Design and Implementation of a Radix-4 Complex Division Unit with Prescaling

Pouya Dormiani, Milos Ercegovac, Jean-Michel Muller

Design and Implementation of a Radix-4 Complex Division Unit with Prescaling

Pouya Dormiani
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90024, USA
Email: pouya@cs.ucla.edu

Miloš D. Ercegovac
4731H Boelter Hall
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90024, USA
Email: milos@cs.ucla.edu

Jean-Michel Muller
CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France
Email: Jean-Michel.Muller@ens-lyon.fr

***Abstract*—We present a design and implementation of a radix-4 complex division unit with prescaling of the operands. Specifically, we extend the treatment of the residual bound and errors due to the use of truncated redundant representation. The requirements for prescaling tables are simplified and a detailed specification of the table design is given. All principal components used in the design are described and the proposed optimizations are explained. The target platform for implementation was an Altera Stratix II FPGA [15] for which we report timing and area requirements. For a precision of 36 bits, the implementation uses 1093 ALUTs, achieving a latency of 97ns. The maximum clock frequency is 268.53 MHz.**

I. Introduction

Complex division is used in applications such as such as signal processing (e.g., the complex SVD), multiantenna systems (MIMO-type) [1], GPS [2], astronomy [3], and nonlinear RF measurement equipments [4]. Unlike for complex multipliers [10], [12], its implementation has been commonly provided in software. To improve its performance, a hardware implementation is considered. With that objective, a hardware-oriented algorithm and the corresponding theory for general radix-r complex valued division based on a digit-recurrence algorithm has been introduced in [6]. A high-level design of a complex divider is discussed in [7] without implementation details. In this paper we focus on the design and implementation of a radix-4 complex-valued division unit with the digit set $\{-3,\ldots,3\}$. The operands and the result are in fractional fixed-point form. We also refine some of the derivation results from [6] to improve the implementation.

Specifically, with the dividend $z = z_R + iz_I$ and divisor $d = d_R + id_I$, $i = \sqrt{-1}$, the design discussed computes $q = z/d$. A high-level description of the algorithm is

Initialization: $j = 0$

$$w[0] = z \tag{1}$$

Recurrence iterations: $j = 1,\ldots,n$

$$q_{j+1} = Sel(4w[j], y) \tag{2}$$

$$w[j+1] = 4w[j] - q_{j+1}y \tag{3}$$

Result:

$$q = \frac{z}{d} = 0.q_1^R q_2^R q_3^R \ldots q_n^R + i0.q_1^I q_2^I q_3^I \ldots q_n^I \tag{4}$$

The recurrence for complex division corresponds to the conventional real-valued division discussed in [5] and similar conditions such as the containment and continuity as well as bounded residuals apply. The complex residual is $w[j] = w^R[j] + iw^I[j]$. The quotient digits are $q_{j+1} = q_{j+1}^R + iq_{j+1}^I$, with the real and imaginary components q_{j+1}^R and $q_{j+1}^I \in \{-3,\ldots,3\}$. These signed-digits can be converted during the iterations using on-the-fly conversion [5] to obtain conventional representation of the result. The complex residual recurrence decomposes into two separate recurrences for the real and imaginary part which can be computed in parallel:

$$w^R[j+1] = 4w^R[j] - q_{j+1}^R d^R + q_{j+1}^I d^I \tag{5}$$

$$w^I[j+1] = 4w^I[j] - q_{j+1}^R d^I - q_{j+1}^I d^R \tag{6}$$

where $w^R[0] = z^R$ and $w^I[0] = z^I$. The quotient-digit selection in the complex domain is a two-dimensional problem because both q_{j+1}^R and q_{j+1}^I must be selected in such a way that the real and imaginary residuals ($w^R[j]$, $w^I[j]$) remain bounded. This is much more difficult than single-digit selection used in the real case. We solve this problem by scaling the operands by factor K such that $Kz/Kd = x/y$ where $y = Kd \approx 1$. Consequently, $y^R \approx 1$ and $y^I \approx 0$, and the selection of q_{j+1}^R and q_{j+1}^I can be performed on the real and the imaginary shifted residuals separately in a manner similar to real-valued division selection. To determine the prescaling factor K, we assume that

$$\|Kd - 1\|_\infty < \epsilon_s \tag{7}$$

where $\|\alpha\|_\infty = \max(|\alpha^R|, |\alpha^I|)$.

After prescaling step, the recurrences are

$$w^R[j+1] = 4w^R[j] - q_{j+1}^R y^R + q_{j+1}^I y^I \tag{8}$$

$$w^I[j+1] = 4w^I[j] - q_{j+1}^R y^I - q_{j+1}^I y^R \tag{9}$$

where $w^R[0] = x^R$ and $w^I[0] = x^I$. Because the scaling makes $y^I \approx 0$ and $y^R \approx 1 - \epsilon_s$ the selection of the real part of the quotient can be performed by rounding the shifted real residual and taking the integer part. Similarly for the selection of the imaginary part of the quotient digit. Moreover, we can use estimates of σ fractional positions of the shifted residuals $4w^R[j]$ and $4w^I[j]$ in the selection. Consequently,

the residuals can be computed in redundant form to keep the cycle time short. The selection functions are

$$q^R_{j+1} = Sel(est(4w^R[j], \sigma)) \tag{10}$$
$$= sign(4w^R[j]) \times \lfloor |est(4w^R[j], \sigma)| + \frac{1}{2} \rfloor$$
$$q^I_{j+1} = Sel(est(4w^I[j], \sigma)) \tag{11}$$
$$= sign(4w^I[j]) \times \lfloor |est(4w^I[j], \sigma)| + \frac{1}{2} \rfloor$$

The selection function Sel satisfies

$$|Sel(est(x, \sigma)) - x| < \frac{1}{2} + 2^{-\sigma} \tag{12}$$

The $est(x, \sigma)$ is x truncated to σ fractional positions with an error bound

$$est_{ERR}(x, \sigma) = |x - est(x, \sigma)| < 2^{-\sigma}$$

If x is in carry-save form $x = x_C + x_S$ then truncating the carry and sum vector to $\sigma + 1$ fractional bits results in the same maximum error committed, i.e. , $est_{ERR}(x, \sigma) < 2^{-\sigma}$ and $est_{ERR}(x_C, \sigma + 1) + est_{ERR}(x_S, \sigma + 1) < 2^{-\sigma}$.

