	$1.6272e{+}3$	$2.7425e{+}3$	$1.1778e{+}2$		$19+24\times$

Z_4: $\bar{\Psi}$-optimal state-space

Example – 2

	$\bar{M}_{L_2}^{W}$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e{+}7$	$3.3562e{+}7$	$1.186e{+}6$	$3.6764e{+}8$	$7 + 8\times$
Z_2	$3.6427e{+}5$	$6.5007e{+}5$	$3.6582e{+}2$		$19 + 24\times$
Z_3	$1.5267e{+}3$	$1.6689e{+}4$	$1.7455e{+}2$		$19 + 24\times$
Z_4	$1.6272e{+}3$	$2.7425e{+}3$	$1.1778e{+}2$		$19 + 24\times$
Z_5	$1.9474e{+}13$	$1.2294e{+}13$	$3.2261e{-}3$		$19 + 24\times$

Z_5: $\bar{G}$-optimal state-space

Example – 2

	$\bar{M}_{L_2}^W$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_1	$1.9046e+7$	$3.3562e+7$	$1.186e+6$	$3.6764e+8$	$7+8\times$
Z_2	$3.6427e+5$	$6.5007e+5$	$3.6582e+2$	$1.1387e+5$	$19+24\times$
Z_3	$1.5267e+3$	$1.6689e+4$	$1.7455e+2$	$5.4111e+4$	$19+24\times$
Z_4	$1.6272e+3$	$2.7425e+3$	$1.1778e+2$	$3.6512e+4$	$19+24\times$
Z_5	$1.9474e+13$	$1.2294e+13$	$3.2261e-3$	$1.7239e+10$	$19+24\times$
Z_6	$2.8696e+3$	$4.5371e+3$	$7.9809e-3$	$6.0078e+0$	$19+24\times$

Tradeoff measure: we are looking for a *good enough* realization:

$$\bar{TO}(Z) \triangleq \frac{\bar{M}_{L_2}^W(Z)}{\bar{M}_{L_2}^{W\ opt}} + \frac{\bar{\Psi}(Z)}{\bar{\Psi}^{opt}} + \frac{\bar{G}(Z)}{\bar{G}^{opt}}$$

Z_6: $\bar{TO}$-optimal state-space

Example – 3

	$\bar{M}_{L_2}^W$	$\bar{\Psi}$	$\bar{G}$	$\bar{TO}$	**Nb. op.**
Z_7	$1.5342e{-2}$	$8.1051e{-2}$	$2.8082e{-8}$	$4.5466e{+0}$	$11 + 12\times$
Z_8	$1.5341e{-2}$	$8.089e{-2}$	$4.217e{-8}$	$4.8783e{+0}$	$11 + 16\times$
Z_9	$1.1388e{-1}$	$2.8203e{-2}$	$3.7783e{-6}$	$9.8937e{+1}$	$11 + 16\times$
Z_{10}	$1.5342e{-2}$	$8.0015e{-2}$	$4.1742e{-8}$	$4.8371e{+0}$	$11 + 16\times$
Z_{11}	$1.6065e{-2}$	$3.8802e{-2}$	$4.7451e{-8}$	$3.5597e{+0}$	$11 + 16\times$

Z_7: $\gamma = \begin{pmatrix}1111\end{pmatrix}^\top$: δ-Direct Form II
Z_8: $\bar{M}_{L_2}^W$-optimal ρDFIIt
Z_9: $\bar{\Psi}$-optimal ρDFIIt
Z_{10}: $\bar{G}$-optimal ρDFIIt
Z_{11}: $\bar{TO}$-optimal ρDFIIt

Word-lengths optimization

word-length optimization

A 2^{nd} possible optimization concerns the hardware implementation

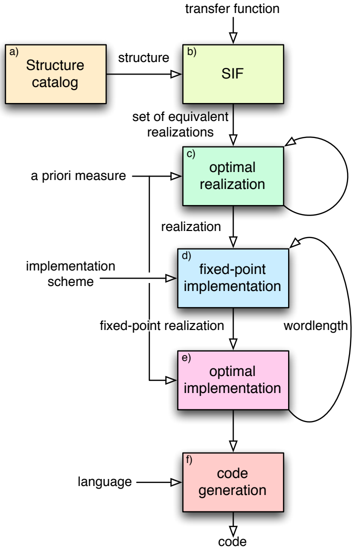
On FPGA or ASIC, one can use multiple wordlength paradigm

Optimal word-length

Let Z be a realization and $w = (w_Z, w_T, w_X, w_Y, w_{ADD})$ the word-length
We consider $\mathcal{J}$ a computational cost (area, power consumption, WCET, etc.). The optimal wordlength problem is:

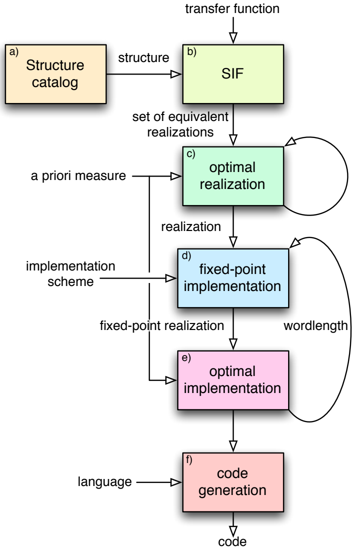
$$w^{\text{opt}} = \underset{w \in \mathcal{B}}{\arg\min}\ \mathcal{J}(w) \text{ with } SQNR \geqslant SQNR_{\min}$$

Global methodology



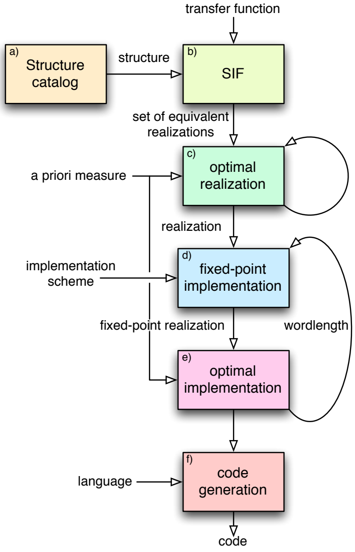
a) Choose a structure (state-space, ρDFIIt, ...)

