On the Design of an On-line Complex Householder Transform

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 an implementation for the Complex Householder Transform, 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 comparisons with (i) a real number on-line arithmetic approach, and (ii) a real number arithmetic parallel approach, to demonstrate a significant improvement in cost.**

I. Introduction

The Householder Transform [4] is an important operation in numerous signal processing applications, including QR decomption and array processing [7]. When the elements of the matrix are complex numbers, it is denoted the Complex Householder Transform (CHT) [2]. The CHT is applied to a column vector $\underline{x}$ to zero out all the elements except the first one. Given a column vector $\underline{x} = [x_1, \ldots, x_k]^T \in C^k$, where $x_1 = |x_1|e^{j\theta}$ with $\theta \in R$, the basic steps of the CHT algorithm are: (i) define a column vector $\underline{u} = \underline{x} + e^{j\theta}||x||_2\underline{e_1}$, where $\underline{e_1}^T = [1, 0, \ldots, 0]$; (ii) define the $k \times k$ CHT B as:

$$B = I - \frac{2}{\underline{u}^H\underline{u}}\underline{u}\underline{u}^H \tag{1}$$

(iii) apply the matrix B to the vector $\underline{x}$ to produce a column vector $y = B\underline{x}$, in which all elements are zeroed out except for the first element, i.e.

$$\begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1k} \\ B_{21} & B_{22} & \cdots & B_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{k1} & B_{k2} & \cdots & B_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} y_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2}$$

In these steps, I is the $k \times k$ identity matrix and $\underline{u}^H$ is the complex conjugate transpose of $\underline{u}$. The parameter $e^{j\theta} = \frac{x_1}{\sqrt{x_1 x_1^*}}$, where x_1^* is the complex conjugate of x_1. The parameter $||x||_2$ is the norm of x of degree two, in which $||x||_2 = \sqrt{x_1x_1^* + \ldots + x_kx_k^*}$. The product u^Hu produces a single real value in which $u^Hu = u_1u_1^* + u_2u_2^* + \ldots + u_ku_k^*$, since u is a column vector and u^H is a row vector. The product uu^H is a $k \times k$ matrix V, in which each real element $v_{ij} = u_iu_j^*$.

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, which is gnerally one cycle after the result is computed. Each successive digit of the result is generated one per cycle. This is illustrated in Figure 1, with on-line delay $\delta = 4$. The latency of an on-line arithmetic operator, assuming m-digit precision is then $\delta + m - 1$.

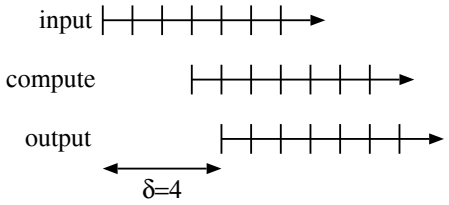

Fig. 1. On-line delay of a function

Complex number on-line arithmetic [6] 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 is 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 [5]. At the binary level, the digits will be represented using borrow-save encoding, in which each digit $x_k \in \{-3,-2,-1,0,1,2,3\}$ is represented as 4 bits $(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0})$ such that $x_k = 2x^+_{k,1} - 2x^-_{k,1} + x^+_{k,0} - x^-_{k,0}$. This is an extension of radix 2 borrow-save encoding, in which each digit $x_k \in \{-1,0,1\}$ is represented as 2 bits (x^+_k, x^-_k) such that $x_k = x^+_k - x^-_k$.

The throughput of a radix $2j$ on-line arithmetic operator is the same as for the radix 2 implementation of a complex arithmetic operator. A radix $2j$ on-line arithmetic operator generates real and imaginary digits in alternate cycles, with each radix $2j$ digit corresponding to two radix 2 digits. The equivalent radix 2 implementation of a complex on-line arithmetic operator generates real and imaginary digits in the same cycle. This is demonstrated below for the output of two radix $2j$ borrow-save encoded digits $z_1 = (z^+_{1,1}, z^-_{1,1}, z^+_{1,0}, z^-_{1,0})$ and $z_2 = (z^+_{2,1}, z^-_{2,1}, z^+_{2,0}, z^-_{2,0})$, and four radix 2 borrow-save encoded digits, consisting of real digits $z_{R,1} = (z^+_{R,1}, z^-_{R,1})$ and $z_{R,2} = (z^+_{R,2}, z^-_{R,2})$ and imaginary digits $z_{I,1} = (z^+_{I,1}, z^-_{I,1})$ and $z_{I,2} = (z^+_{I,2}, z^-_{I,2})$.