Using (10), (11) and (12), a bound on the residual is deduced which ensures that the digit (q^R_{j+1}, q^I_{j+1}) selected by rounding is in the digit set $\{-3, \ldots, 3\}$. Namely,

$$\| w[j] \|_\infty \leq \frac{1}{4} \left(3 + \frac{1}{2} + 2^{-\sigma} \right) \tag{13}$$

As shown in [6], assuming that the scaling error is ϵ_s and $a = 3$, the residual is bounded by

$$\|w[j]\|_\infty < 2 \times 3 \times \epsilon_s + \frac{1}{2} + 2^{-\sigma} \tag{14}$$

Consequently,

$$6\epsilon_s + \frac{1}{2} + 2^{-\sigma} \leq \frac{1}{4} \left(3 + \frac{1}{2} + 2^{-\sigma} \right) \tag{15}$$

Satisfying this condition guarantees convergence of the digit-recurrence algorithm and allows the choice of ϵ_s and σ to optimize the implementation characteristics.

II. Design

The design of the complex division unit consists of several components: the prescaling module, the recurrence modules for the real and imaginary parts, the on-the-fly converters to obtain conventional representations, in addition to a simple controller. A high level block diagram of the design is shown in Fig. 1 with the timing shown in Fig. 2. The prescaling module in Fig. 1 performs a ROM look-up using a short estimate of the value of the divisor d as an address, in which the ROM stores $K = 1/d$. It then computes the complex product Kz, which is used to initialize $w[0]$ in the recurrence modules. The prescaling module computes Kd in parallel to the initialization of the recurrence modules, which is then used to perform the iterations of the recurrence. The initial delay of the module to perform prescaling can be amortized

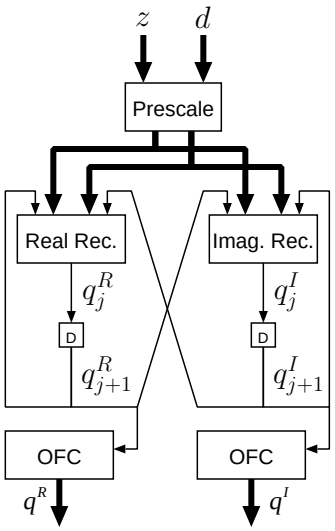

Fig. 1. High-level block diagram of the complex division unit.

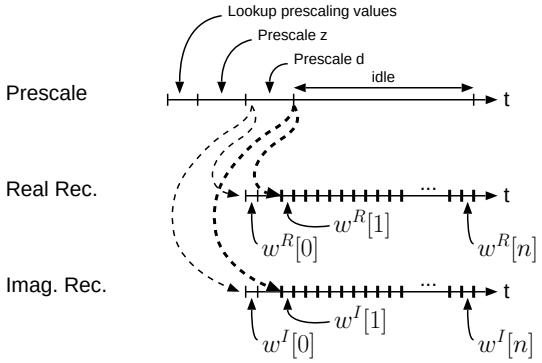

Fig. 2. Timing relationships between modules.

by overlapping the prescaling of the next operation with digit-recurrence iterations of the current operation–this however has not been performed in the current implementation. Detailed design of the prescaling is discussed in Section II-A.

The two recurrence modules (one for the real recurrence and one for the imaginary) perform nearly identical operations which can be mapped to the same hardware. Detailed design of the recurrence module is discussed in Section II-C.

A. Prescaling

Prescaling consists of several steps: obtaining the factor K from a table based on an short-precision estimate of d, and computing Kz and Kd.

We define a function $rnd(a, b)$ which returns a rounded value of a to b fractional places, s.t. $|a - rnd(a, b)| \leq \frac{1}{2} 2^{-b}$. The factor K can be determined by using a short estimate of d to q fractional positions, i.e., $rnd(d^R, q)$, $rnd(d^I, q)$ as an address to a ROM which stores the corresponding values of K with precision of t fractional positions,

$$K^R = rnd(1/rnd(d^R, q), t)$$
$$K^I = rnd(1/rnd(d^I, q), t)$$

Error analysis for the choices of parameters q and t is performed in [13]. These effect ϵ_s used in (15) to guarantee convergence of the algorithm. Radix 4, with digit set $\{-3, \ldots, 3\}$ offers the most favorable choice of parameters by minimizing the number of bits required for the ROM among radices 4, 8, and 16, except radix 2 which has lowest memory requirements. Over-redundant digit sets are another design choice but we decided to restrict our design to maximally redundant digit set which allows faithful rounding [6].

r	a	σ	q	t	KBits (approx.)
2	1	4	5	12	16
4	2	5	7	14	256
4	3	4	6	13	64
Radix: 8, 16 see [13]					≥ 256

TABLE I
TODO: UPDATE TO NEW RESULTS DESIGN SPACE FOR MEMORY REQUIREMENTS OF DIFFERENT RADIX (r), DIGIT SET ($\{-a, \ldots, a\}$), PRECISION OF RESIDUAL ESTIMATE FOR SELECTION (σ), PRECISION OF d USED TO PERFORM TABLE LOOK-UP (q) AND PRECISION OF TABLE ENTRIES (t).

