A Design of Radix-2 On-line Division Using LSA Organization

Alexandre F. Tenca
Electrical and Computer Engr.
Oregon State University
Oregon, USA
tenca@ece.orst.edu

Syed Ubaid Hussaini
CME Department
University of Appl. Sci.
Offenburg, Germany
ubaid_hussaini@yahoo.com

Abstract

This paper presents the application of the Linear Sequential Array (LSA) organization to on-line division. The LSA method was originally developed for conventional digit-recurrence algorithms, but in this paper we apply it to the on-line division algorithm. The resulting architecture is a modular and fast pipelined structure which, due to a constant fan-out, make the critical path delay, and consequently the clock cycle time, less sensitive to operand's precision. Such approach is particularly suitable for FPGA implementation for its modularity and reduced fanout. The basics of on-line division is presented, followed by the derivation of data dependencies and architecture according to the LSA design methodology. Experimental data is provided for both the LSA on-line divider design and standard on-line design, using 0.5μm CMOS ASIC and FPGA technologies.

1 Introduction

On-line arithmetic is attractive in many digital signal applications [7, 9, 1], in particular in implementing recursive filters and composite operations such as rotation factors [3]. On-line operations are particularly suitable to be used with A/D and D/A converters, which usually generate most-significant (MS) bits first. The most-significant-digit-first (MSDF) operation mode of all on-line arithmetic operators allows the design of highly concurrent networks of sequential modules.

The advantages of on-line over conventional arithmetic improves with longer precision. However, long precision increases the fan-out of some signals and, consequently, the cycle time. Moreover, a long-precision implementation may require several chips which, due to a significant off-chip connection delay, leads to further performance degradation. The key objective of this paper is to present a LSA organization [5] for radix-2 on-line division that solves the performance degradation problems caused by high signal fan-out and off-chip delays.

The application of this technique to on-line multiplication was done in [8]. In this paper we apply the technique to on-line division, for which the broadcast problem is even more severe than that for on-line multiplication. We obtain a fast and scalable design alternative.

The LSA organization follows the same design approach of systolic designs [4]. The work shown in [5] describes a conversion method from full-precision digit-recurrence arithmetic algorithm to a linear sequential array organization. The application of this approach to digit-recurrence division algorithm was presented in [6]. This paper presents the application of this technique to radix-2 on-line division, and compares the resulting implementation with the standard on-line division algorithm implementation [10, 11].

The next section presents an overview of radix-2 on-line division algorithm and the following section briefly describes major concepts for the LSA organization and the conversion method that is applied to on-line division. Section 4 describes the design of LSA modules for on-line division. Section 5 and 6 present simulation results and performance measurements and Section 7 includes final remarks and summary of this work.

2 On-line Division Overview

Algorithms for on-line division were presented in [2, 10, 12, 13, 11]. On-line division processes the operand digits serially. The quotient digits are generated at the same rate as the operand digits are received, and the first output digit is produced δ clock cycles after the input of the first operands' digit. The on-line delay δ varies from one on-line operation to another and depends on the number system, digit radix, and operand's value. A radix-2 on-line division unit to compute $Q = \frac{N}{D}$ implements the following recurrence equation:

$$W[j] = 2(W[j-1] - q_{j-1}D[j-1]) + n_{j+4}2^{-4} - d_{j+4}Q[j-1]2^{-4} \quad (1)$$

with the following conventions for the digit vectors:

$$N[j] = \sum_{i=0}^{j+\delta} n_i 2^{-i}, \quad D[j] = \sum_{i=0}^{j+\delta} d_i 2^{-i},$$

$$W[j] = \sum_{i=0}^{j} w_i 2^{-i}, \text{ and } Q[j] = \sum_{i=0}^{j} q_i 2^{-i}$$

