ρ-Direct Form transposed II and Residue Number Systems for filter implementations

JC. Bajard, L-S Didier and T. Hilaire
LIP6, University Pierre et Marie Curie (UPMC), Paris, France

7–10 July 2011 — MWSCAS'11, Seoul

Pitch

Residue Number System offer a good trade off for FIR implementation

- Small
- Fast
- Low power

In DSP and control applications IIR are more widely used

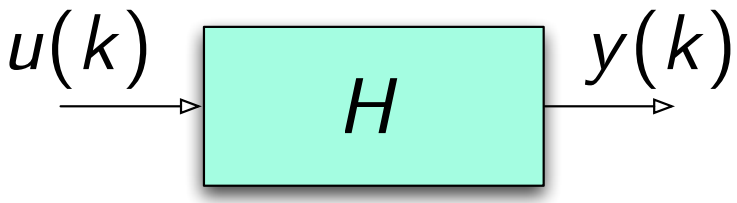
Is it a good strategy for IIR ?

IIR Filter

A IIR[1] filter is defined by its transfer function (input-output relationship in frequency domain)

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}$$

where the a_i and b_i are the coefficients of the filter.
Note that z^{-1} represents the delay operator.



[1] Infinite Impulse Response

IIR Filter

A IIR[1] filter is defined by its transfer function (input-output relationship in frequency domain)

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}$$

where the a_i and b_i are the coefficients of the filter.
Note that z^{-1} represents the delay operator.
In time-domain, the output at time k can be computed by

Direct Form I

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

[1] Infinite Impulse Response

Residue Number Systems

Integer representation.

A base of i relatively primes:

$$B_m^n = (m_1, m_2, \ldots, m_n)$$

The RNS representation of an integer X is:

$$X_{RNS} = (x_1, x_2, \ldots, x_n)$$

$$\left\| \begin{array}{l} x_1 \equiv X \bmod m_1 \\ x_2 \equiv X \bmod m_2 \\ \qquad \vdots \\ x_n \equiv X \bmod m_n \end{array} \right.$$

Chinese Remainder Theorem: An unique solution in $[a, a + M[$

$$M = \prod_{i=1}^{n} m_i$$

Operations

- Additions, multiplications and substractions with no carry propagation:

$$X \odot Y \bmod M = ((x_1 \odot y_1) \bmod m_1, \ldots, (x_n \odot y_n) \bmod m_n)$$

- Operations on small numbers
- Modular operation can be simplified using specific bases

Example : Base $\{3, 7, 13, 19\}$

$$X = 147 \qquad Y = 31$$
$$X_{RNS} = \{0, 0, 4, 14\} \qquad Y_{RNS} = \{1, 3, 5, 12\}$$

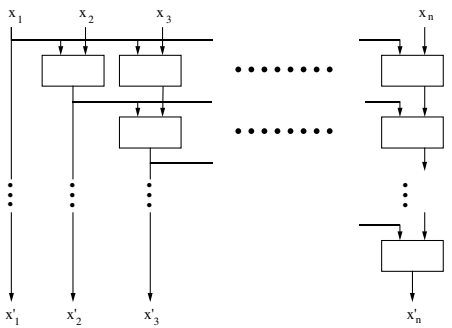
$$\begin{aligned}
X_{RNS} + Y_{RNS} &= \{ \ |0+1|_3 , \ |0+3|_7 , \ |4+5|_{13} , \ |14+12|_{19} \ \} \\
&= \{ \ 1 , \ 3 , \ 9 , \ 7 \ \} \\
&= 178 \\
X_{RNS} \times Y_{RNS} &= \{ \ |0 \times 1|_3 , \ |0 \times 3|_7 , \ |4 \times 5|_{13} , \ |14 \times 12|_{19} \ \} \\
&= \{ \ 0 , \ 0 , \ 7 , \ 16 \ \} \\
&= 4557
\end{aligned}$$

Difficulties

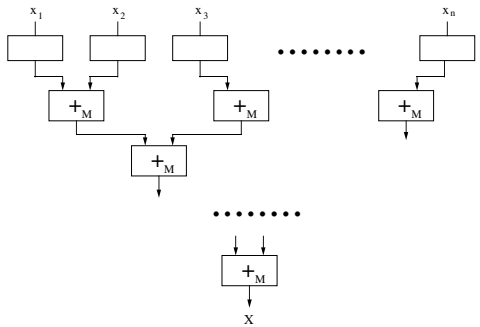
Example : Base $\{3, 7, 13, 19\}$

$$X = 147 \qquad Y = 31$$
$$X_{RNS} = \{0, 0, 4, 14\} \qquad Y_{RNS} = \{1, 3, 5, 12\}$$