Global methodology



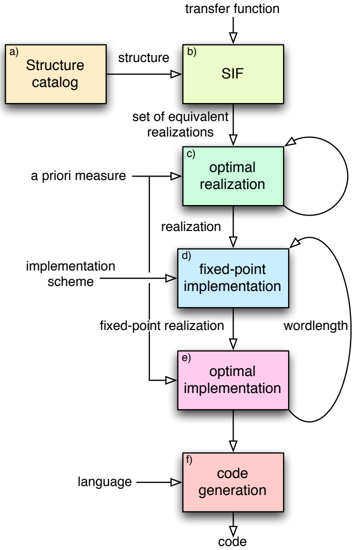
b) Formalization SIF and set of equivalent realizations

Global methodology



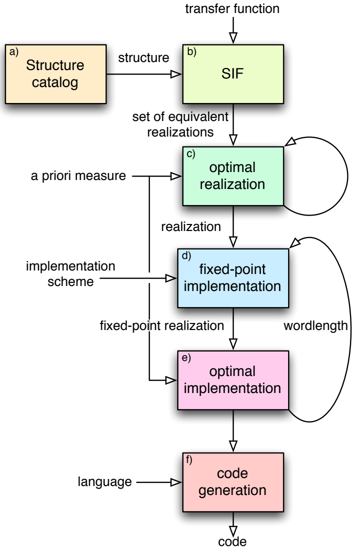
c) Optimization of an *a priori* criterion

Global methodology



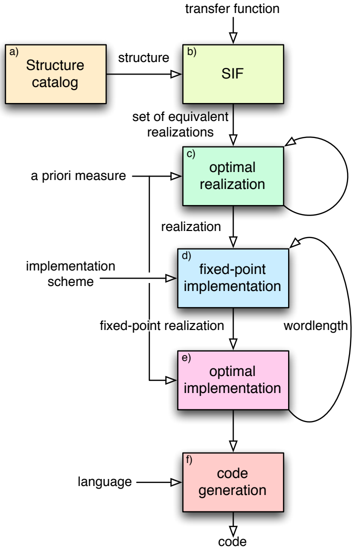
d) Given some word-lengths and an sum-of-products evaluation scheme, the fixed-point representation are defined

Global methodology



e) Optimization of the implementation: optimization of a computational cost under *a priori* criterion constraint

Global methodology



f) Fixed-point code generation (C, VHDL, ...)

Example – 1

An other example to be implemented on 8-bit device (with 16-bit accumulator)

$$H(z) = \frac{0.02088z^2 + 0.04176z + 0.02088}{z^2 - 1.651z + 0.7342}$$

Example – 1

An other example to be implemented on 8-bit device (with 16-bit accumulator)

$$H(z) = \frac{0.02088z^2 + 0.04176z + 0.02088}{z^2 - 1.651z + 0.7342}$$

Some possible realizations:

$R1$: Direct Form I

$R2$: Direct Form II

$R3$: Balanced state-space

$R4$: $\sigma^2_{\Delta H}$-optimal state-space

$R5$: optimal state-space with δ-optimal

$R6$: optimal ρ-Direct Form II

Example – 2

Roundoff Before Multiplication - 8 bits

realisation	$\sigma^2_{\Delta H}$	$\|H - H^\dagger\|_2$	$\sigma^2_{\xi'}$	$Nb + \quad \times$
$R1$	114.8242	0.024423	0.0092127	$4 + 5\times$
$R2$	51.0143	0.02797	0.017759	$4 + 6\times$
$R3$	8.7612	0.012219	0.0012579	$6 + 9\times$
$R4$	11.3538	0.0089503	0.0012985	$6 + 9\times$
$R5$	4.6303	0.0040723	0.0037008	$8 + 11\times$
$R6$	7.3958	0.010575	0.00064769	$6 + 7\times$

Algorithms

Input: u: 8 bits integer
Output: y: 8 bits integer
Data: xn: array [1..5] of 8 bits integers
Data: T: 8 bits integer
Data: Acc: 16 bits integer
begin

```
    // Intermediate variables
    Acc ← (xn(1) ∗ −47);
    Acc ← Acc + (xn(2) ∗ 106);
    Acc ← Acc + xn(3);
    Acc ← Acc + (xn(4) ∗ 3);
    Acc ← Acc + u;
    T1 ← Acc >> 6;
    // States
    xn(1) ← xn(2);
    xn(2) ← T1;
    xn(3) ← xn(4);
    xn(4) ← u;
    // Outputs
    y ← T1;
```

end

Algorithm 1: R1 : Direct Form I

Input: u: 8 bits integer
Output: y: 8 bits integer
Data: xn: array [1..5] of 8 bits integers
Data: T: array [1..5] of 8 bits integers
Data: Acc: 16 bits integer
begin

```
    // Intermediate variables
    Acc ← (xn(1) ∗ −31);
    Acc ← Acc + (xn(2) ∗ 114);
    Acc ← Acc + (u ∗ −8);
    T1 ← Acc >> 7;
    Acc ← (xn(1) ∗ −64);
    Acc ← Acc + (xn(2) ∗ −116);
    Acc ← Acc + (u ∗ −114);
    T2 ← Acc >> 8;
    // States
    Acc ← T1 << 5;
    Acc ← Acc + xn(1) << 6;
    xn(1) ← Acc >> 6;
    Acc ← T2 << 5;
    Acc ← Acc + xn(2) << 6;
    xn(2) ← Acc >> 6;
    // Outputs
    Acc ← (xn(1) ∗ −92);
    Acc ← Acc + (xn(2) ∗ −31);
    Acc ← Acc + (u ∗ 3);
    y ← Acc >> 7;
```