where

$$W[j] = 2A[j-1] + n_{j+4}2^{-4} - d_{j+4}Q[j-1]2^{-4} \quad (2)$$

and

$$A[j] = |(N[j] - Q[j]D[j])2^j| < D[j] \quad (3)$$

are different forms of scaled residuals. The initial condition is $W[0] = N[0]$. The digit set for all vectors is $\{-1, 0, 1\}$. This type of signed digit is also called *sbit* in this work. It is worth to mention the on-line form for the operands, using N as an example:

$$N[j] = N[j-1] + n_{j+\delta}2^{-(j+\delta)} \quad (4)$$

The on-line delay for division was analyzed in [2, 11]. For radix 2, the on-line delay was calculated as 4 clock cycles. The bounds on the residual is given as $|A[j]| < D[j] - \frac{1}{8}$, with an estimate of the W ($\hat{W}$) truncated at the third fractional bit position ($t = 3$).

The quotient digit is produced by the *selection function*:

$$q_j = Sel(\hat{W}) = \begin{cases} 1 & \text{if } \hat{W} \geq 1/4 \\ 0 & \text{if } -1/4 \leq \hat{W} < 1/4 \\ -1 & \text{if } \hat{W} < -1/4 \end{cases} \quad (5)$$

Some conditions must be imposed to the input operands, such as: (i) $1/2 \leq D < 1$, (ii) $N < D$, since it is desirable that $Q < 1$.

The format of Eq. 1 is not adequate for implementation since q_{j-1} must be appended to $Q[j-1]$ in the same clock cycle for computation, and q_{j-1} is generated from data available at the beginning of the clock cycle. The same problem does not happen to d_{j+4}, which is an input value. A better form for the recurrence equation is:

$$W[j] = 2(W[j-1] - q_{j-1}D[j]) + n_{j+4}2^{-4} - d_{j+4}Q[j-2]2^{-4} \quad (6)$$

which can be derived from Eq. 1 and the on-line form of digit vectors (Eq. 4).

2.1 Standard Implementation

The standard implementation of an on-line division algorithm uses full precision append registers (for $D[j]$ and $Q[j-2]$), sbit-by-vector multipliers, full-precision adders (redundant adders), and a residual register, as shown in Figure 1. The divisor (d) and quotient (q) digits are broadcast to all bit positions of the append registers (APRs) and all bit positions of the sbit-by-vector multiplier (SBVM). These are the signals with the highest fan-out problem. For division, this problem is exacerbated by the fact that the quotient digit must be generated internally and then broadcasted to APR and SBVM components. This signal is in the critical path of the standard on-line division implementation.

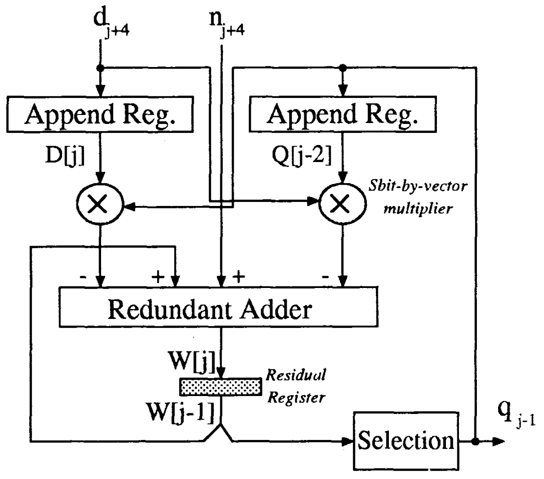


Figure 1. Standard on-line division implementation

3 Conversion to LSA organization

In this section, the basic concepts of the LSA organization are described and the methodology presented in [5] is applied to the on-line division algorithm.

3.1 The LSA organization

The LSA conversion algorithm is used to create a modular architecture. Modules are composed of submodules (or substages) as shown in Figure 2. Each module works on different algorithm steps, computing a unique set of weighted bits of the scaled residual.

Assume that a particular module (*reference module*) is working on step j, the modules that are working on step $j+i$, $i > 0$, are upstream to the module working on step j. The modules that are working on steps $j - i$, $i > 0$ are downstream to the reference module. The LSA theory [5] shows that the reference module may receive as much information as necessary from any upstream module (not shown in the Figure) by creating a delay buffer to make the data reach the reference module at the proper time. A module may also depend on downstream data, but only from its immediate downstream neighbor, as shown in Figure 2. Modulo 0 works with the most-significant digits.

