On the Design of an On-line Complex FIR Filter

Robert McIlhenny
Computer Science Department
California State University, Northridge
Northridge, CA 91330
rmcilhen@csun.edu

Miloš D. Ercegovac
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
milos@cs.ucla.edu

***Abstract*— In this paper, we present a novel implementation for an N-tap complex finite impulse response (FIR) filter, using complex number on-line arithmetic, based on adopting a redundant complex number system (RCNS) to represent complex operands as a single number. We present cost comparisons with (i) a real number on-line arithmetic approach, and (ii) a real number parallel arithmetic approach, to demonstrate a significant improvement in cost.**

I. Introduction

The N-tap finite impluse response (FIR) filter is defined as an output sequence y_n $(n = 1, \ldots)$ of an input sequence x_n $(n = 1, \ldots)$, in which

$$y_n = \sum_{k=0}^{N-1} h_k x_{n-k} \tag{1}$$

where $h_k (k = 0, 1, \ldots, N-1)$ are the filter coefficients. The standard implementation is shown in Figure 1. Assuming m-bit precision, it requires N m-bit multipliers and an N-operand m-bit adder. For a complex FIR filter, the filter coefficients as well the the input and output sequences are complex numbers. This significantly increases the size of the design, since an m-bit complex number multiplier is equivalent to 4 m-bit real number multipliers and 2 m-bit real number adders. Since area is a critical factor in FPGA design, we propose an approach that utilizes a radix $2j$ number system, and that yields a significant lower cost than an alternative radix 2 on-line implementation and a real number bit-parallel implementation.

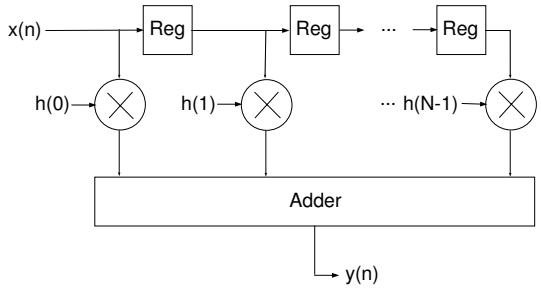

Fig. 1. FIR filter implementation

II. Complex number on-line floating-point arithmetic

On-line arithmetic [3] is a class of arithmetic operations in which all operations are performed digit serially, in a most significant digit first (MSDF) manner. Several advantages, compared to conventional parallel arithmetic include: (i) ability to overlap dependent operations, since on-line algorithms produce the output serially, most-significant digit first, enabling successive operations to begin before previous operations have completed; (ii) low-bandwidth communication, since intermediate results pass to and from modules digit-serially, so connections need only be one digit wide; and (iii) support for variable precision, since once a desired precision is obtained, successive outputs can be ignored. One of the key parameters of on-line arithmetic is the *on-line delay*, defined as the number of digits of the operand(s) necessary in order to generate the first digit of the result. Each successive digit of the result is generated one per cycle. This is illustrated in Figure 2, with on-line delay $\delta = 4$. The latency of an on-line arithmetic operator, assuming m-digit precision is then $\delta + m - 1$.

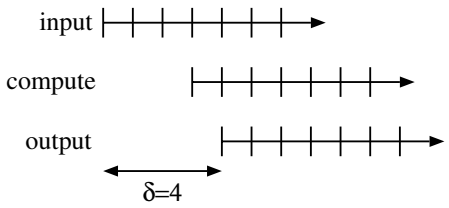

Fig. 2. On-line delay of a function

Complex number on-line arithmetic [5] uses a class of on-line arithmetic operators on complex number operands. For efficient representation, a *Redundant Complex Number System* (RCNS) [1] is adopted. A RCNS a radix rj system, in which digits are in the set $\{-a, \ldots, 0, \ldots, a\}$, where $r \geq 2$ and $\lceil r^2/2 \rceil \leq a \leq r^2 - 1$. Such a number system can be denoted $RCNS_{rj,a}$. A Redundant Complex Number System with $r = 2$, $a = 3$ denoted $RCNS_{2j,3}$, allows ease of the definition of primitive on-line arithmetic modules, as well as ease of conversion to and from other representations. This number system was introduced as Quarter-imaginary Number System in [4]. For implementation of the complex FIR filter,

0-7803-8622-1/04/$20.00 ©2004 IEEE

in order to permit a relatively wide range of input values, we assume floating-point arithmetic. Two on-line floating-point arithmetic operations are used: (i) $RCNS_{2j,3}$ on-line floating-point addition; and (ii) $RCNS_{2j,3}$ on-line floating-point constant coefficient multiplication. The recurrence algorithms and implementation parameters when mapped to a Xilinx Virtex FPGA are discussed in detail.