The value of the divisor is in the usual range

$$\frac{1}{2} \leq \|d\|_\infty < 1 \tag{16}$$

Its estimate $rnd(d, q)$ can be represented as 2 two's complement numbers for the real and imaginary parts

$$rnd(d, q) = rnd(d^R, q) + i\ rnd(d^I, q) \tag{17}$$
$$rnd(d^R, q) = \kappa_0^R.\kappa_1^R\kappa_2^R\kappa_3^R \ldots \kappa_{q-1}^R\kappa_q^R \tag{18}$$
$$rnd(d^I, q) = \kappa_0^I.\kappa_1^I\kappa_2^I\kappa_3^I \ldots \kappa_{q-1}^I\kappa_q^I \tag{19}$$

An additional bit κ_{-1} is required (to represent $+1$) as $\|rnd(d, q)\|_\infty \leq 1$, which will be handled as a special case. To reduce the number of address bits, the table can store corresponding values for $|rnd(d^R, q)|$ and $|rnd(d^I, q)|$,

$$|rnd(d^R, q)| = 0.\alpha_1^R\alpha_2^R\alpha_3^R \ldots \alpha_q^R \tag{20}$$
$$|rnd(d^I, q)| = 0.\alpha_1^I\alpha_2^I\alpha_3^I \ldots \alpha_q^I \tag{21}$$

which eliminates the need for bits κ_0^R and κ_0^I (the sign) to be used when forming an address. Likewise, since $\|d\|_\infty \geq \frac{1}{2}$ we know that either $\alpha_1^R = 1$ or $\alpha_1^I = 1$ [6] (or both). Had an address been formed using

$$\alpha_1^R\alpha_2^R\alpha_3^R \ldots \alpha_q^R\alpha_1^I\alpha_2^I\alpha_3^I \ldots \alpha_q^I$$

then the address would require $2q$ bits. Given that

$$\gamma(d^R, d^I) = \frac{1}{d^R + id^I} = \frac{d^R - id^I}{(d^R)^2 + (d^I)^2} \tag{22}$$
$$\gamma(d^I, d^R) = -\gamma(d^R, d^I) \tag{23}$$

we could check if $\alpha_1^R = 1$, if so then the address is formed via

$$\alpha_2^R\alpha_3^R \ldots \alpha_q^R\alpha_1^I\alpha_2^I\alpha_3^I \ldots \alpha_q^I$$

otherwise, it must be true that $\alpha_1^I = 1$ so the address is formed as

$$\alpha_2^I\alpha_3^I \ldots \alpha_q^I\alpha_1^R\alpha_2^R\alpha_3^R \ldots \alpha_q^R$$

and the results obtained from the table look-up are negated based on (23). This reduces the number of address bits to $2q - 1$ (halving the memory required) while introducing little additional overhead.

Extra care must be taken with the aforementioned approach; although it is true that dividend is assumed to be bounded by $-1 < \|d\|_\infty < 1$, it is certainly not true that $-1 < rnd(d^R, q) < 1$, in fact $-1 \leq rnd(d^R, q) \leq 1$ (same holds for $rnd(d^I, q)$). The two's complement representation of the rounded divisor shown in equations (18) and (19) has range $[-1, 1)$. Negating -1 in two's complement with the given representation is a special case; recalling that +1 is also a special case, the input is divided into two cases: $\|rnd(d, q)\|_\infty < 1$ and $\|rnd(d, q)\|_\infty = \pm 1$.

Another special case occurs when negating the results obtained from the table look-up due to the swapping discussed earlier. For positive values of d^R and d^I the real part of $1/d$ is positive and the imagary part negative. The real part of $1/d$ is positive for positive values of d^R and the imaginary part of $1/d$ is negative for positive values of d^I. Since $1/2 \leq \|rnd(d, q)\|_\infty \leq 1$ and the table only stores values for positive d^R and d^I values, then $0 \leq K^R \leq 2$ and $-2 \leq K^I \leq 0$. Therefore the table should only contain the magnitude of the value, which can be represented in $2 + t$ bits–this will present no anomalies if 3 integer bits are used for the negated values, i.e., the ROM will store $2 + t$ bits, but the negated value will be $3 + t$ bits

Here we describe the operation of the table incorporating

the special cases,

$$rnd(d^R, q) = \kappa_{-1}^R \kappa_0^R . \kappa_1^R \kappa_2^R \kappa_3^R \dots \kappa_q^R$$
$$rnd(d^I, q) = \kappa_{-1}^I \kappa_0^I . \kappa_1^I \kappa_2^I \kappa_3^I \dots \kappa_q^I$$
$$\mathbf{A}^R = |rnd(d^R, q)| = \alpha_0^R . \alpha_1^R \alpha_2^R \alpha_3^R \dots \alpha_q^R$$
$$\mathbf{A}^I = |rnd(d^I, q)| = \alpha_0^I . \alpha_1^I \alpha_2^I d_3^I \dots \alpha_q^I$$
$$A = \begin{cases} \alpha_2^R \alpha_3^R \dots \alpha_q^R \alpha_1^I \alpha_2^I \alpha_3^I \dots \alpha_q^I & \text{if } \alpha_1^R{=}1, \\ \alpha_2^I \alpha_3^I \dots \alpha_q^I \alpha_1^R \alpha_2^R \alpha_3^R \dots \alpha_q^R & \text{otherwise} \end{cases}$$
$$A_s = \begin{cases} \alpha_1^I \alpha_2^I \alpha_3^I \dots \alpha_q^I & \text{if } \mathbf{A}^R{=}{\pm}1, \\ \alpha_1^R \alpha_2^R \alpha_3^R \dots \alpha_q^R & \text{otherwise} \end{cases}$$
$$(U^R, U^I) = \begin{cases} (1/2, -1/2) & \text{if } \mathbf{A}^R{=}1, \mathbf{A}^I{=}1, \\ (1/2, 1/2) & \text{if } \mathbf{A}^R{=}1, \mathbf{A}^I{=}{-}1, \\ (-1/2, -1/2) & \text{if } \mathbf{A}^R{=}{-}1, \mathbf{A}^I{=}1, \\ (-1/2, 1/2) & \text{if } \mathbf{A}^R{=}{-}1, \mathbf{A}^I{=}{-}1, \\ ROM_s[A_s] & \text{if } \mathbf{A}^R{=}{\pm}1 \text{ and } \mathbf{A}^I{\neq}{\pm}1 \\ & \text{or } \mathbf{A}^I{=}{\pm}1 \text{ and } \mathbf{A}^R{\neq}{\pm}1 \\ ROM[A] & \text{otherwise} \end{cases}$$
$$neg^R = \begin{cases} 1 & \text{if real and imaginary swapped,} \\ 0 & \text{otherwise} \end{cases}$$
$$neg^I = \begin{cases} 1 & \text{if real and imaginary } \textbf{not} \text{ swapped,} \\ 0 & \text{otherwise} \end{cases}$$
$$K^R = (-1)^{neg^R} U^R$$
$$K^I = (-1)^{neg^I} U^I$$

TODO: Update to new results. From Table I, we have $q = 6$ and $t = 13$ for radix $r = 4$ and $a = 3$.