end

Algorithm 2: R5 - δ-state-space

3 Further work

Outline

Research for new structures

Realizations

New structures

Exploit new interesting structures

- lattice and lattice wave digital filters
- ρ-modal state-space
- LGC et LCW-structures
- sparse state-spaces
- combine efficiently all the realizations
- ...

parametrized filters, LPV

Parametrized filters – 1

We are also considering filters/controllers where the coefficients depend on extra parameters θ

parametrized filters, LPV

Parametrized filters – 1

We are also considering filters/controllers where the coefficients depend on extra parameters θ

Example: 2nd order filter

$$H(z) = \frac{b_0 z^2 + b_1 z + b_2}{a_0 z^2 + a_1 z + a_2}$$

with

- $\mathbf{b_0} = gT^2, \quad \mathbf{b_1} = 2gT^2, \quad \mathbf{b_2} = gT^2$
- $\mathbf{a_0} = 4\xi\omega_c T + \omega_c^2 T^2 + 4$, $\mathbf{a_1} = 2\omega_c^2 T^2 - 8$, $\mathbf{a_2} = \omega_c^2 T^2 - 4\xi\omega_c T + 4$

parametrized filters, LPV

Parametrized filters – 1

We are also considering filters/controllers where the coefficients depend on extra parameters θ

Example: 2nd order filter

$$H(z) = \frac{b_0 z^2 + b_1 z + b_2}{a_0 z^2 + a_1 z + a_2}$$

with

- $\mathbf{b_0} = gT^2, \quad \mathbf{b_1} = 2gT^2, \quad \mathbf{b_2} = gT^2$
- $\mathbf{a_0} = 4\xi\omega_c T + \omega_c^2 T^2 + 4$, $\mathbf{a_1} = 2\omega_c^2 T^2 - 8$, $\mathbf{a_2} = \omega_c^2 T^2 - 4\xi\omega_c T + 4$

$$\theta = \begin{pmatrix} g \\ T \\ \omega_c \\ \xi \end{pmatrix} \quad \text{or} \quad \theta = \begin{pmatrix} g \\ Fe \\ F_c \\ \pi \\ \xi \end{pmatrix}, \dots$$

Parametrized filters – 2

parametrized filters, LPV

Question: What will be the impact of the quantization of these parameters (π for example)?

Parametrized filters – 2

parametrized filters, LPV

Question: What will be the impact of the quantization of these parameters (π for example)?
So, we want to consider

- the quantization of parameters θ
- the computation of $Z(\theta^\dagger)$ and the associated quantization

Parametrized filters – 2

parametrized filters, LPV

Question: What will be the impact of the quantization of these parameters (π for example)?
So, we want to consider

- the quantization of parameters θ
- the computation of $Z(\theta^{\dagger})$ and the associated quantization

What is the *best* realization for all $\theta \in \Theta$?

Hardware choices

hardware choices

A lot of possibilities for the hardware implementation

- Use appropriate operators (FPGA)
  - use *optimized*, *dedicated* tools
  - flat computations or combined operators
  - use dedicated DSP blocks
  - multiple constants multiplications
  - etc.

Hardware choices

A lot of possibilities for the hardware implementation

- Use appropriate operators (FPGA)
  - use *optimized*, *dedicated* tools
  - flat computations or combined operators
  - use dedicated DSP blocks
  - multiple constants multiplications
  - etc.
- Residue Number Systems
  - Which basis $\{2^k, 2^k - 1, 2^k + 1\}$ ou $\{2^k \pm c\}$
  - Scaling could be a problem

Best fixed-point filter

best fixed-point filter

Given a *real* transfer function H, is it possible to find the *best* ($\|.\|_2$ or $\|.\|_\infty$) transfer function $H^\dagger$ with fixed-point coefficients ?

Best fixed-point filter

best fixed-point filter

Given a *real* transfer function H, is it possible to find the *best* ($\|.\|_2$ or $\|.\|_\infty$) transfer function $H^\dagger$ with fixed-point coefficients ?

- Transpose work done on polynomials
  - possible for FIR
  - much more complicated for IIR
- For a given structure, find the best fixed-point coefficients $Z^\dagger$

Best fixed-point filter

best fixed-point filter

Given a *real* transfer function H, is it possible to find the *best* ($\|.\|_2$ or $\|.\|_\infty$) transfer function $H^\dagger$ with fixed-point coefficients ?