Using $RCNS_{2j,3}$, a floating-point complex number $x = (X_R + jX_I) \cdot (2j)^{e_x}$ can be normalized with regard either to the real component X_R or the imaginary component X_I, depending on which has larger absolute value. The exponent e_x is shared between the real and imaginary component. Exponent overflow/underflow can be handled by setting an exception flag, and allowing processing of results (although erroneous) to continue.

A $RCNS_{2j,3}$ fraction x is considered normalized if $2^{-1} \leq \max(|X_R|, |X_I|) < 1$. The output of a complex number operation can be undernormalized for several reasons:

1. The range of an output determined by the on-line algorithm allows it to be undernormalized.
2. Digit cancellation resulting from the addition/subtraction of numbers with the same exponent value.

In this paper, we assume operands of an $RCNS_{2j,3}$ on-line algorithm have non-zero most significant digits and are normalized. When the result Z exceeds the range of a normalized fraction (i.e. $\max(|Z_R, Z_I|) \geq 1$) then the exponent is incremented. When the result is below the range of a normalized fraction (i.e. $\max(|Z_R, Z_I| < \frac{1}{2})$, then the exponent is decremented and leading zeros are discarded. The normalization algorithm which takes as input the generated output digit z_k, the output exponent e_z and the on-line delay for the arithmetic operation δ is shown below. This is similar to the normalization algorithm presented in [2] for radix-2 on-line rotation.

NORM(z_k, e_z, δ)

/* Initialization */
$done = 0$

/* Computation */
if $k = (\delta - 2)$ and $z_k \neq 0$ then
$e_z = e_z + 2$
$done = 1$
if $k = (\delta - 1)$ and $z_k \neq 0$ and not($done$) then
$e_z = e_z + 1$
$done = 1$
else if $k \geq \delta$ and $z_k = 0$ and not($done$) then
$e_z = e_z - 1$
else if ($k \geq \delta$ and $z_k \neq 0$) then
$done = 1$
end if

III. Recoding algorithms

Although $RCNS_{2j,3}$ allows flexibility in representation, there are also several drawbacks:

- Handling digits 3 and -3 requires producing significand multiples $3X$ and $-3X$, requiring an extra addition step.
- A significand X with fractional real and imaginary components X_R and X_I can have integer digits, such as $(11.32\overline{12})_{2j} = \frac{3}{8} + \frac{3}{8}j$, which can complicate ensuring complex significands within the range $\max(|X_R|, |X_I|) < 1$.

To handle these cases, several recoding modules are presented: (i) digit-set recoding; and (ii) most-significant-digit recoding.

A. Digit-set recoding

In order to reduce the complexity introduced by handling digits -3 and 3, digit-set recoding initially recodes a $RCNS_{2j,3}$ digit $x_k \in \{-3, \ldots, 3\}$ into a pair of digits (t_{k-2}, w_k), in which $t_{k-2} \in \{-1, 0, 1\}$ and $w_k \in \{-2, \ldots, 2\}$ such that $x_k = -4t_{k-2} + w_k$. Then a $RCNS_{2j,2}$ digit χ_k is computed as $\chi_k = t_k + w_k$. In order to restrict $\chi_k \in \{-2, \ldots, 2\}$, two cases of pairs of values must be prevented: (i) $t_k = 1$, $w_k = 2$, (ii) $t_k = -1$, $w_k = -2$. To do so, x_{k+2} is examined. If $x_{k+2} \leq -2$ and $x_k = 2$, which could allow the first case, x_k is recoded as $(\overline{1}, \overline{2})$, otherwise as $(0, 2)$. In the same way, if $x_{k+2} \geq 2$ and $x_k = -2$, which could allow the second case, x_k is recoded as $(1, 2)$, otherwise as $(0, \overline{2})$ Then it is assured that $\chi_k \in \{-2, \ldots, 2\}$. The digit-set recoding algorithm DSREC is shown below.