- No easy magnitude estimation
- No easy overflow detection
- Conversions



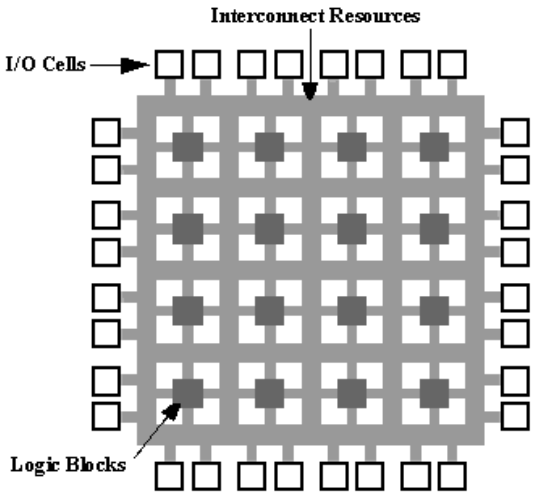
Conversion via MRS



Conversion via CRT

Technological issues

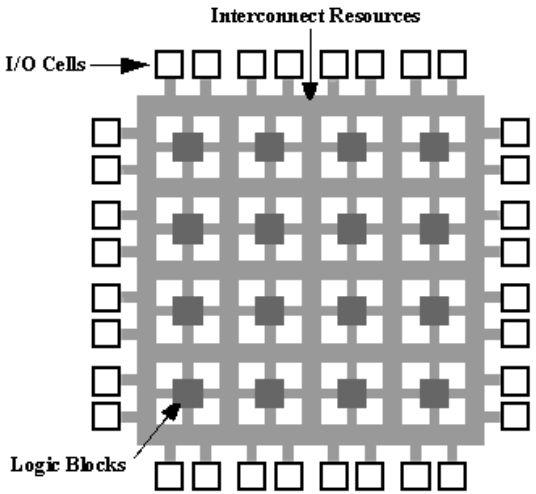
Targetted technology : Xilinx Virtex 4 FPGA



- A specific mechanism: Fast Carry propagation
- Carry ripple adder more efficient for medium width

Technological issues

Targetted technology : Xilinx Virtex 4 FPGA



- A specific mechanism: Fast Carry propagation
- Carry ripple adder more efficient for medium width

Key technological issue

Small delay increase for larger adders

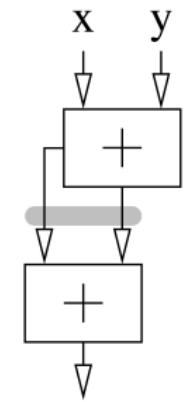
	5 bits	15 bits
Delay (ns)	6.6	7.1

FPGA implementation

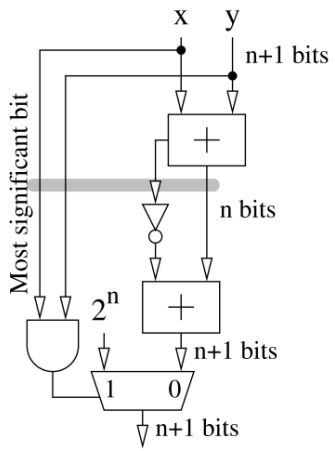
- **Specific modular base** $\{2^n - 1, 2^n, 2^n + 1\}$
- Modular additions
- Modular constant multiplication
- Scaling

FPGA implementation

- Specific modular base $\{2^n - 1, 2^n, 2^n + 1\}$
- **Modular additions**
- Modular constant multiplication
- Scaling



modulo $2^n - 1$ addition



modulo $2^n + 1$ additions

FPGA implementation

- Specific modular base $\{2^n - 1, 2^n, 2^n + 1\}$
- Modular additions
- **Modular constant multiplication**
- Scaling

- Up to 7 bits: lookup tables
  Roughly same delay as an adder
- Over 7 bits: shift and add-like algorithm
  delay similar up to 5 adders

FPGA implementation

- Specific modular base $\{2^n - 1, 2^n, 2^n + 1\}$
- Modular additions
- Modular constant multiplication
- **Scaling**
  Up to 7 bits: lookup tables

FPGA implementation

- Specific modular base $\{2^n - 1, 2^n, 2^n + 1\}$
- Modular additions
- Modular constant multiplication
- **Scaling**
  Over 7 bits: shift and add-like algorithm

Scaling is necessary

Direct Form I

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

DFI IIR in RNS on FPGA...