Each LSA module has two substages. Substage 1 computes the most-significant (MS) bits of the bit group allocated to the module. Due to the redundant representation of the residual W, these MS bits can be computed using only (1) the pipelined input data bits and (2) data generated and stored by the module itself. Substage 2 of the LSA module then generates and stores the least-significant (LS) bits of the module's bit group. These LS bits cannot be computed without data from step $j-1$ that is generated by the first substage of its neighboring downstream module. Thus, substage 1 passes its results to substage 2 of its neighboring upstream module so that its substage 2 may complete the algorithm step in the current cycle. When both substages have finished computing the group of bits assigned to the module, a new cycle begins.

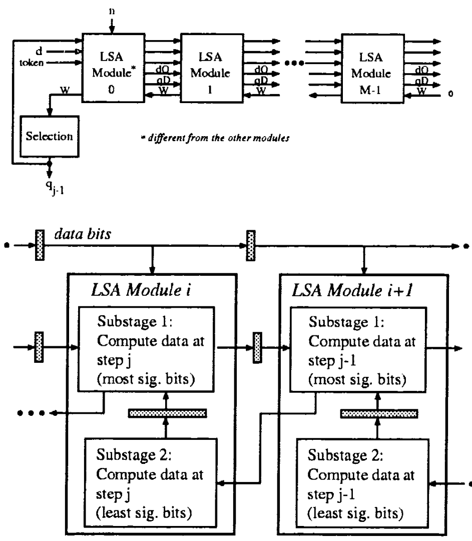

n
d
token
LSA Module* 0
LSA Module 1
LSA Module M-1
dQ
qD
W
0
Selection
* different from the other modules
q_{j-1}
data bits
LSA Module i
LSA Module i+1
Substage 1: Compute data at step j (most sig. bits)
Substage 1: Compute data at step j-1 (most sig. bits)
Substage 2: Compute data at step j (least sig. bits)
Substage 2: Compute data at step j-1 (least sig. bits)

Figure 2. LSA modular organization (adapted from [4] and [8])

3.2 Conversion Method

Conversion to a LSA scheme is based on matrices, data vectors, and dependency functions. The dependency functions are obtained from the input/output dependency for the redundant adders used to implement the recurrence in Equation (6). In this work we consider Carry-Save (CS) adders. Let the input operands to each CS adder be represented by the notation a, b, c, and d where (a, b) corresponds to the CS representation of the shifted residual $W[j-1]$, c corresponds to $d_{j+4}Q2^{-4}$, and d corresponds to $q_{j-1}D$. If the result $W[j]$ is represented by (WS, WC), corresponding to the sum and carry bits, we have the following dependencies for each bit position b of the output vectors generated by the 4-input CS adder (Figure 3):

$$\begin{aligned} WS_b &= g(a_b, a_{b+1}, b_b, b_{b+1}, c_b, c_{b+1}, d_b) \qquad (7) \\ WC_b &= h(a_{b+1}, a_{b+2}, b_{b+1}, b_{b+2}, c_{b+1}, c_{b+2}, d_{b+1}) \end{aligned}$$

where g and h are boolean functions. .

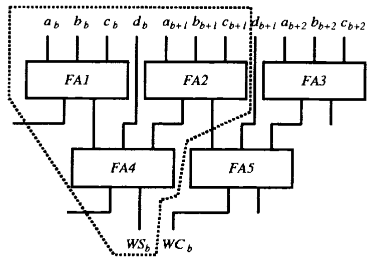

a_b b_b c_b d_b a_{b+1} b_{b+1} c_{b+1} d_{b+1} a_{b+2} b_{b+2} c_{b+2}
FA1
FA2
FA3
FA4
FA5
WS_b
WC_b