- Transpose work done on polynomials
  - possible for FIR
  - much more complicated for IIR
- For a given structure, find the best fixed-point coefficients $Z^\dagger$

Very promising work, but need dedicated time

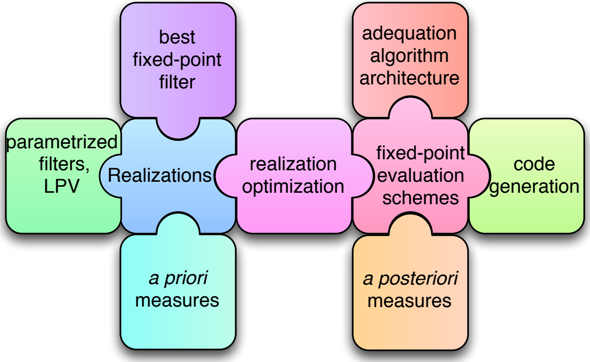
Multi-objectives optimization

As seen before, we have a multi-objective optimization problem (performance/accuracy/computational cost)

- We are looking at different strategies:
  - GA for ρ-operators
  - approximate *a priori* criteria by LMI
- We would like to show *where* are the realizations

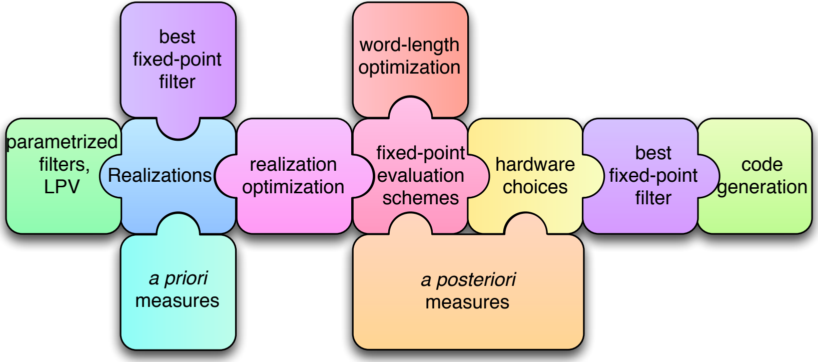
Methodology – 1

For software implementation, the different parts are (for the moment):



Methodology – 2

For hardware implementation, much more parts are required:



4 Conclusions

Outline

Conclusions

Conclusion

We try to answer the following question:

For a given controller, how to produce *optimal* implementation ?

The main idea is to propose a global approach:

- Model all the equivalent realizations
- Propose some analytical robustness criteria
- Model the fixed-point implementation (evaluation scheme)
- Propose some precision/performance/cost criteria

Perspectives – 1

A lot of work still need to be done:

- Look for new realizations (error-feedback, mix ρ and q Direct Form, etc.)
- Multi-objectives optimization
- Parametrized or LPV controllers
- Study different arithmetics (RNS, etc.)
- Given a real controller, find the closest controller with fixed-point coefficients
- Etc.

Perspectives – 1

A lot of work still need to be done:

- Look for new realizations (error-feedback, mix ρ and q Direct Form, etc.)
- Multi-objectives optimization
- Parametrized or LPV controllers
- Study different arithmetics (RNS, etc.)
- Given a real controller, find the closest controller with fixed-point coefficients
- Etc.

These future work should lead to a tool suite allowing to automatically transform a controller into fixed-point code with controlled numerical behavior

$\rightarrow$ An initial matlab toolbox have been designed: the *Finite Wordlength Realization Toolbox* (been redesign with Python)

Perspectives – 2

Of course, it seems very promising to *plug* this approach to hardware optimized synthesis

What can we do together ?

Any questions ?

5 Appendix

Outline

Filters

Signals

- Continous-time signal: $s(t)$, $t \in \mathbb{R}$
- Discrete-time signal: $s'(k)$, $k \in \mathbb{Z}$

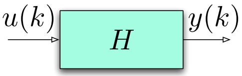
$$s'(k) \triangleq s(kT_e), \qquad T_e \text{ sampling period}$$

Filters

Signals

- Continous-time signal: $s(t)$, $t \in \mathbb{R}$
- Discrete-time signal: $s'(k)$, $k \in \mathbb{Z}$

$$s'(k) \triangleq s(kT_e), \qquad T_e \text{ sampling period}$$



Filter

A filter is defined by

- its impulse response
- its transfer function
- its frequency response

Transfer function

$$H(z) = \frac{Y(z)}{U(z)}$$

where $X(z)$ and $Y(z)$ atre the $\mathcal{Z}$-transform of $u(k)$ and $y(k)$
($H(z)$ is the $\mathcal{Z}$-transform of the impulse response $h(k)$)

Transfer function

$$H(z) = \frac{Y(z)}{U(z)}$$

where $X(z)$ and $Y(z)$ atre the $\mathcal{Z}$-transform of $u(k)$ and $y(k)$
($H(z)$ is the $\mathcal{Z}$-transform of the impulse response $h(k)$)

$\mathcal{Z}$-transform

It is the discrete equivalent of the *Laplace*-transform:

$$\mathcal{Z}\left[x(k)\right] : \begin{array}{rcl} \mathcal{D}_{cv} & \rightarrow & \mathbb{C} \\ z & \mapsto & X(z) \end{array}$$

with

$$X(z) \triangleq \sum_{k=0}^{\infty} x(k) z^{-k}$$

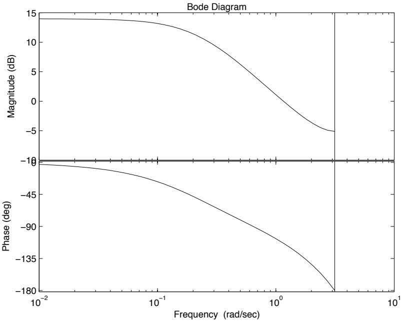
Frequency response

Frequency response: $H(e^{j\omega})$

- for $z = e^{j\omega}$, $\omega \in [0, 2\pi]$
- for $z = e^{j2\pi\frac{f}{Fe}}$ with $0 \leqslant f \leqslant Fe$

$$H(z) = \frac{1}{z-0.8}$$

$$\left|H(e^{j\Omega})\right|$$
$$\arg\left(H(e^{j\Omega})\right)$$



Back

Intermediate variables

The intermediate variables have been introduced to

- make explicit all the computations done
- show the linking of the computations

Intermediate variables

The intermediate variables have been introduced to

- make explicit all the computations done
- show the linking of the computations

Implicit Form

The intermediate variables are computed by:

$$J.T(k+1) = M.X(k) + N.U(k)$$

with J lower triangular with 1 on diagonal, so

- no need to compute J^{-1}
- an intermediate variable can be computed from another intermediate variable that have been previously computed (in the same step)
  $\Rightarrow$ is able to express computations such as
  $Y(k) = \left(M_1.\left(M_2 \ldots \left(M_i U_k\right)\right)\right)$