- ROM: This ROM has 11 address bits and is 30 bits wide, which can be mapped to 15 Altera Stratix II M4K RAM blocks. Give a percentage
- ROM_S : This ROM has 6 address bits and is 30 bits wide, which can be mapped to a single Altera Stratix II M4K RAM block. Give a percentage

A schematic corresponding to the described look-up scheme is shown in Fig. 3.

The other two parts of the prescaling step involve computing Kz and Kd which will be used to initialize and carry out the digit recurrence algorithm. Once K is determined x and y can be computed via,

$$\begin{aligned} x &= (K^R + iK^I)(z^R + iz^I) \\ &= (K^R z^R - K^I z^I) + i(K^I z^R + K^R z^I) \\ y &= (K^R + iK^I)(d^R + id^I) \\ &= (K^R d^R - K^I d^I) + i(K^I d^R + K^R d^I) \end{aligned}$$

Since multipliers are costly in hardware, the complex valued products will be computed one at a time. Coincidentally, $y = Kd$ is not required until after the residuals have been initialized with $x = Kz$, which can be computed in the previous cycles. Figure 4 shows the block diagram for the scaling module. The module uses several signals to control the

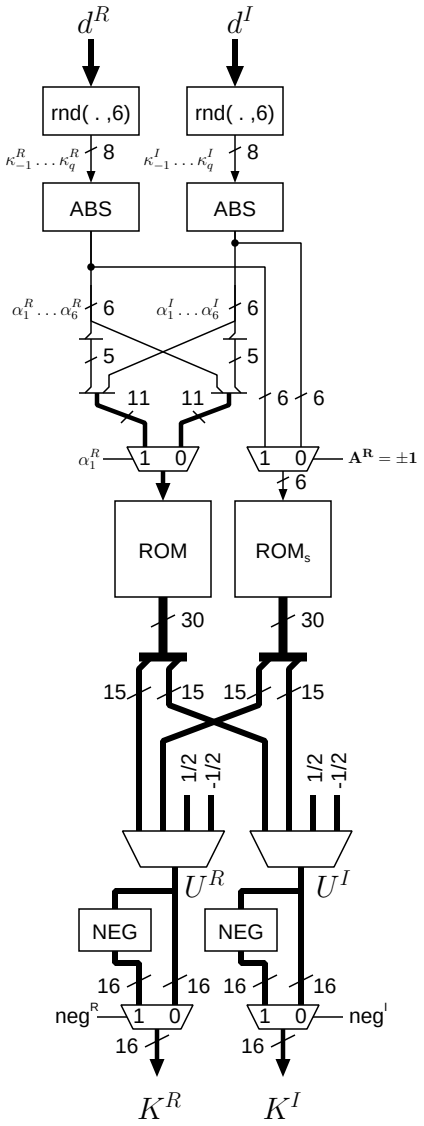

Fig. 3. TODO: update the figure. Prescaling ROM. The ABS block computes the absolute value of a two's complement number. Blocks $rnd(.,6)$ round their argument to the sixth fractional position. NEG blocks negate their argument, a two's complement number.

data path: en_{inputs}, en_{pres}, en_{sc}, and sel_{mul}. Control signals en_x are clock enable signals to registers to control when data is latched. Clock enables on registers are used to facilitate multi-cycle paths which are necessary due to the larger delay of the prescaling logic.

In Fig. 4 en_{inputs} controls when the inputs to the complex division unit are latched such that the values can be retained throughout the course of the operation–this is not necessarily unique and depends on the how the module is interfaced to other logic. For example, if the external logic feeds the arguments to the complex division unit in two cycles: sending (z^R, z^I) in the first and (d^R, d^I) in the second, then only 2 register banks are required for the inputs as opposed to 4. The current design reflects the assumption that the module receives its arguments in the same cycle, i.e., as (z^R, z^I, d^R, d^I).

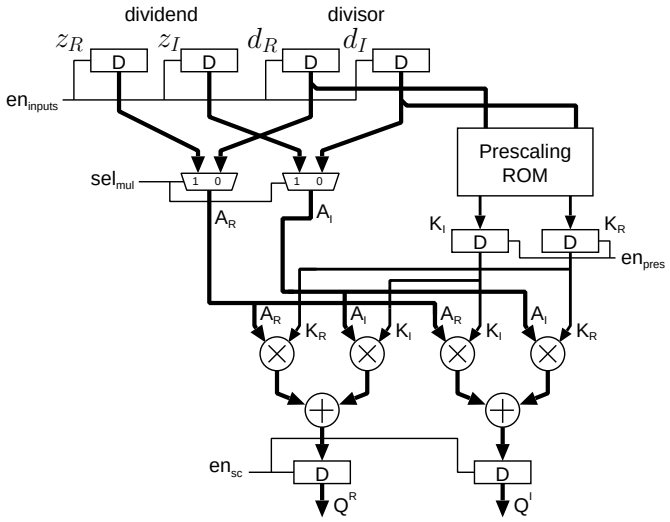

Fig. 4. Prescaling module. The Prescaling ROM block above is the module shown in Fig. 3.