Figure 3. Section of CS adder that shows generation of WS and WC at bit position b

Figure 3 shows the dependency of a bit output position with respect to input signals coming to a 4-input CS adder. From the Figure it is clear that the calculation of the bit position b of the vector *WS* (WS_b) depends on adders FA1, FA2 and FA4. Therefore, WS_b can be calculated by knowing all the external inputs coming into these adders, which are a_b, b_b, c_b, d_b, a_{b+1}, b_{b+1}, and c_{b+1}. Similarly the calculation of the bit position b of the vector WC (WC_b) depends on the adders FA2, FA3, and FA5 only. Therefore, WC_b is calculated from all the inputs coming into these FAs, which are a_{b+1}, b_{b+1}, c_{b+1}, d_{b+1}, a_{b+2}, b_{b+2}, and c_{b+2}.

We will use this dependency information in the architecture derivation later on.

The generation of the residual (Eq. 6) requires the addition of n_{j+4}, but it happens only in one particular bit position of the adder. This position is allocated at the leftmost module of the LSA organization (MS) which will then require a special design to handle this extra sbit. The MS LSA module will also interface with the selection function. Both types of modules are presented in Section 4.

Vectors D and Q are easily obtained from the input sbits (d_{j+4}) and the quotient sbits (q_{j-1}), respectively, by the use of Append Registers (as shown in Figure 1). These registers include on-the-fly conversion to transform the signed-digit representation to two's complement representation [14, 8] since CS adders are used to generate the residual. Given that different modules are going to work on different bit slices of these vectors, the partitioning of APRs to each LSA module does not require any special design; we just split them over the modules.

The sbit-by-vector multiplier (SBVM) to obtain $q_{j-1}D[j]$ and $d_{j+4}Q[j-2]$ is implemented into a LSA module by breaking the full precision multiplier into smaller parts which are allocated to each LSA module (same as APR). The design of the Append Register and the SBVM components was presented in [8]. There are some issues involved in the allocation of specific bits of Q and D to a particular module which will be discussed in Section 4.

The generation of the residual W is the major design problem, and to solve it we first define the data vectors and dependency functions involved in the residual generation, and then apply the LSA conversion algorithm. The data vectors for conversion of the radix-2 on-line division algorithm are defined as: $v^1 = -d_{j+4}Q2^{-4}$, $v^2 = -2q_{j-1}D$, $v^3 = WS$, and $v^4 = WC$. The residual vector W is represented in CS form by two vectors, sum (WS) and carry (WC). The LSA conversion method is used over the data vectors according to CS adder data dependency functions shown in (Eq. 7) and the recurrence equation for on-line division (Eq. 6):

$$\begin{aligned}
v^1[j] &= w(Sel(v^3[j], v^4[j])) \text{ - msbits only} \\
v^2[j] &= z(Sel(v^3[j], v^4[j])) \text{ - msbits only} \\
v^3_b[j] &= g(v^1_b[j\text{-}1], v^1_{b+1}[j\text{-}1], v^2_b[j\text{-}1], v^2_{b+1}[j\text{-}1], \\
&\quad v^3_{b+1}[j\text{-}1], v^3_{b+2}[j\text{-}1], v^4_{b+1}[j\text{-}1]) \\
v^4_b[j] &= h(v^1_{b+1}[j\text{-}1], v^1_{b+2}[j\text{-}1], v^2_{b+1}[j\text{-}1], v^2_{b+2}[j\text{-}1], \\
&\quad v^3_{b+2}[j\text{-}1], v^3_{b+3}[j\text{-}1], v^4_{b+2}[j\text{-}1])
\end{aligned}$$

where the step number is represented by j, or $j-1$, between brackets. Subscripts represent the bit position. Observe that $v^3_b[j]$ depends on $v^3_{b+1}[j-1]$ which corresponds to a left shift of the previous residual, as shown in Equation 6. There are other inputs used to generate $v^3_b[j]$ which are obtained from the adder structure, as shown in Figure 3. The dependency of v^1 and v^2 on the MS bits of W (v^3 and v^4) will impact only the most significant module. For the conversion algorithm that follows this equation is not going to be used.