DSREC(x_k, x_{k+2})

$$(t_{k-2}, w_k) = \begin{cases} (1,1) & \text{if } x_k = \overline{3} \\ (1,2) & \text{if } x_k = \overline{2} \text{ and } x_{k+2} \geq 2 \\ (0,\overline{2}) & \text{if } x_k = \overline{2} \text{ and } x_{k+2} < 2 \\ (0,\overline{1}) & \text{if } x_k = \overline{1} \\ (0,0) & \text{if } x_k = 0 \\ (0,1) & \text{if } x_k = 1 \\ (0,2) & \text{if } x_k = 2 \text{ and } x_{k+2} > -2 \\ (\overline{1},\overline{2}) & \text{if } x_k = 2 \text{ and } x_{k+2} \leq -2 \\ (\overline{1},\overline{1}) & \text{if } x_k = 3 \end{cases}$$

$\chi_k = t_k + w_k$

B. Most-significant-digit recoding

In order to handle carries produced when performing operations on significands consisting of $RCNS_{2j,3}$ digits, most-significant-digit recoding recodes most-significant residual digits $w_{-1}, w_0 \in \{-1, 0, 1\}$ of respective weights $(2j)^1 = 2j$ and $(2j)^0 = 1$, and digits $w_1, w_2 \in \{-3, \ldots, 3\}$, of respective weights $(2j)^{-1}$ and $(2j)^{-2}$, into digits $\omega_1, \omega_2 \in \{-3, \ldots, 3\}$ of respective weights $(2j)^{-1}$ and $(2j)^{-2}$. The algorithm MSREC for recoding general digits w_{k-2} and w_k into digit ω_k is shown below.

MSREC(w_{k-2}, w_k)

$$\omega_k = \begin{cases} \overline{3} & \text{if } (w_{k-2}=0 \text{ and } w_k=\overline{3}) \text{ or } (w_{k-2}=1 \text{ and } w_k=1) \\ \overline{2} & \text{if } (w_{k-2}=0 \text{ and } w_k=\overline{2}) \text{ or } (w_{k-2}=1 \text{ and } w_k=2) \\ \overline{1} & \text{if } (w_{k-2}=0 \text{ and } w_k=\overline{1}) \text{ or } (w_{k-2}=1 \text{ and } w_k=3) \\ 0 & \text{if } w_{k-2}=0 \text{ and } w_k=0 \\ 1 & \text{if } (w_{k-2}=0 \text{ and } w_k=1) \text{ or } (w_{k-2}=\overline{1} \text{ and } w_k=\overline{3}) \\ 2 & \text{if } (w_{k-2}=0 \text{ and } w_k=2) \text{ or } (w_{k-2}=\overline{1} \text{ and } w_k=\overline{2}) \\ 3 & \text{if } (w_{k-2}=0 \text{ and } w_k=3) \text{ or } (w_{k-2}=\overline{1} \text{ and } w_k=\overline{1}) \end{cases}$$

IV. $RCNS_{2j,3}$ ON-LINE FLOATING-POINT ADDITION

$RCNS_{2j,3}$ floating-point addition ($z = x + y$) is defined such that given inputs $x = (X_R + jX_I) \cdot (2j)^{e_x}$ and $y = (Y_R + jY_I) \cdot (2j)^{e_y}$, the output $z = (Z_R + jZ_I) \cdot (2j)^{e_z}$ is produced such that

$$\begin{aligned} Z_R &= X_R + Y_R \\ Z_I &= X_I + Y_I \\ e_z &= \max(e_x, e_y) \end{aligned} \tag{2}$$