Opérateur δ

A realization with δ-operator is:

$$\left\{ \begin{array}{rcl} \delta[X(k)] & = & A_\delta X(k) + B_\delta U(k) \\ Y(k) & = & C_\delta X(k) + D_\delta U(k) \end{array} \right. \qquad \delta \triangleq \frac{q-1}{\Delta}$$

Opérateur δ

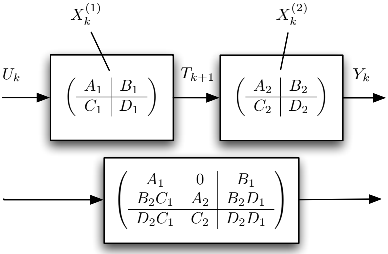
A realization with δ-operator is:

$$\left\{ \begin{array}{rcl} \delta[X(k)] & = & A_\delta X(k) + B_\delta U(k) \\ Y(k) & = & C_\delta X(k) + D_\delta U(k) \end{array} \right. \qquad \delta \triangleq \frac{q-1}{\Delta}$$

And it is given by (SIF):

$$\begin{pmatrix} I & 0 & 0 \\ -\Delta I & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & A_\delta & B_\delta \\ 0 & I & 0 \\ 0 & C_\delta & D_\delta \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

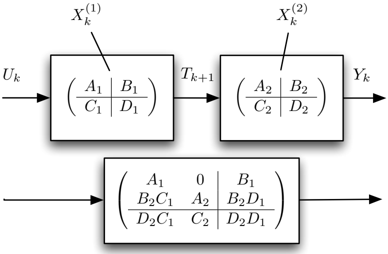
Cascad decomposition



$$\begin{pmatrix} I & 0 & 0 \\ \begin{pmatrix} 0 \\ -B_2 \end{pmatrix} & I & 0 \\ -D_2 & 0 & I \end{pmatrix} \begin{pmatrix} T(k+1) \\ \begin{pmatrix} X(k+1)^{(1)} \\ X(k+1)^{(2)} \end{pmatrix} \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & \begin{pmatrix} C_1 & 0 \end{pmatrix} & D_1 \\ 0 & \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} & \begin{pmatrix} B_1 \\ 0 \end{pmatrix} \\ 0 & \begin{pmatrix} 0 & C_2 \end{pmatrix} & 0 \end{pmatrix} \begin{pmatrix} T(k) \\ \begin{pmatrix} X(k)^{(1)} \\ X(k)^{(2)} \end{pmatrix} \\ U(k) \end{pmatrix}$$

Back

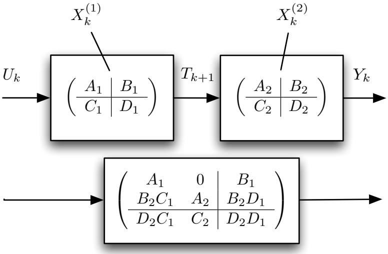
Cascad decomposition



$$\begin{pmatrix} I & 0 & 0 \\ \begin{pmatrix} 0 \\ -B_2 \end{pmatrix} & I & 0 \\ -D_2 & 0 & I \end{pmatrix} \begin{pmatrix} \color{red}{\mathbf{T(k+1)}} \\ \begin{pmatrix} X(k+1)^{(1)} \\ X(k+1)^{(2)} \end{pmatrix} \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & \begin{pmatrix} \color{blue}{\mathbf{C_1}} & 0 \end{pmatrix} & \color{blue}{\mathbf{D_1}} \\ 0 & \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} & \begin{pmatrix} B_1 \\ 0 \end{pmatrix} \\ 0 & \begin{pmatrix} 0 & C_2 \end{pmatrix} & 0 \end{pmatrix} \begin{pmatrix} T(k) \\ \begin{pmatrix} X(k)^{(1)} \\ X(k)^{(2)} \end{pmatrix} \\ U(k) \end{pmatrix}$$

Back

Cascad decomposition



$$\begin{pmatrix} I & 0 & 0\\ \begin{pmatrix}0\\ \mathbf{-B_2}\end{pmatrix} & I & 0\\ \mathbf{-D_2} & 0 & I\end{pmatrix}\begin{pmatrix}T(k+1)\\ \begin{pmatrix}X(k+1)^{(1)}\\ \mathbf{X(k+1)^{(2)}}\end{pmatrix}\\ \mathbf{Y(k)}\end{pmatrix} = \begin{pmatrix}0 & \begin{pmatrix}C_1 & 0\end{pmatrix} & D_1\\ 0 & \begin{pmatrix}A_1 & 0\\ 0 & A_2\end{pmatrix} & \begin{pmatrix}B_1\\ 0\end{pmatrix}\\ 0 & \begin{pmatrix}0 & C_2\end{pmatrix} & 0\end{pmatrix}\begin{pmatrix}T(k)\\ \begin{pmatrix}X(k)^{(1)}\\ X(k)^{(2)}\end{pmatrix}\\ U(k)\end{pmatrix}$$