Signal en_{pres} controls storing of the results of the prescaling ROM look-up, retained throughout the course of the operation. Signals sel_{mul} and en_{sc} are used to share the multipliers so that prescaling of the dividend and the divisor occurs in separate prescaling cycles. Although the prescaled value $x = Kz$ is also fed through the registers controlled by en_{sc}, its value is not retained but over-written in the next cycle by $y = Kd$. The same enable signal (en_{sc}) is used once more to assure that the value of y is retained in these registers which feed the recurrence modules discussed in Section II-C.

B. Bounds of Values

It is important to characterize the bounds of the inputs to the complex division module in addition to the bounds of the prescaled values which predetermine the width of inputs to the recurrence modules.

The input d is in the range $1/2 \leq \|d\|_\infty < 1$, and through our convergence analysis further constrained $\|Kd - 1\|_\infty < \epsilon_s$. This implies that the prescaled value y satisfies

$$\begin{aligned} & \max(|y^R - 1|, |y^I|) < \epsilon_s \\ \Rightarrow\ & |y^R - 1| < \epsilon_s \\ & |y^I| < \epsilon_s \end{aligned}$$

Since $|y^R| < 1 + \epsilon_s$, its representation in two's complement would require 2 integer bits and n fractional bits.

Likewise, the constraint (14) determines the maximum value that the residual could possibly take. For our design point $\sigma = 4$ which means that the residual is bounded by,

$$\begin{aligned} & \|w[j]\|_\infty \leq \frac{1}{4}\left(3 + \frac{1}{2} + 2^{-4}\right) = 57/64 \\ \Rightarrow\ & |w^R[0]| = |\Re(Kz)| = |x^R| \leq 57/64 \\ & |w^I[0]| = |\Im(Kz)| = |x^I| \leq 57/64 \end{aligned}$$

Therefore, the prescaled value (x^R, x^I) requires only a single integer bit, and n fractional bits. We are interested in determining a bound on z which we can derive from the bound on w,

$$\|w[0]\|_\infty = \|Kz\|_\infty \leq 2\|K\|_\infty \|z\|_\infty \leq 57/64 \tag{24}$$

since $\|K\|_\infty \leq 2$ then

$$\|z\|_\infty \leq 57/256 \tag{25}$$

requiring only $n-1$ fractional bits, with most siginificant bit having weight 2^{-2}.

C. Digit-Recurrence Iterations

The digit-recurrence iterations compute the residuals (5) (6) and perform quotient-digit selection based on a short non-redundant estimate of the residuals as shown in Eq. (10) and (11).

The recurrences in (5) and (6) are structurally the same. Namely,

$$w[j+1] = 4w[j] + \sigma_1 y^R + \sigma_2 y^I \tag{26}$$

The residuals are computed in redundant form in order to reduce the cycle time by eliminating the need for long carry chains. In our implementation we used a carry-save form. The operation is expressed as

$$\begin{aligned} & (w_C[j+1], w_S[j+1]) = \\ & ADD_{[6:2]}(4w_C[j], 4w_S[j], \sigma_1^1 y^R, 2\sigma_1^2 y^R, \sigma_2^1 y^I, 2\sigma_2^2 y^I) \end{aligned} \tag{27}$$

where $ADD_{[6:2]}(a, b, c, d, e, f)$ is a $[6:2]$ carry-save adder taking 6 inputs and producing a carry vector and sum vector, shown in Fig. 5. The digits σ_1 and σ_2 are in the digit set $\{-3, \ldots, 3\}$ so we implement this digit multiplication by decomposing $\sigma_k = 2\sigma_k^2 + \sigma_k^1$ where $\sigma_k^i \in \{-1, 0, 1\}$. Multiplying by negative one is achieved by inverting the input and adding a carry-in to the reduction module.

Digit selection is performed by finding a short precision estimate of the residual and rounding to the nearest integer via a small CPA and table. In the discussion that follows we generally say residual without referring specifically to the real or imaginary part–the analysis holds for both residuals w^R and w^I. In Section II-B we determined that the residual has a single integer bit and n fractional bits, i.e., it is of the form $w = w_0.w_1 w_2 \ldots w_n$ with value $\sum_{i=0}^{n} w_i 2^{-i}$. In redundant form $w = w_c + w_s$ where $(w_c, w_s) = (C_0.C_1C_2C_3 \ldots C_n, S_0.S_1S_2S_3 \ldots S_n)$. Recalling that selection is performed via,

$$q_{j+1} = Sel(est(4w[j], \sigma))$$

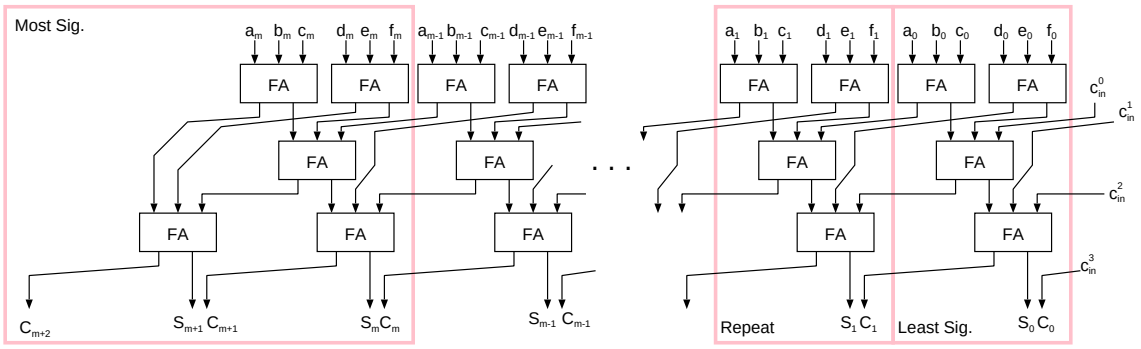

Fig. 5. $[6:2]$ Adder module. The adder consists of three different slices: the least significant slice, which sums 6 arguments and takes 4 carry-ins, the repeat slice which sums 6 arguments and takes 4 lateral carries and produces 4 lateral carries to the subsequent slice, and the most significant slice.