Each output digit at step k, namely z_k is generated based on input digits $x_{k+\delta-1}$ and $y_{k+\delta-1}$. The algorithm is shown below, where $\overline{W_E[k]}$ is the low-precision estimate of the even-indexed (real) component of the recurrence $W[k]$. The design of a m-digit significand and e-bit exponent $RCNS_{2j,3}$ on-line floating point adder is shown in Figure 3. The SUBE unit computes the difference of the exponents. The ALIGN unit performs alignment of operand y' to synchronize the arrival of the input digits. The SWAP unit exchanges the operands if necessary. The PPM and MMP modules are simple full-adders that appropriately negate (indicated by "-" on the port) inputs and outputs to perform borrow-save addition. The NORM unit normalizes the result by updating the output exponent z_k. A summary of cost of individual modules is shown in Table I. The design requires $3m+4e+4$ CLB slices. Assuming $m = 24$ and $e = 8$, the cost is 108 CLB slices.

TABLE I
COST OF $RCNS_{2j,3}$ ON-LINE FLOATING-POINT ADDER

Module	CLB slices
SUBE	e
ALIGN	$3m$
SWAP	e
PPM/MMP	4
NORM	$2e$
Total cost	$3m + 4e + 4$

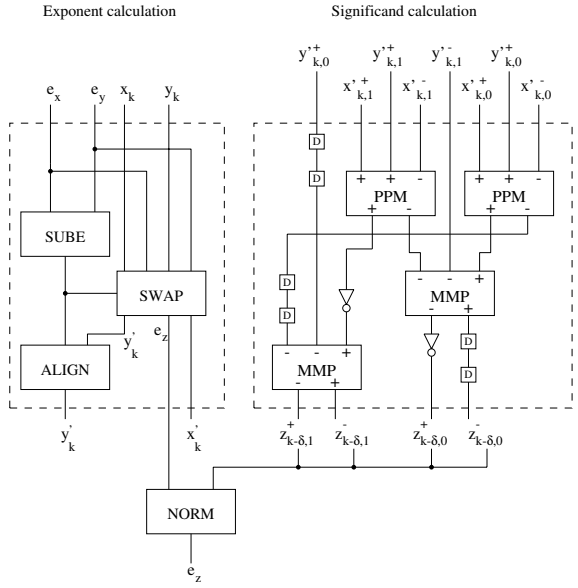

Fig. 3. $RCNS_{2j,3}$ on-line floating-point adder

$RCNS_{2j,3}$ On-line Floating-Point Addition

/* Initialization */
$e_d = e_x - e_y$
e_z=max(e_x, e_y)
$W[-\delta + 1] = 0$
$z_0 = 0$
for $k = -\delta + 2$ to 0 do

$$(x'_{k+\delta-1}, y'_{k+\delta-1}) = \begin{cases} (0, y_{k+\delta-1}) \text{ if } e_d < 0 \\ (x_{k+\delta-1}, 0) \text{ if } e_d > 0 \\ (x_{k+\delta-1}, y_{k+\delta-1}) \text{ if } e_d = 0 \end{cases}$$

$$W[k] = 2j(W[k-1]) + (2j)^{-\delta+1}(x'_{k+\delta-1} + y'_{k+\delta-1})$$

end for

/* Recurrence */
for $k = 1$ to m do

$$(x'_{k+\delta-1}, y'_{k+\delta-1}) = \begin{cases} (0, y_{k+\delta-1}) \\ \quad \text{if } k \le |e_d| \text{ and } e_d < 0 \\ (x_{k+\delta-1}, 0) \\ \quad \text{if } k \le |e_d| \text{ and } e_d \ge 0 \\ (x_{k+\delta-1-|e_d|}, y_{k+\delta-1}) \\ \quad \text{if } k > |e_d| \text{ and } e_d < 0 \\ (x_{k+\delta-1}, y_{k+\delta-1-|e_d|}) \\ \quad \text{if } k > |e_d| \text{ and } e_d \ge 0 \end{cases}$$

$$W[k] = 2j(W[k-1] - z_{k-1}) + (2j)^{-\delta+1}(x'_{k+\delta-1} + y'_{k+\delta-1})$$