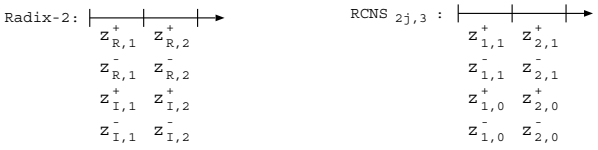

Fig. 2. Throughput of radix 2 and radix $2j$ digits

Since the throughputs are equal (differing only in respective on-line delays), a radix $2j$ output can be converted to a radix 2 format for input to radix 2 on-line arithmetic operators, and vice versa, at the digit level. The algorithm below demonstrates the conversion from a radix $2j$ borrow-save digit $z_k = (z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0})$ to equivalent individual real and imaginary radix 2 borrow-save digits $z_{R,k} = (z^+_{R,k}, z^-_{R,k})$ and $z_{I,k} = (z^+_{I,k}, z^-_{I,k})$.

Radix $2j$ to Radix 2 Borrow-save Conversion

if k mod 4=0 then
$(z^+_{R,k}, z^-_{R,k}) = (z^+_{k,0}, z^-_{k,0})$
$(z^+_{I,k}, z^-_{I,k}) = (\overline{z^+_{k+1,0}}, \overline{z^-_{k+1,0}})$
else if k mod 4=1 then
$(z^+_{R,k}, z^-_{R,k}) = (\overline{z^+_{k+1,0}}, \overline{z^-_{k+1,0}})$
$(z^+_{I,k}, z^-_{I,k}) = (\overline{z^+_{k,0}}, \overline{z^-_{k,0}})$
else if k mod 4=2 then
$(z^+_{R,k}, z^-_{R,k}) = (\overline{z^+_{k,0}}, \overline{z^-_{k,0}})$
$(z^+_{I,k}, z^-_{I,k}) = (z^+_{k+1,0}, z^-_{k+1,0})$
else if k mod 4=3 then
$(z^+_{R,k}, z^-_{R,k}) = (z^+_{k+1,0}, z^-_{k+1,0})$
$(z^+_{I,k}, z^-_{I,k}) = (z^+_{k,0}, z^-_{k,0})$

The algorithm below demonstrates conversion from real and imaginary radix 2 borrow-save digits $z_{R,k} = (z^+_{R,k}, z^-_{R,k})$ and $z_{I,k} = (z^+_{I,k}, z^-_{I,k})$ to a radix $2j$ borrow-save digit $z_k = (z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0})$.

Radix 2 to Radix $2j$ Borrow-save Conversion

if k mod 4=0 then
$(z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0}) = (z^+_{R,k-1}, z^-_{R,k-1}, z^+_{R,k}, z^-_{R,k})$
else if k mod 4=1 then
$(z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0}) = (\overline{z^+_{I,k-1}}, \overline{z^-_{I,k-1}}, \overline{z^+_{I,k}}, \overline{z^-_{I,k}})$
else if k mod 4=2 then
$(z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0}) = (\overline{z^+_{R,k-1}}, \overline{z^-_{R,k-1}}, \overline{z^+_{R,k}}, \overline{z^-_{R,k}})$
else if k mod 4=3 then
$(z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0}) = (z^+_{I,k-1}, z^-_{I,k-1}, z^+_{I,k}, z^-_{I,k})$

Using a radix $2j$ representation, 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. A radix $2j$ fraction x is considered normalized if $2^{-1} \leq \max(|X_R|, |X_I|) < 1$. 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.

NORM(z_k, e_z, δ)
$done = 0$
while not($done$) loop
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
end loop

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 $\max(|X_R|, |X_I|) < 1$.

Several recoding algorithms to handle these issues are described, including: (i) digit-set recoding; and (ii) most-significant-digit recoding.

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})

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

$\chi_k = t_k + w_k$

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 next.

MSREC(w_{k-2}, w_k)

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

III. CHT Implementation

The CHT produces a complex column vector y_k in which y_1 is the only non-zero element. Simplifying the computation results in

$$y_1 = -e^{j\theta}||x||_2 = -x_1\sqrt{\frac{x_1x_1^* + \ldots + x_kx_k^*}{x_1x_1^*}} \tag{3}$$

The implementation requires k complex multipliers (CMULT), $k-1$ real adders (RADD), 1 real divider (RDIV), a real square root unit (RSQRT), a unit to negate a digit (NEG), and a complex-real multiplier (CRMULT) as shown in Figure 3. Since the product of a complex-conjugate multiplier is a real number, radix 2j to radix 2 converters will be used to convert the outputs into radix 2 representation for input to the real adders. Likewise, radix 2 to radix 2j converters will be used to convert the output of the real square root unit to radix $2j$ representation for input to the complex-real multiplier, which produces the complex output y_1. The recurrence algorithms and designs of the radix $2j$ on-line floating-point complex-conjugate multiplier and the radix $2j$ on-line floating-point complex-real multiplier are described next.