Filter	Scaling	Mul	Add	RNS WL (bits)	FP WL (bits)
DFI	13	13	12	13	36

		delay (ns)	area (slices)
DFI	fixed-point	20.61	1071
	RNS	54.76	3405

Because

- FPGA technology gives no gain in splitting wordlength
- Scaling is costly
- Wordlength too large to implement multiplication trough tables

ρDFIIt

Hopefully, some other algorithms are possible.

ρDFIIt

Hopefully, some other algorithms are possible.
i.e. it is possible to **reparametrized** the transfer function as

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

with

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

for some (ρ_i), (Δ_i).

ρDFIIt

Hopefully, some other algorithms are possible.
i.e. it is possible to **reparametrized** the transfer function as

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

with

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

for some (ρ_i), (Δ_i).

It uses $3n + 1$ parameters (instead of $2n + 1$) but is very efficient numerically

ρDFIIt advantage

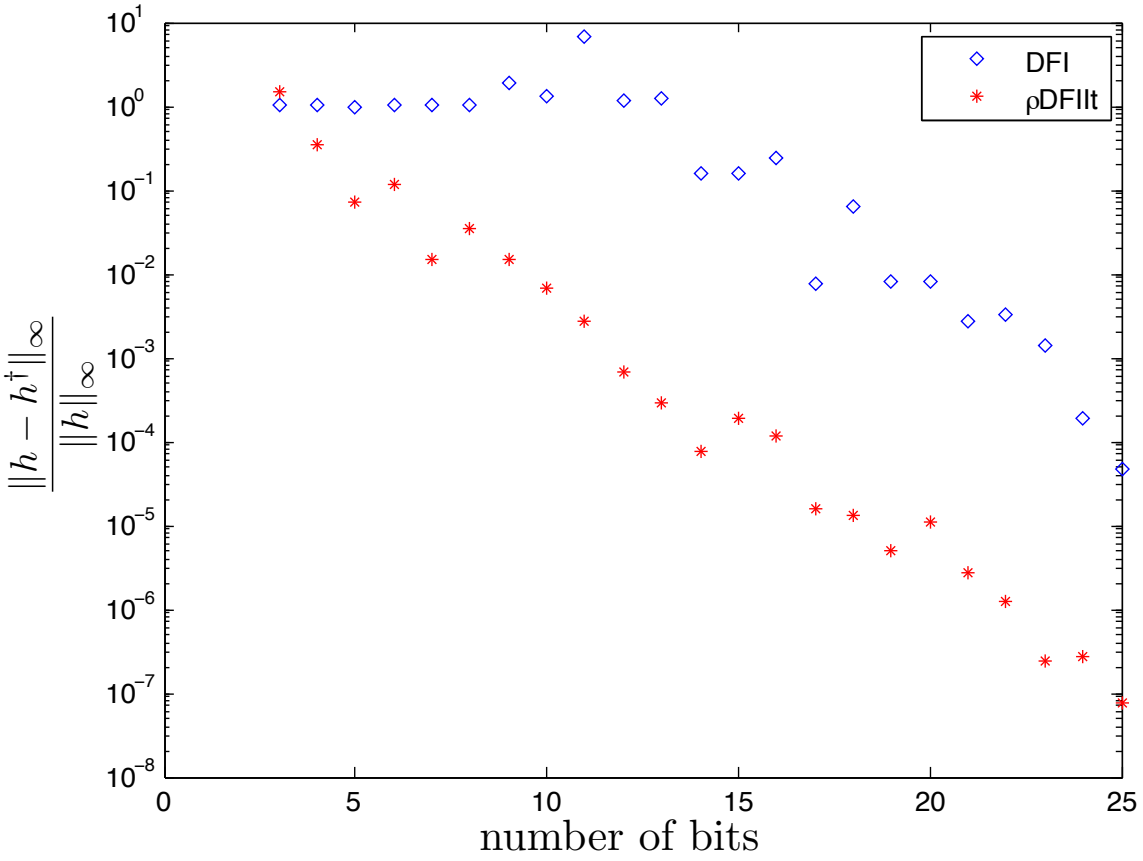
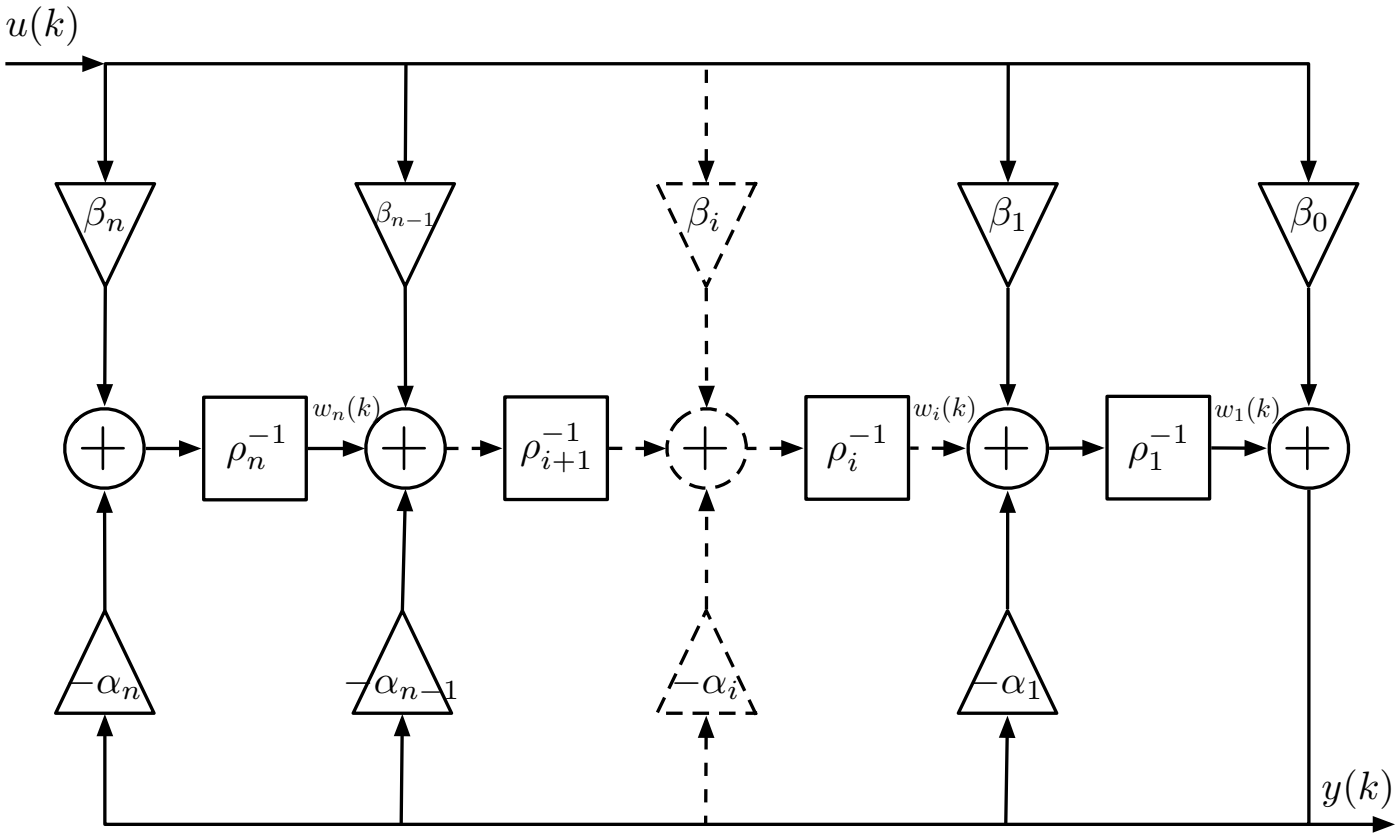


Figure: Relative difference between the ideal transfer function and the fixed-point implemented transfer function