The conversion algorithm works on three two-dimension matrices S_1 (for substage 1), S_2 (for substage 2), and S_3 (for latch count). Each matrix has 4 rows (corresponding to the number of data vectors in the data path) and as many columns as needed to represent data dependencies. Considering that the bit position being analyzed is b, the matrix has one column for each bit position $(b+i)$, $i > 0$. The conversion algorithm has 7 steps. We summarize the discussion presented in [5] with some simplifications applicable to the on-line algorithm as follows [8]:

- Step 1: *Satisfy the downstream input dependencies*: set bits that are required to generate bit b. mark $S1_{k,b+i} = 1$ if $v^{k'}_b[j] = f(v^k_{b+i}[j-1])$ for $k, k' \in \{1,2,3,4\}$ and $i > 0$.
- Step 2: *Mark dependencies on substage 1 by bits upstream from the initial latched bits:* for each row of S_1, mark with a 1 all the positions to the left of the rightmost 1 in the row.
- Step 3: *Satisfy the downstream input dependencies of marked bits in S_1:* similar to step 1 however considering now the bits already marked on S1. mark $S2_{k,b+i+h} = 1$ if $v^{k'}_{b+i}[j] = f(v^k_{b+i+h}[j-1])$, for $k, k' \in \{1,2,3,4\}$, $h \geq 0$, and $i > 0$.
- Step 4: *For bits that are marked on both S_1 and S_2, unmark the bits in S_2 and mark the bits in S_3.* The bits generated by substage 1 are defined.
- Step 5: *Specify latches for remaining bits that depend on downstream bits generated in substage 1*: for $k, k' \in \{1,2,3,4\}, h \geq 0, L \geq c > b$, where L is the index of furthest marked column from b of combined $S1$ and $S2$
  if $S1_{k',c} = 0$ and $S2_{k',c} = 0$ then {
  . $S2_{k',c} = 1$,
  . mark $S3_{k,c+h} = 1$ if $(v^{k'}_c[j] = f(v^k_{c+h}[j-1]))$ and $S1_{k,c+h} = 1$ }
- Step 6: *Mark latches for all bits in substage 2:* if $S2_{k,i} = 1$ then $S3_{k,i} = 1$.
- Step 7: *Satisfy the upstream input dependencies.* This step is not required for the on-line division algorithm.

The application of the conversion algorithm based on the previously defined data vectors and data dependency functions results in the following three matrices:

$$S_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}, S_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$and\ S_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

These matrices provide a template for the LSA on-line division module that is presented in the next section. Substage 1 computes the MS bits while substage 2 computes the LS bits of the residual represented in CS form. Matrix S1 indicates that the generation of $v_b^3[j]$ and $v_b^4[j]$ requires the bits $b+1$ and $b+2$ of v^1, v^2, v^3 and v^4, plus bit $b+3$ of v^3. Matrix S2 shows that other bits at position $b+3$ to $b+5$ are required to generate the output of substage 2, which will be used by substage 1 in the next clock cycle. They show that each module must handle groups of 5 bits (result similar to LSA on-line multiplication [8]).

Matrix S3 has 2 zeros in the last two rows ($S_3(3,1)$ and $S_3(4,1)$). This information indicates that the MS bits of the sum and carry representation of the residual do not need to be saved in registers. In fact, as it will be shown in the next section, these bits are computed by substage 1 of the downstream neighbor. A snapshot of the data used/generated by a *reference module* and part of a downstream module in the LSA on-line divider is shown in Figure 4. The inside brackets notation was replaced by a superscript with the letters j, $j+1$, and $j-1$ to simplify the presentation. Vector v^1 is represented as dQ, and vector v^2 is represented as qD. The polygons show the data used by substages 1 and 2 in the reference module. The shaded area indicates the data bits stored in the reference module.