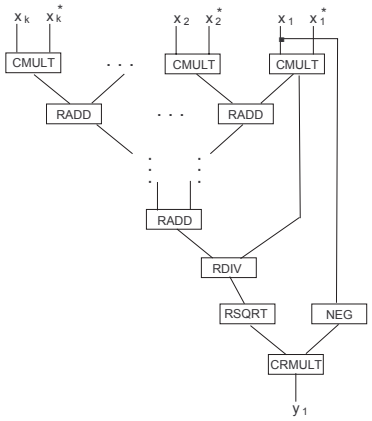

Fig. 3. Computational graph of CHT

A. Radix $2j$ on-line floating-point complex-conjugate multiplication

Radix $2j$ floating-point complex-conjugate multiplication ($z = xx^*$) is defined such that given inputs $x = (X_R + jX_I) \cdot (2j)^{e_x}$ and $x^* = (X_R - jX_I) \cdot (2j)^{e_x}$, the output $z = (Z_R + jZ_I) \cdot (2j)^{e_z}$ is produced such that

$$\begin{aligned} Z_R &= X_R^2 + X_I^2 \\ Z_I &= 0 \\ e_z &= 2e_x \end{aligned} \tag{4}$$

The recurrence formula for radix $2j$ on-line multiplication ($z = xy$) is the following:

$$\begin{aligned} W[k] = \; & (2j)(W[k-1] - z_{\lfloor}k-1]) \\ & +(2j)^{-\delta+1}(X[k]y_{k+\delta-1} + Y[k-1]x_{k+\delta-1}) \\ z_k = \; & \lfloor \overline{W_E[k]} \rfloor \end{aligned} \tag{5}$$

where $\overline{W_E[k]}$ is the low-precision estimate of the even-indexed (real) component of $W[k]$.

For radix $2j$ on-line floating-point complex-conjugate multiplication ($z = xx^*$), since

$$\begin{aligned} x^*_{k+\delta-1} &= \begin{cases} x_{k+\delta-1} & \text{if } k+\delta-1 \text{ is even} \\ -x_{k+\delta-1} & \text{if } k+\delta-1 \text{ is odd} \end{cases} \\ z &= \begin{cases} \lfloor \overline{W_E[k]} \rfloor & \text{if } k+\delta-1 \text{ is even} \\ 0 & \text{if } k+\delta-1 \text{ is odd} \end{cases} \end{aligned} \tag{6}$$

the recurrence can be rewritten as

$$W[k] = \begin{cases} (2j)(W[k-1]) + (2j)^{-\delta+1}((2X_R[k-1] \\ +x_{k+\delta-1}(2j)^{-k-\delta+1})x_{k+\delta-1}) \\ \text{if } k+\delta-1 \text{ is even} \\ \\ (2j)(W[k-1]-z_{k-1}) \\ +(2j)^{-\delta+1}(-2X_I[k-1] \\ -x_{k+\delta-1}(2j)^{-k-\delta+1})x_{k+\delta-1}) \\ \text{if } k+\delta-1 \text{ is odd} \end{cases} \tag{7}$$

For the implementation, two types of modular slices are required. An odd-indexed slice M_{2k-1} ($k = 1$ to $\lceil m/2 \rceil$) consists of one borrow-save digit multiplier, a 2:1 borrow-save digit adder, a digit-wide latch, a D flip-flop, a bit-wide D flip-flop, a digit-wide D flip-flop, a TWICE unit for computing $2X[k-1]$, a 3-to-1 MUX for appropriately appending digit x_k to vector $2X[k-1]$, and a 2-to-1 MUX for appropriately shifting the residual. An even-indexed slice M_{2k} ($k = 1$ to $\lfloor m/2 \rfloor$) consists of a digit-wide latch, a bit-wide D flip-flop, and a TWICE unit for computing $2X[k-1]$. A flag bit eo controls switching between odd-indexed and even-indexed slices ($eo = 1$ for an odd-indexed slice, and $eo = 0$ for an even-indexed slice). The CONJ unit generates digits of x^*. The digit x_k is recoded into digit set $\{-2, \ldots, 2\}$ using one DSREC unit. The most-significant digit of the recurrence is determined using one MSREC unit, which performs output digit selection as well as handles most significant carry-out bits of the adder. The MULT2E unit multiplies the operand exponent by two to produce the output exponent e_z. The NORM unit normalizes the result by updating the output exponent e_z. The design of a m-digit significand and e-bit exponent radix $2j$ on-line floating-point complex-conjugate multiplier unit is shown in Figure 4. The number of individual module types utilized, the cost per module type, and the total overall cost are summarized in Table I. Assuming $m = 24$ and $e = 8$, the cost is 224 CLB slices. The on-line delay is $\delta = 9$.