$$z_k = \lceil \overline{W_E[k]} + \tfrac{1}{2} \rceil)$$

$$e_z = NORM(z_k, e_z, \delta)$$

end for

V. $RCNS_{2j,3}$ ON-LINE FLOATING-POINT CONSTANT COEFFICIENT MULTIPLICATION

$RCNS_{2j,3}$ floating-point coefficient multiplication ($z = xy$) is defined such that given constant coefficient parallel input $x = (X_R + jX_I) \cdot (2j)^{e_x}$ and variable input $y = (Y_R + jY_I) \cdot (2j)^{e_y}$, the output $z = (Z_R + jZ_I) \cdot (2j)^{e_z}$ is produced such that

$$\begin{aligned} Z_R &= X_R Y_R - X_I Y_I \\ Z_I &= X_R Y_I + X_I Y_R \\ e_z &= e_x + e_y \end{aligned} \tag{3}$$

Each output digit at step k, namely z_k is generated based on parallel input vector X and input digit $y_{k+\delta-1}$. The algorithm is shown below, where $\overline{W_E[k]}$ is the low-precision estimate of the even-indexed (real) component of the recurrence $W[k]$. The design of a m-digit significand and e-bit exponent $RCNS_{2j,3}$ on-line floating point constant coefficient multiplier is shown in Figure 4. The ADDER unit computes the sum of the exponents. The digit-vector multiplier computes the product Xy_k at each iteration. The borrow-save adder computes the sum W_k of the previous residual W_{k-1} and the intermediate product Xy_k, and stores the result in the register REG W. The NORM unit normalizes the result based on the current output exponent e_z, the on-line delay δ, and the output digit z_k. A summary of cost of individual modules is shown in Table II. The design requires $8m + 3e + 32$ CLB slices. Assuming $m = 24$ and $e = 8$, the cost is 248 CLB slices.

TABLE II
COST OF $RCNS_{2j,3}$ ON-LINE FLOATING-POINT CONSTANT COEFFICIENT MULTIPLIER

Module	CLB slices
Adder	e
DSREC	12
Digit-vector multiplier	$4m$
Borrow-save adder	$4m$
NORM	$2e$
MSREC	20
Total cost	$8m + 3e + 32$

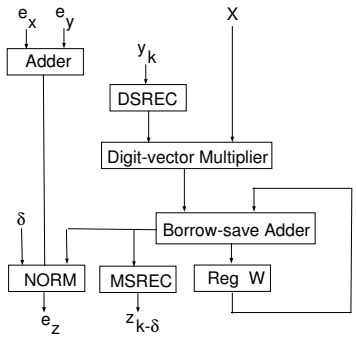

Fig. 4. $RCNS_{2j,3}$ on-line floating-point constant coefficient multiplier

$RCNS_{2j,3}$ On-line Floating-Point Constant Coefficient Multiplication

/* Initialization */
$e_z = e_x + e_y$
$W[-\delta + 1] = 0$
$Y[-\delta + 1] = 0$
$z_0 = 0$

for $k = -\delta + 2$ to 0 do
$W[k] = (2j)(W[k-1]) + (2j)^{-\delta+1}(Xy_{k+\delta-1})$
$Y[k] = Y[k-1] + y_{k+\delta-1}(2j)^{-k-\delta+1}$
end for

/* Recurrence */
for $k = 1$ to m do
$W[k] = (2j)(W[k-1] - z_{k-1}) + (2j)^{-\delta+1}(Xy_{k+\delta-1})$
$z_k = \lceil \overline{W_E[k] + \frac{1}{2}} \rceil)$
$Y[k] = Y[k-1] + y_{k+\delta-1}(2j)^{-k-\delta+1}$
$e_z = NORM(z_k, e_z, \delta)$
end for

VI. IMPLEMENTATION

Each tap (slice) of the FIR filter, assuming 24-digit significand and 8-bit exponent floating-point operands, consists of 24 digit-wide registers to store individual digits of inputs $x(n), x(n-1), \ldots, x(n-N-1)$ and a complex number on-line floating point constant coefficient multiplier. Each multiplier product is fed to one of the operands of a complex number on-line floating point adder. The parallel adder in Figure 1 can be implemented as a binary tree of complex number on-line floating point adders, each one initially adding two intermediate multiplier outputs and producing an intermediate sum output, until the final output $y(n)$ is computed.