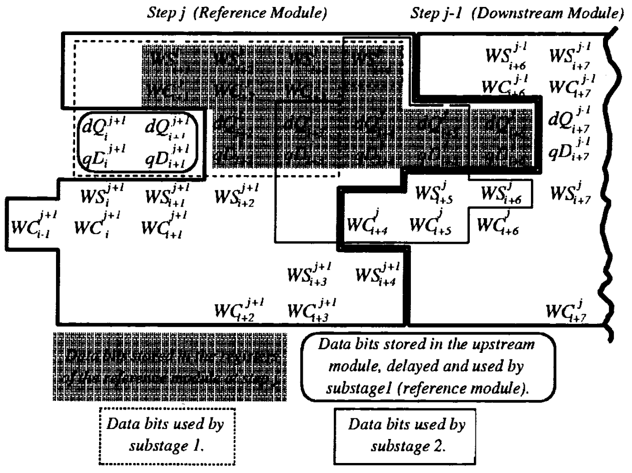

Step j (Reference Module)
Step j-1 (Downstream Module)
WS_{i+6}^{j-1} WS_{i+7}^{j-1}
WC_{i+6}^{j-1} WC_{i+7}^{j-1}
dQ_{i+7}^{j-1}
qD_{i+7}^{j-1}
dQ_i^{j+1} dQ_{i+1}^{j+1}
qD_i^{j+1} qD_{i+1}^{j+1}
WS_i^{j+1} WS_{i+1}^{j+1} WS_{i+2}^{j+1}
WS_{i+5}^{j} WS_{i+6}^{j} WS_{i+7}^{j}
WC_{i-1}^{j+1} WC_i^{j+1} WC_{i+1}^{j+1}
WC_{i+4}^{j} WC_{i+5}^{j} WC_{i+6}^{j}
WS_{i+3}^{j+1} WS_{i+4}^{j+1}
WC_{i+2}^{j+1} WC_{i+3}^{j+1}
WC_{i+7}^{j}
Data bits stored in the upstream module, delayed and used by substage1 (reference module).
Data bits used by substage 1.
Data bits used by substage 2.

Figure 4. Snapshot of data used and generated in a LSA on-line division module

Let us discuss the allocation of particular bits of dQ and qD to modules. Observe that substage 2 of the reference module (M_k) needs the bits dQ_{i+5}^j and qD_{i+5}^j to complete the computation of the residual values. However, these bits cannot be generated by the downstream module M_{k+1} since it is operating on step $j-1$. For this reason, at least these bits should be generated/stored in M_k. In our design we decided to store the bits dQ_{i+5}^j, dQ_{i+6}^j, qD_{i+5}^j, and qD_{i+6}^j inside module M_k, as shown in Figure 4. However, module M_{k+1} needs dQ_{i+5}^{j-1}, dQ_{i+6}^{j-1}, qD_{i+5}^{j-1}, and qD_{i+6}^{j-1} as inputs for its substage 1. The solution is to send these bits available on M_k to M_{k+1}, after a delay, such that M_{k+1} receives the correct information at the proper time.

Another observation is that the LS carry bit and the MS carry and sum bits of the bit-group representation of the residual (WC_{i+4}, WS_i, and WC_i) do not need to be stored. This information is used to reduce the number of memory elements required in the module design.

4 Design of radix-2 LSA on-line Divider Modules

From the previous analysis of the matrices (which provided a template for the LSA on-line module) and data dependencies we obtain the following data path design for the general LSA on-line division module (not the MS module) as shown in Figure 5.

The Figure shows the local communication between the reference module and its neighboring upstream and downstream modules. The reference module sends the two MS bits of WS and WC, generated by the CSA substage 1 block, to it's upstream neighbor module, and it also sends the two delayed LS bits of the products qD and dQ to its downstream neighbor. Similarly, the reference module receives the two delayed bits of the products qD and dQ from it's upstream module and the two MS bits of WS and WC calculated by the CSA substage 1 of the downstream neighbor. This kind of simultaneous data communication between modules is the most important characteristics of the LSA organization which allows each module in the array to share the job allocated to the entire system. This job sharing is done such a way that each module works in a different algorithm step.