B. Radix $2j$ on-line floating point complex-real multiplication

Radix $2j$ floating-point complex-real multiplication ($z = xy$) is defined such that given complex input $x = (X_R + jX_I) \cdot (2j)^{e_x}$ and real input $y = Y \cdot (2)^{e_y}$, the output $z = (Z_R + jZ_I) \cdot (2j)^{e_z}$ is produced such that

$$\begin{aligned} Z_R &= X_R Y \\ Z_I &= X_I Y \\ e_z &= e_x + e_y \end{aligned} \tag{8}$$

TABLE I

COST OF RADIX $2j$ ON-LINE FLOATING-POINT COMPLEX-CONJUGATE MULTIPLIER

Module	Count	CLB slices
MULT2E	1	e
CONJ	1	4
DSREC	1	12
3-to-1 MUX	$\frac{m}{2}$	$2.25m$
2-to-1 MUX	$\frac{m}{2}$	m
BSD mult. ($\otimes$)	$\frac{m}{2}$	$2m$
BSD adder (2:1)	$\frac{m}{2}$	$2m$
MSREC	1	10
NORM	1	$2e$
Total cost		$\lceil 7.25m + 3e + 26 \rceil$

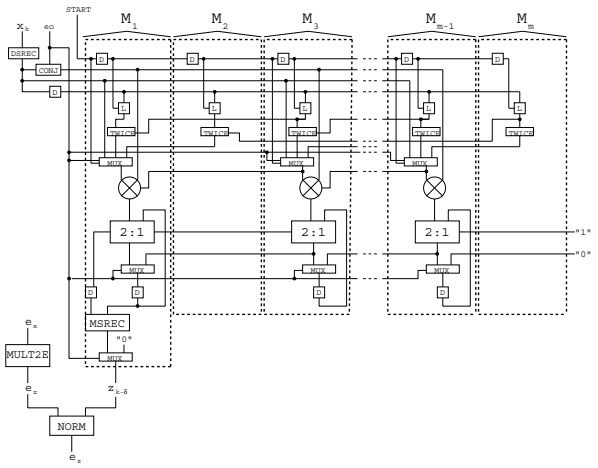

Fig. 4. Radix $2j$ on-line floating-point complex-conjugate multiplier

For radix $2j$ on-line floating-point complex-real multiplication, the recurrence from Equation 5 can be rewritten as:

$$W[k] = \begin{cases} (2j)(W[k-1]-z_{k-1}) \\ +(2j)^{-\delta+1}(X[k]y_{k+\delta-1} + Y[k-1]x_{k+\delta-1}) \\ \text{if } k+\delta-1 \text{ is even} \\ \\ (2j)(W[k-1]-z_{k-1}) \\ +(2j)^{-\delta+1}(Y[k-1]x_{k+\delta-1}) \\ \text{if } k+\delta-1 \text{ is odd} \end{cases} \tag{9}$$

For the implementation two types of modular slices are required. An odd-indexed slice M_{2k-1} ($k = 1$ to $\lceil m/2 \rceil$) consists of one borrow-save digit multiplier, a 2:1 borrow-save digit adder, a digit-wide latch, a bit-wide D flip-flop, and a digit-wide D flip-flop. An even-indexed slice M_{2k} ($k = 1$ to $\lfloor m/2 \rfloor$) consists of two borrow-save digit multipliers, a 3:1 borrow-save digit adder, two digit-wide latches, a bit-wide D flip-flop, and a digit-wide D flip-flop. The digits x_k and y_k are recoded into the digit set $\{-2, \ldots, 2\}$ using two DSREC units. The most-significant digits of the recurrence are determined using two MSREC units, which perform output digit selection

as well as handling the most significant carry-out bits of the adder. The ADDE unit adds the operand exponents to produce the output exponent e_z. The NORM unit normalizes the result by updating the output exponent e_z. The design of a m-digit significand and e-bit exponent radix $2j$ on-line floating-point complex-real multiplier unit is shown in Figure 5. The number of individual module types utilized, the cost per module type, and the total overall cost are summarized in Table II. Assuming $m = 24$ and $e = 8$, the cost is 356 CLB slices. The on-line delay is $\delta = 9$.