Back

ρ-DFIIt – 1

It is possible to **re-parametrized** the transfer function:

$$H(z) = \frac{b_0 z^n + b_1 z^{n-1} + \ldots + b_{n-1} z + b_n}{z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n}$$

by

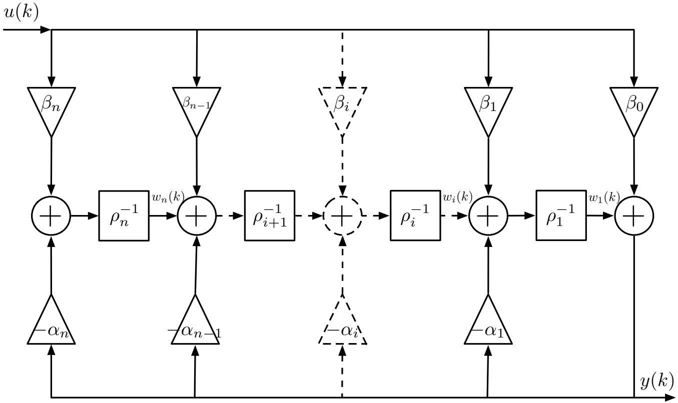
$$H(z) = \frac{\beta_0 \varrho_0(z) + \beta_1 \varrho_1(z) + \ldots + \beta_{n-1} \varrho_{n-1}(z) + \beta_n \varrho_n(z)}{1 + \alpha_1 \varrho_1(z) + \ldots + \alpha_{n-1} \varrho_{n-11}(z) + \alpha_n \varrho_n(z)}$$

with

$$\varrho_i : z \mapsto \prod_{j=i+1}^{n} \rho_j(z) \text{ and } \rho_i : z \mapsto \frac{z - \gamma_i}{\Delta_i}$$

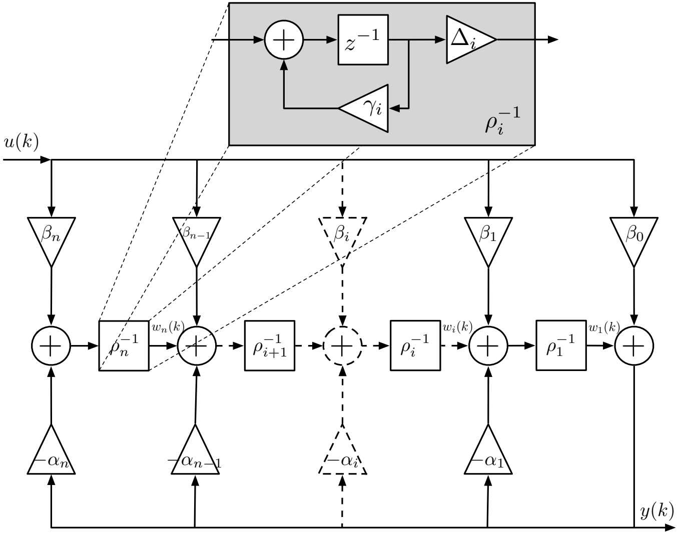
and to use the ρ_i-operator in a direct form II transposed.

ρ-DFIIt – 2



$u(k)$
β_n
β_{n-1}
β_i
β_1
β_0
ρ_n^{-1}
$w_n(k)$
ρ_{i+1}^{-1}
ρ_i^{-1}
$w_i(k)$
ρ_1^{-1}
$w_1(k)$
$-\alpha_n$
$-\alpha_{n-1}$
$-\alpha_i$
$-\alpha_1$
$y(k)$

ρ-DFIIt – 2

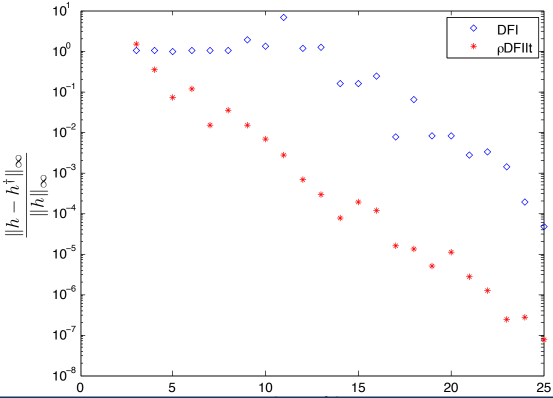

z^{-1}
Δ_i
γ_i
ρ_i^{-1}
$u(k)$
β_n
β_{n-1}
β_i
β_1
β_0
ρ_n^{-1}
$w_n(k)$
ρ_{i+1}^{-1}
ρ_i^{-1}
$w_i(k)$
ρ_1^{-1}
$w_1(k)$
$-\alpha_n$
$-\alpha_{n-1}$
$-\alpha_i$
$-\alpha_1$
$y(k)$

ρ-DFIIt – 3

Example : a 6th-order Butterworth filter

We looking at the transfer function relative error $\frac{\|h - h^\dagger\|_\infty}{\|h\|_\infty}$, with respect to the wordlengths used for the coefficients:

- Direct Form I (12 coefs)
- ρ-DFIIt (18 coefs)



ρ-DFIIt – 3

Example : a 6th-order Butterworth filter

We looking at the transfer function relative error $\frac{\|h-h^\dagger\|_\infty}{\|h\|_\infty}$, with respect to the wordlengths used for the coefficients:

- Direct Form I (12 coefs)
- ρ-DFIIt (18 coefs)

So with *equivalent* precision

	area (slices)
DFI (18 bits)	1071
ρDFIIt (5 bits)	206

Preliminar results for FPGA implementation (Xilink Virtex4) with Residue Number System $\mathcal{B} = \{2^k, 2^k - 1, 2^k + 1\}$