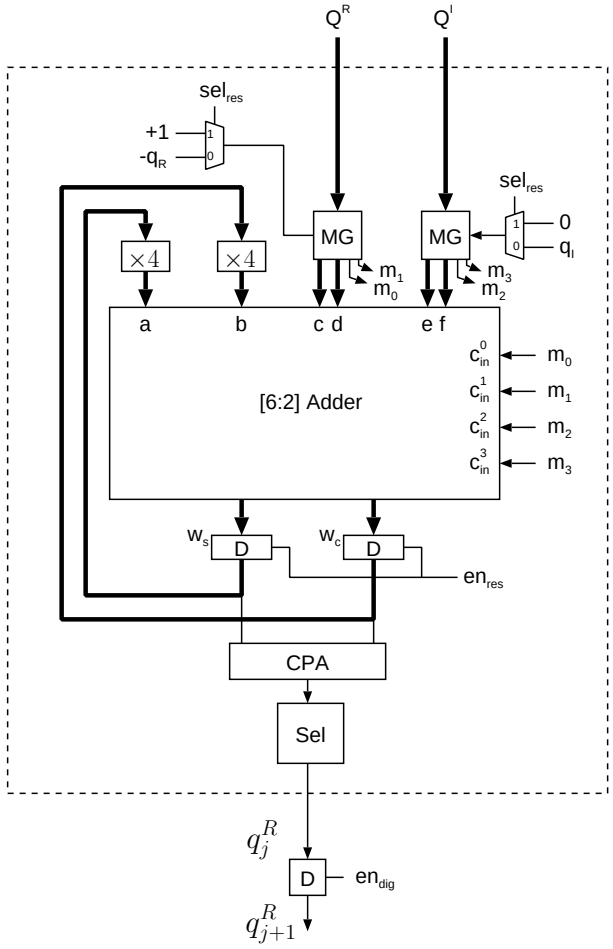

Fig. 6. Real recurrence module. Blocks $\times 4$ shift their argument right by 2 binary places. Blocks MG compute σ times their argument using the σ_k^i decomposition discussed. The CPA module is a carry propagate adder which computes a short non-redundant estimate of the residual. The Sel module takes as argument this estimate and outputs the next quotient digit.

where $\sigma = 4$ as determined in section II-A, we know that

$$est(4w[j], 4) = w_0 w_1 w_2 . w_3 w_4 w_5 w_6 = \sum_{i=0}^{n} w_i 2^{-i+2}$$
$$est_{ERR}(4w[j], 4) < 2^{-4}$$

now since w is in redundant form,

$$\begin{aligned} est(4w_c[j], 5) &= C_0 C_1 C_2 . C_3 C_4 C_5 C_6 C_7 \\ est(4w_s[j], 5) &= S_0 S_1 S_2 . S_3 S_4 S_5 S_6 S_7 \\ g &= est(4w_c[j], 5) + est(4w_s[j], 5) \\ est_{ERR}(4w_c[j], 5) &+ est_{ERR}(4w_s[j], 5) < 2^{-4} \end{aligned} \tag{28}$$

which gives us the short precision estimate of the residual g. It is important to realize that $g \neq est(4w[j], 4)$ in general but that they commit the same maximum error 2^{-4} in their approximation of $w[j]$. The addition in equation (28) requires the CPA that we have been referring to during this discussion.

$$\begin{aligned} g_{-2} g_{-1} g_0 . g_1 \ldots g_5 = \\ CPA(C_0 C_1 C_2 . C_3 \ldots C_7, S_0 S_1 S_2 . S_3 \ldots S_7) \end{aligned} \tag{29}$$

In order to round g and take the integer part one can use a small table as in table II by introducing an additional variable $g_z = g_2 + g_3 + g_4 + g_5$ (i.e. the logical or of bits g_2 through g_5). This table is a function of 5 bits and produces three bits of output (for the encoding of q_{j+1}) and will efficiently map to LUTs.

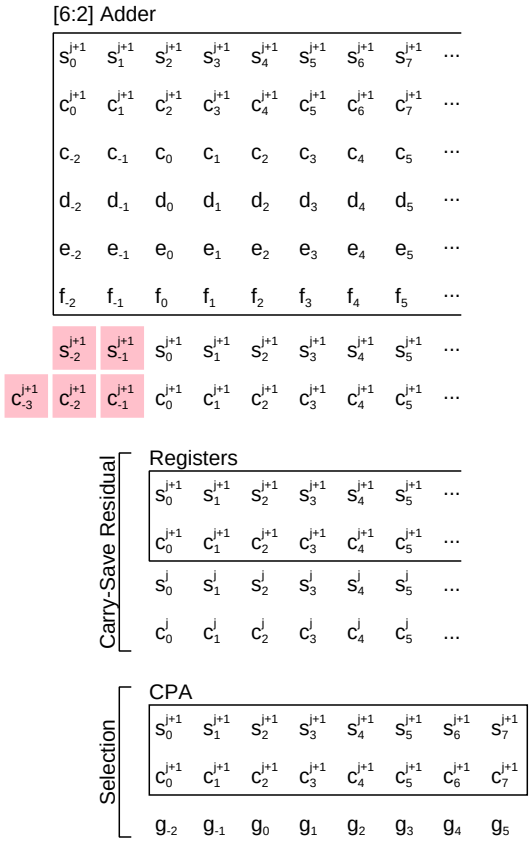

Fig. 7. First implementation of recurrence reduction. Each rectangular box represents some functional block where the bits inside show the inputs to that block and the bits beneath show the corresponding outputs. There are 5 bits produced by the $[6:2]$ adder in this figure which have a shaded square background to signify that these output bits don't drive any logic and are left "open".

g_{-2}	g_{-1}	g_0	g_1	g_z	q_{j+1}
0	0	0	0	-	0
0	0	0	1	-	1
0	0	1	0	-	1
0	0	1	1	-	2
0	1	0	0	-	2
0	1	0	1	-	3
0	1	1	0	-	3
1	0	0	1	1	-3
1	0	0	1	1	-3
1	0	1	0	-	-3
1	0	1	1	0	-3
1	0	1	1	1	-2
1	1	0	0	-	-2
1	1	0	1	1	-1
1	1	1	0	-	-1
1	1	1	1	1	0

TABLE II
ROUNDING TO INTEGER PART.