TABLE II

COST OF RADIX $2j$ ON-LINE FLOATING-POINT COMPLEX-REAL MULTIPLIER

Module	Count	CLB slices
ADDE	1	e
DSREC	2	24
BSD mult. ($\otimes$)	$\frac{3m}{2}$	$6m$
BSD adder (3:1)	$\frac{m}{2}$	$4m$
BSD adder (2:1)	$\frac{m}{2}$	$2m$
MSREC	2	20
NORM	1	$2e$
Total cost		$12m + 3e + 44$

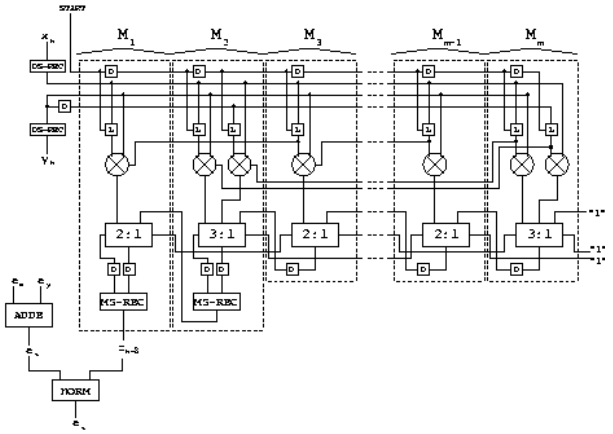

Fig. 5. Radix $2j$ on-line floating-point complex-real multiplier

Three approaches for the design of the CHT are compared: (i) a radix $2j$ approach which uses a combination of radix $2j$ on-line arithmetic modules for complex inputs and radix 2 on-line arithmetic modules for real inputs; (ii) a radix 2 approach which strictly uses radix 2 on-line arithmetic modules; and (iii) a radix 2 parallel approach which uses the Xilinx library of floating-point parallel arithmetic operators [8]. The results in terms of cost and latency are shown next.

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 CHT unit which operates on a k-digit vector $\underline{x}$, for various values of k. 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 CLB COSTS FOR CHT

k	Radix $2j$ on-line	Radix 2 on-line	Radix 2 parallel
32	9820	13468	32693
64	19228	26524	64117
128	38044	52636	126965
256	75676	104860	252661

The delay 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 CHT unit for various values of k. In each case, we assume floating-point operands consisting of 24-digit (or bit) significands and 8-bit exponents, as shown in Table IV.

TABLE IV

COMPARISON OF CYCLE LATENCIES FOR CHT

k	Radix $2j$ on-line	Radix 2 on-line	Radix 2 parallel
32	60	58	126
64	63	61	137
12	66	64	148
256	69	67	159

IV. CONCLUSION

We have demonstrated a new approach for the implementation of a CHT unit, 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, as well as a significant reduction in latency in comparison to a radix 2 parallel approach have been shown. This motivates the research into other application to utilize complex on-line arithmetic.

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] K-L Chung and W-M Yan, "The complex householder transform," *IEEE transactions on signal processing*, Vol. 45, no. 9, Sept. 1997, pp. 2374-2376.

[3] M.D. Ercegovac and T. Lang, "Digital arithmetic," Morgan Kaufmann Publishers, 2004.

[4] A.S. Householder, "Unitary triangularization of a nonsymmetric matrix", *Journal of the ACM*, 5 (4), 158, pp. 339-342.

[5] D.E. Knuth, "The art of computer programming," Vol. 2, 1973.

[6] R. McIlhenny, "Complex number on-line arithmetic for reconfigurable hardware: algorithms, implementations, and applications," *Ph.D. Dissertation, University of California, Los Angeles*, 2002.

[7] C.F.T. Tang, K.J.R Liu, S.F. Hsieh, and K. Yao, "VLSI algorithms and architectures for complex householder transformation with applications to array processing," *University of Maryland, College Park, Technical Report TR 91-84*, 1991, pp. 1-36.

[8] Xilinx Corporation, "Xilinx Data Book," 2004.