A. Radix 2j on-line network

An N-tap complex FIR filter can be designed as a network of radix-2j on-line floating-point arithmetic operators, where, assuming m-digit significands and e-bit exponents, a radix 2j on-line floating-point adder has a cost of $3m + 4e + 4$ CLB slices, and a radix 2 on-line floating-point multiplier has a cost of $8m + 8e + 32$ CLB slices. For $m = 24$ and $e = 8$, the cost of a radix 2j on-line floating-point adder is 108 CLB slices and the cost of a radix 2j on-line floating-point multiplier is 248 CLB slices. Since for an N-tap complex FIR filter, $N - 1$ radix 2j floating-point adders and N radix 2j floating-point multipliers are used, then the cost is $356N - 108$ CLB slices.

B. Radix 2 on-line network

An N-tap complex FIR filter can be alternatively designed as a network of radix-2 on-line floating-point arithmetic operators, where, assuming m-digit significands and e-bit

exponents, a radix 2 on-line floating-point adder has a cost of $1.5m + 3e + 2$ CLB slices, and a radix 2 on-line floating-point multiplier has a cost of $3m + 3e + 2$ CLB slices. For $m = 24$ and $e = 8$, the cost of a radix 2 on-line floating-point adder is 70 CLB slices and the cost of a radix 2 on-line floating-point multiplier is 98 CLB slices. Since for an N-tap complex FIR filter, $3N - 1$ radix 2 floating-point adders and $4N$ radix 2 floating-point multipliers are used, then the cost is $602N - 70$ CLB slices.

C. Radix 2 parallel network

An N-tap complex FIR filter can be alternatively design as a network of radix-2 parallel arithmetic operators. The library of Xilinx CORE arithmetic modules [6], which can be scaled in terms of precision is used. Since the modules are defined for fixed-point arithmetic, appropriate exponent handling units are used to support floating-point arithmetic. For 24-bit significands and 8-bit exponents, the cost of a radix 2 parallel floating-point adder is 30 CLB slices and the cost of a radix 2 parallel floating-point multiplier is 320 CLB slices. Since for an N-tap complex FIR filter, $3N-1$ radix 2 floating-point adders and $4N$ radix 2 floating-point multipliers are used, then the cost is $1370N - 30$ CLB slices.

D. Cost comparison

The cost of the proposed radix 2j on-line network, and the alternative radix 2 on-line network and the radix 2 parallel network are compared for the implementation of an N-tap complex FIR filter for common values of N, including N=8,16,64, and 256. In each case, we assume floating-point operands consisting of 24-digit (or bit) significands and 8-bit exponents, as shown in Table III.

TABLE III
COMPARISON OF COSTS FOR FLOATING-POINT N-TAP COMPLEX FIR FILTER ($m = 24$, $e = 8$)

N	$RCNS_{2j,3}$ on-line	Radix-2 on-line	Radix-2 parallel
8	2740	4746	10930
16	5588	9562	21890
64	22676	38458	87650
256	91028	154042	350690

VII. CONCLUSION

We have demonstrated a new approach for implementating an N-tap complex FIR filter, based on using complex number on-line arithmetic modules which adopt a redundant complex number system (RCNS) for efficient representation. Significant improvement in cost in comparison to a radix-2 on-line approach and a radix-2 parallel approach have been shown. This offers motivation for further research into other applications utilizing complex number operations.

REFERENCES

[1] T. Aoki, Y. Ohi, and T. Higuchi, "Redundant complex number arithmetic for high-speed signal processing," *1995 IEEE Workshop on VLSI Signal Processing*, Oct. 1995, pp. 523-532.
[2] M.D. Ercegovac and T. Land, "On-line scheme for computing rotation factors," *Journal of paralle and distributed computing*, 1988. pp. 209-227.
[3] M.D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufmann Publishers, 2004.
[4] D.E. Knuth, "The art of computer programming," Vol. 2, 1973.
[5] R. McIlhenny, "Complex number on-line arithmetic for reconfigurable hardware: algorithms, implementations, and applications," *Ph.D. Dissertation, University of California, Los Angeles*, 2002.
[6] Xilinx Corporation, "Xilinx Data Book," 2004.