D. Optimizing the Recurrence Implementation

A straightforward implementation of the recurrence is shown in Fig. 7. There are several opportunities for optimization in Fig. 7,

- Since the residual is bounded in range $(-57/64,\ 57/64)$, there is only one integer bit required to store the value of the residual. Based on this observation there is no need to find the sum of bits with weight greater than $2^0 = 1$.
- The recurrence implementation can be optimized in the most significant bits by using the non-redundant value computed for selection in the adder as opposed to the redundant form stored in the registers. Although addition of a short CPA delay to the most significant bits seems counter-intuitive to optimization, it turns out that for all fitting attempts to the Stratix II architecture this path was not the critical path–paths with routing delays dominated the critical path (short carry chains don't exhibit routing delays as there are dedicated carry paths in Adaptive Logic Modules [15]). Since using the non-redundant portion didn't introduce a new critical path and reduced the input bits it served as a pragmatic optimization technique. The non-redundant approximation g computed for selection can be used in the addition as opposed to using the $[6:2]$ adders. This simplifies the $[6:2]$ adder to a $[5:2]$ adder requiring 4 lateral carries (as opposed to a conventional $[5:2]$ adder which only requires 3 lateral carries) which we denoted as $[5:2]^4$–the lateral carries come from the previous $[6:2]$ adder. The interface between the $[6:2]$ adder, the $[5:2]^4$ adder and the XOR slice is shown in Fig. 9.
- The $[5:2]^4$ adder produces both s_1^{j+1} and c_0^{j+1}, it is unnecessary to produce s_0^{j+1} with the same module since we will discard c_{-1}^{j+1}. Bit s_0^{j+1} is just the sum modulus 2 of all bits of weight 1 plus the lateral carries, which can be computed via exclusive-ors (XOR).

Applying all mentioned optimizations we get an improved design shown in Fig. 8.

III. Design Methodology and Results

A. Methodology

The proposed designs were written at the RTL level using VHDL and simulated for functional correctness with Modelsim-Altera Edition 8.1. They were mapped to an Altera Stratix II architecture using Quartus II 8.1 flow tools. The Quartus Classic Timing Analyzer was used to determine the timing characteristics of the circuit in addition to placing constraints on ROM look-up and prescaling registers to inform the tool of multi-cycle paths.

The multiplies performed in the prescaling module map to the Altera DSP blocks for precisions up to 36 bits–these modules support up to 36×36 multiplication. It does not really make sense to go beyond this precision as the current design choice is targeted for an architecture which supports fast multipliers. For larger precisions it seems more sensible to design an efficient custom rectangular multiplier.

B. Implementation Area and Delay Characteristics

The results show the number of ALUTs (Adaptive LUTs [15]), of which there are two in every ALM (Adaptive Logic Module) : the basic building blocks for logic in Altera Stratix II devices. The DSP blocks on Stratix II architectures support

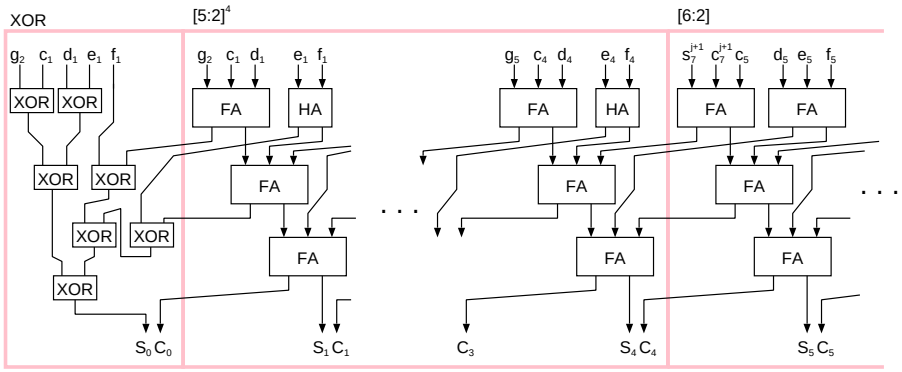

Fig. 9. Interface of the $[6:2]$ adder, $[5:2]^4$ adder and the XOR slice.

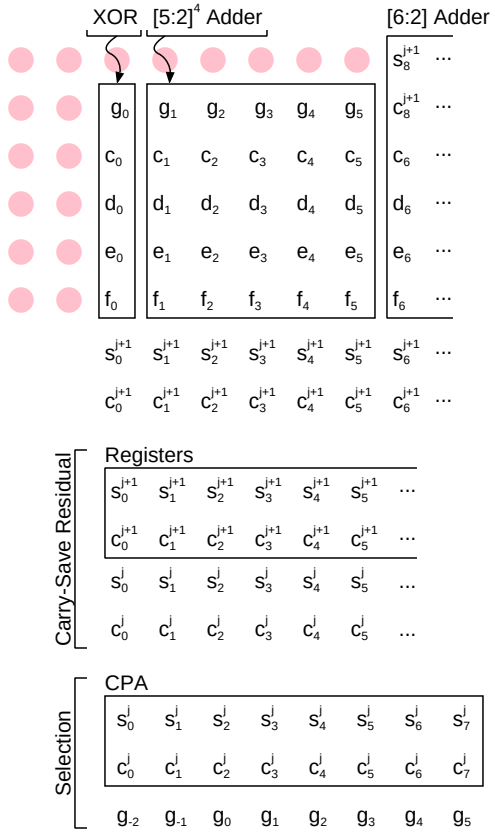

Fig. 8. Optimized implementation of recurrence reduction. The effective reduction is visualized–the shaded circles signify input bits that were removed as they were deemed unnecessary.