Back

ρ-DFIIt – 4

It can be expressed with the SIF:

$$Z = \left(\begin{array}{cccc:ccc:c}
-1 & & & & \Delta_1 & & & \beta_0 \\
& \ddots & & & & \Delta_2 & & 0 \\
& & \ddots & & & & \ddots & \vdots \\
& & & -1 & & & \Delta_n & 0 \\
\hdashline
-\alpha_1 & 1 & & & \gamma_1 & & & \beta_1 \\
-\alpha_2 & 0 & \ddots & & & \gamma_2 & & \beta_2 \\
\vdots & & \ddots & 1 & & & \ddots & \vdots \\
-\alpha_n & & & 0 & & & \gamma_n & \beta_n \\
\hdashline
1 & 0 & \dots & 0 & 0 & \dots\ \dots & 0 & 0
\end{array}\right)$$

Back

Statistical approach – 1

An other approach is to consider that the coefficients Z are modified in $Z^\dagger = Z + \Delta Z$,
where ΔZ_{ij} are random independent centered variables, uniformly distributed in $-\frac{q_{Z_{ij}}}{2} \leqslant \Delta Z_{ij} < \frac{q_{Z_{ij}}}{2}$,
and $q_{Z_{ij}}$ are the quantization step of Z_{ij} (depends on its fixed-point representation).

Statistical approach – 1

An other approach is to consider that the coefficients Z are modified in $Z^\dagger = Z + \Delta Z$,
where ΔZ_{ij} are random independent centered variables, uniformly distributed in $-\frac{q_{Z_{ij}}}{2} \leqslant \Delta Z_{ij} < \frac{q_{Z_{ij}}}{2}$,
and $q_{Z_{ij}}$ are the quantization step of Z_{ij} (depends on its fixed-point representation).
Their order-2 moment is

$$\sigma^2_{\Delta Z_{ij}} \triangleq E\left\{(\Delta Z_{ij})^2\right\} = \frac{q^2_{Z_{ij}}}{12}\delta_{Z_{ij}}$$

Statistical approach – 2

The idea behind this is that H is changed in $H^\dagger = H + \Delta H$, where ΔH is a transfer function with random variables as coefficients.
So we define a new measure

$$\sigma^2_{\Delta H} \triangleq \frac{1}{2\pi} \int_0^{2\pi} E\left\{ \left\| \Delta H\left(e^{j\omega}\right) \right\|_F^2 \right\} d\omega$$

Statistical approach – 2

The idea behind this is that H is changed in $H^\dagger = H + \Delta H$, where ΔH is a transfer function with random variables as coefficients.
So we define a new measure

$$\sigma^2_{\Delta H} \triangleq \frac{1}{2\pi} \int_0^{2\pi} E\left\{ \left\| \Delta H\left(e^{j\omega}\right) \right\|_F^2 \right\} d\omega$$

$$\rightarrow \sigma^2_{\Delta H} = \left\| \frac{\partial H}{\partial Z} \times \Xi_Z \right\|_2^2$$

with

$$(\Xi_Z)_{ij} \triangleq \begin{cases} \frac{2^{-\beta_{Z_{ij}}+1}}{\sqrt{3}} \lfloor Z_{ij} \rfloor_2 (\delta_Z)_{ij} & \text{if } Z_{ij} \neq 0 \\ 0 & \text{if } Z_{ij} = 0 \end{cases}$$

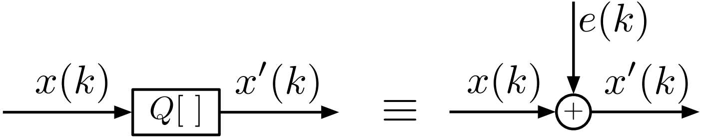
Statistical approach – 3

In a same way, the quantization of the coefficients changes λ_k in $\lambda_k^\dagger \triangleq \lambda_k + \Delta\lambda_k$.
So we define $\sigma^2_{\Delta|\lambda|}$ by

$$\sigma^2_{\Delta|\lambda|} \triangleq \sum_{k=1}^{n} \omega_k E\left\{ (\Delta\left|\lambda_k\right|)^2 \right\}$$

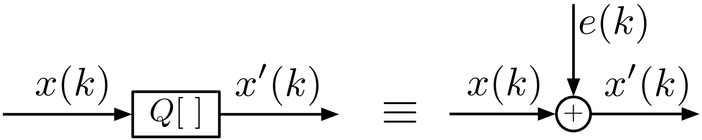
Back

Roundoff noise analysis – 1



Quantized a signal $x(k)$ is equivalent (under certain conditions) to add a white independent noise $e(k)$ with known moments:

Roundoff noise analysis – 1



Quantized a signal $x(k)$ is equivalent (under certain conditions) to add a white independent noise $e(k)$ with known moments:

Right-shift of d bits:

	truncation	best roundoff
$\mu_e \triangleq E\{e(k)\}$	$2^{-\gamma-1}(1-2^{-d})$	$2^{-\gamma-d-1}$
$\sigma_e^2 \triangleq E\{e(k)^\top e(k)\}$	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$

Roundoff noise analysis – 2

During the implementation, the algorithm becomes:

$$\left\{\begin{array}{rcl} J.T(k+1) & \leftarrow & M.X(k) + N.U(k) \\ X(k+1) & \leftarrow & K.T(k+1) + P.X(k) + Q.U(k) \\ Y(k) & \leftarrow & L.T(k+1) + R.X(k) + S.U(k) \end{array}\right.$$

Roundoff noise analysis – 2

During the implementation, the algorithm becomes:

$$\left\{ \begin{array}{rcl} J.T(k+1) & \leftarrow & M.X(k) + N.U(k) + \xi_T(k) \\ X(k+1) & \leftarrow & K.T(k+1) + P.X(k) + Q.U(k) + \xi_X(k) \\ Y(k) & \leftarrow & L.T(k+1) + R.X(k) + S.U(k) + \xi_Y(k) \end{array} \right.$$