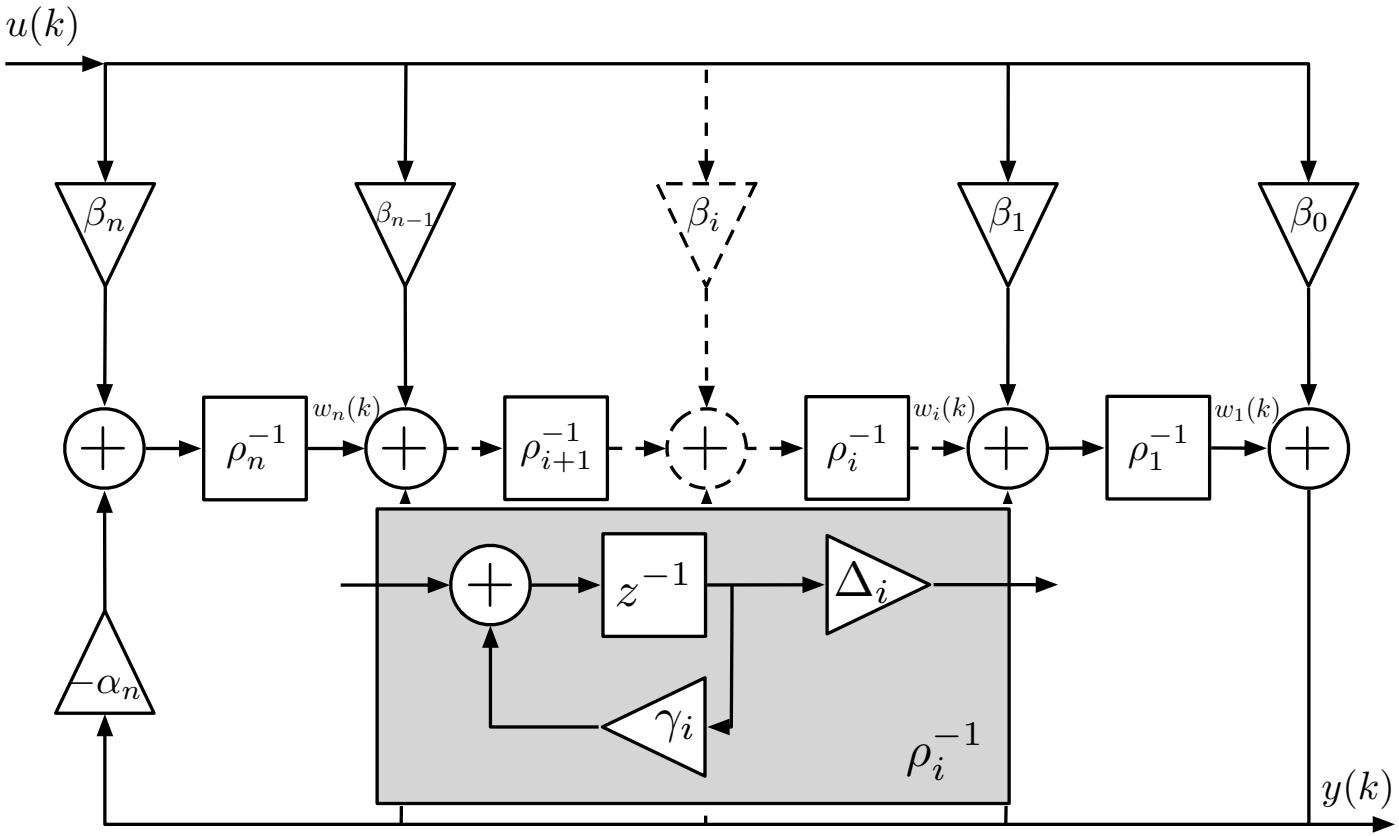
ρ-Direct-Form II

Then a *Direct Form* can be use with cascaded ρ_i^{-1}-operators.



ρ-Direct-Form II

Then a *Direct Form* can be use with cascaded ρ_i^{-1}-operators.



Results

		delay (ns)	area (slices)
DFI	fixed-point	20.61	1071
	RNS	54.76	3405
ρDFIIt	fixed-point	16.37	206
	RNS tables	17.03	1167

- Multiplications are more efficient trough tables
- FPGA technology gives no gain on the size and delay of RNS adders
- Scaling in RNS is costly
- New filter form gives improvement in delay and area

Conclusion - Discussion

- New efficient forms
- Forms with less scaling
- CMOS technology would give better results for RNS implementations
- Using larger base to benefit the parallelism gain. For instance $\{2^k-1, 2^k+1, 2^k-2^{k-r}-1, 2^k-2^{k-r}+1\}$