either eight 9×9 multiplies, four 18×18 multiplies or one 36×36 multiply. The proposed design is limited to the availability of multiplication units and therefore we have only reported results for two design points, one utilizing a single DSP block with four 18×18 multipliers and the other using four DSP blocks each performing 36×36 multiplies. The on-the-fly conversion module was actually not implemented for this particular design and results were kept in signed-digit form. Each OFC block contains two multiplexers and register

Precision [bits]	18	36
ALUTs	612	1093
DSP Block (9-bit elements)	8	32
Registers	214	394
M4K RAM blocks	15	15
Critical path (ns)	3.606	3.724
Max. frequency (MHz)	277.32	268.53
Prescaling look-up (Cycles)	2	2
Prescaling (Cycles)	2*2	2*3
Total prescaling (Cycles)	6	8
Iterations (cycles)	9	18
Total time (latency) (ns)	54	97

TABLE III
TODO: UPDATE TO NEW RESULTS. RESULTS FOR PRECISION 18 AND 36 COMPLEX DIVISION UNITS IMPLEMENTED ON AN ALTERA STRATIX II FPGA.

banks n bits wide, with a small amount of control logic. For the 18 bit design, this would amount to approximately 72 ($2\times$ OFC units) $\times$($2\times$ Register banks per OFC unit) $\times$(18 bits per register bank) additional registers–since each ALM holds two registers, it can be fairly extrapolated that 36 additional ALMs: 72 ALUTs and registers would suffice to form the conventional output format for the 18-bit complex division unit (and 72 ALMs: 144 ALUTs and registers for the 36-bit design).

The most common scenario we forsee a designer will face when determining the usefulness of a complex division unit is when comparing performance to a software based solution. One such software solution presented in [14] is based on the following,

$$\frac{a+jb}{c+jd} = \begin{cases} \frac{a+b(d/c)}{c+d(d/c)} + j\frac{b-a(d/c)}{c+d(d/c)} & \text{if } |c| \geq |d| \\ \frac{b+a(c/d)}{d+c(c/d)} + j\frac{a-b(c/d)}{d+c(c/d)} & \text{if } |d| \geq |c| \end{cases} \quad (30)$$

which requires significantly more arithmetic operations, 4 conventional divisions ad 3 multiplications. A complex divider have been described in [16] implementing Smith's formula with a pipelined multiplier, divider, and adder for an 8-bit precision (+4 guard bits). The scheme uses small number of Xilinx Virtex-II slices and operates at 100 MHz. Another design for a complex divider is proposed in [17]. It uses an algorithm similar to the SRT division. It also has an efficient

implementation and a latency for 15-bit precision of about 600ns, and a throughput of 1.6MHz. These two approaches are not comparable to our higher-radix approach in terms of speed. They have an advantage that there is no prescaling and no tables for prescaling factors. Radix-2 complex online arithmetic developed in [9] is not directly comparable to our implementation.

IV. Conclusions and Future Work

We presented the design and implementation of a radix-4 complex division unit with a single prescaling table. The implementation requires 1093 ALUTs, with a critical path of 3.724 ns, and a maximum frequency of 268.53 MHz. The prescaling table requires 2K words of 30 bits. To our knowledge no comparable implementation exists at the time and our results initiate a point of reference for other hardware based designs. In future work we plan on exploring the use of multipartite tables to reduce the table requirements in addition to developing specialized rectangular multipliers to enable higher radix designs.

Acknowledgments. We thank Altera Corporation for providing the tools and FPGA devices used in this research.

References

[1] A. F. Molisch. Wireless Communications. John Wiley and Dons Ltd., 2005.

[2] J. X., L. Guo, Y. Chen, and J. Zhang. Study of GPS Adaptive Antenna Technology Based on Complex Number AACA, *IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1-4.

[3] S.R. Dicker et al. Cbm observations with the Jodrell Bank - iac interferometer at 33 Ghz. *Mon. Not. R. Astron. Soc.*, 2000, 00:1-12.

[4] G. Vandersteen et al. Comparison of arithmetic functions with respect to Boolean circuits. *In 58th ARFTG Conference Digest RF Measurements for a Wireless World*, 2001, pp. 466-470.

[5] M.D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann Publishers, San Francisco, 2004.

[6] M.D. Ercegovac and J.-M. Muller. Complex Division with Prescaling of Operands. *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 293-303, 2003.

[7] M.D. Ercegovac and J.-M. Muller, Design of a complex divider. *Proc. SPIE on Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, pp. 51-59, 2004.

[8] M.D. Ercegovac and J.-M. Muller. Complex Square Root with Operand Prescaling. *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 293-303, 2004.

[9] R.D. McIlhenny, *Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications*, Ph.D. Dissertation, Computer Science Department, University of California, 2002.

[10] V. Oklobdzija, D. Villeger and T. Soulas, An Integrated Multiplier for Complex Numbers. *J. of VLSI Signal Processing*, vol.7, no. 3, pp.213-222, May 1994.

[11] A.F. Tenca, M.D. Ercegovac. Design of high-radix digit slices for online computations. In *SPIE Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic*, Bellingham, 1996.

[12] B.W.Y. Wei, H. Du, and H. Chen, A Complex-Number Multiplier Using Radix-4 Digits. *Proc. 12th IEEE Symposium on Computer Arithmetic*, pp. 84-90, 1995

[13] P. Dormiani, M.D. Ercegovac, and J-M. Muller, On the Design and Implementation of Complex-valued Division Unit with Operands Prescaling. Computer Science Department, UCLA, Internal Report 2009.

[14] R.L. Smith. Algorithm 116: Complex division. Communications of the ACM, 5(8):435, 1962.

[15] http://www.altera.com/

[16] F. Edman and V. Oewall, Fixed-point Implementation of a Robust Complex Valued Divider Architecture,Proceedings of ECCTD05, Cork, Ireland, August 2005.

[17] J. Liu, B. Weaver and Y. Zakharov, "FPGA Implementation of Multiplication-Free Complex Division", Electronic Letters, 17th January 2008, Vol. 44, No. 2.