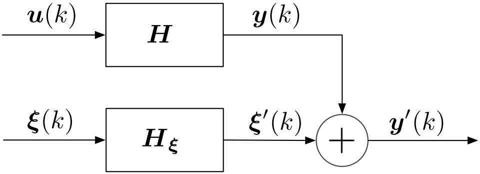
The roundoff leads to the add of the noise $\xi(k)$:

$$\xi(k) \triangleq \begin{pmatrix} \xi_T(k) \\ \xi_X(k) \\ \xi_Y(k) \end{pmatrix}$$

Roundoff noise analysis – 3

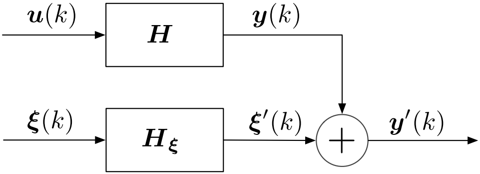
Back

The implemented system is then equivalent to



Roundoff noise analysis – 3 ◂ Back

The implemented system is then equivalent to

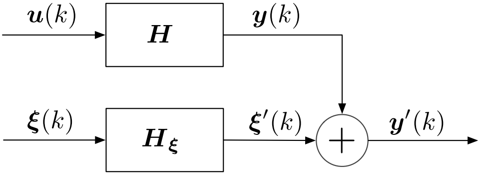


The Signal to Quantization Noise ratio is then defined by:

$$SQNR = \frac{\sigma_Y^2}{\sigma_{\xi'}^2}$$

Roundoff noise analysis – 3 Back

The implemented system is then equivalent to



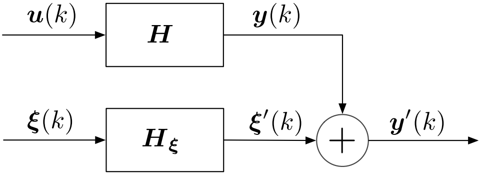
The Signal to Quantization Noise ratio is then defined by:

$$SQNR = \frac{\sigma_Y^2}{\sigma_{\xi'}^2}$$

H_1 is easily obtained, and $\sigma_{\xi'}^2 = \|H_\xi \varphi_\xi\|_2^2$ where φ_ξ is such $\psi_\xi = \varphi_\xi \varphi_\xi^\top$ and ψ_ξ the covariance matrix of ξ

Roundoff noise analysis – 3 Back

The implemented system is then equivalent to



The Signal to Quantization Noise ratio is then defined by:

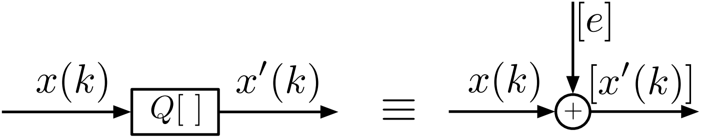
$$SQNR = \frac{\sigma_Y^2}{\sigma_{\xi'}^2}$$

H_1 is easily obtained, and $\sigma_{\xi'}^2 = \|H_\xi \varphi_\xi\|_2^2$ where φ_ξ is such $\psi_\xi = \varphi_\xi \varphi_\xi^\top$ and ψ_ξ the covariance matrix of ξ

φ_ξ only depends on implementation choices, whereas H_ξ only depends on the choice of the realization.

Link with interval error – 1

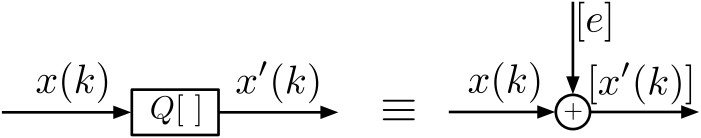
NEW!



Quantized a signal $x(k)$ is equivalent to add an interval error $[e](k)$:

Link with interval error – 1

NEW!



Quantized a signal $x(k)$ is equivalent to add an interval error $[e](k)$:

Right-shift of d bits:

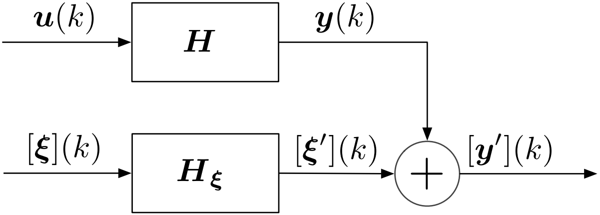
	truncation	best roundoff
mid	$\frac{q}{2}$	0
rad	$\frac{q}{2}$	$\frac{q}{2}$

with $q = 2^{-\gamma-d}$

Link with interval error – 2

NEW!

The implemented system is still equivalent to



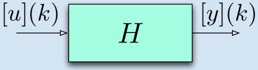
with $[\xi](k)$ the interval error added *in* the system

Link with interval error – 3 ◂ Back

NEW!

Interval through a filter

Let H be a SISO filter and $[u](k)$ be an interval input, centered in u (constant), with radius r_x (constant).



Then $[y](k)$ is an interval signal, centered in y (constant), with radius r_y such as

$$y = H(0)u, \quad \text{and} \quad r_y \leqslant \|H\|_{\ell_1} r_x$$

$H(0)$ is the DC-gain of the filter H.

The ℓ_1-norm or H is defined by $\|H\|_{\ell_1} \triangleq \sum_{k=0}^{\infty} |h(k)|$, where $h(k)$ is its impulse response.
If H is a state-space (A, B, C, d), then $\|H\|_{\ell_1} = \sum_{k=0}^{\infty} \left| CA^k B \right| + d$.