The signals *token*1 and *token*2 are control signals that enable the Append Register slice allocated to a module to start appending the incoming digits to the vectors D and Q. To maintain the alignment condition on the data vectors, *token*2 is delayed 5 clock cycles related to *token*1. This way, *token*2 is generated when

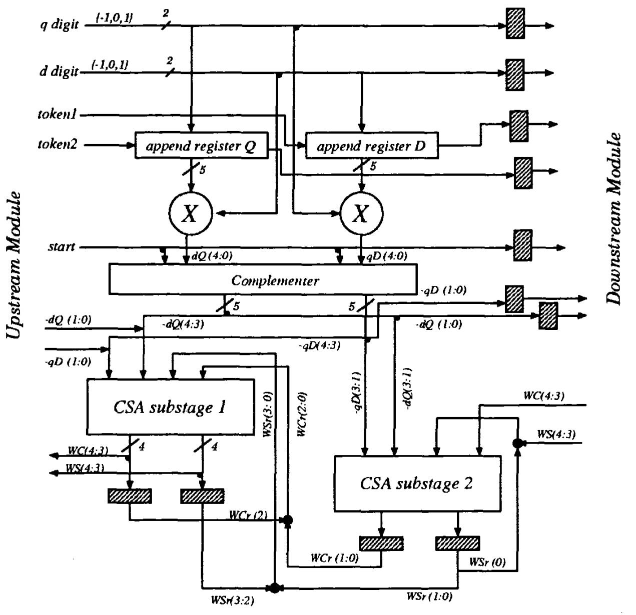


Figure 5. Data path for general LSA on-line division module

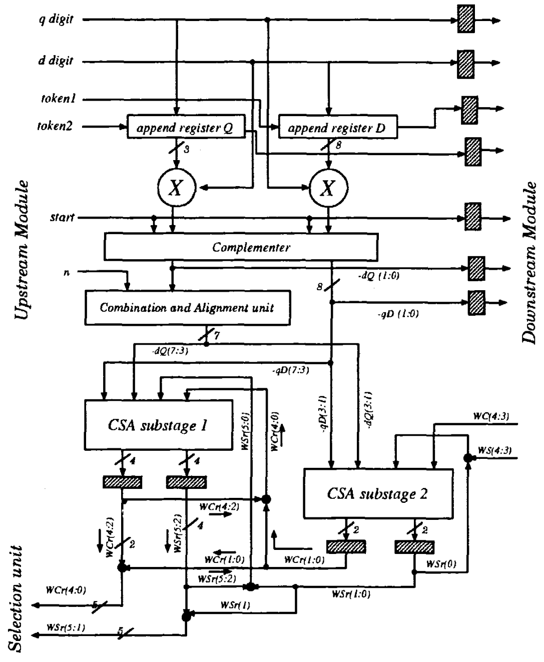


Figure 6. Implementation of MS LSA Module

the first fractional digit of the quotient shows up at the output of the selection logic. The selection logic does not generate output digits for the first 5 clock cycles of the on-line divider operation (including the clock cycle when the first fractional input digit is received).

The *complementer* block has an active high start control signal for the negation of the data vectors coming from the SBVM modules, which are in two's complement system. Basically, the complementer is a series of 2-input XOR gates which have one of its inputs connected to the *start* signal. The control signal is required to avoid the generation of complemented values before a module gets a valid input data to work with.

The design of the MS LSA module is a little bit different, as mentioned before. The data path for the MS LSA module was designed as shown in Figure 6. It considers several conditions already defined: the on-line delay ($\delta = 4$) for the division operator, the truncation requirements of the residue estimate imposed by the selection function, and the addition of digit n_{j+4} (n in the Figure).

The MS LSA module communicates with the downstream neighbor the same way as the other modules. The MS bits of the residual generated at the MS LSA module is transferred to the Selection Function module, which in turn computes the output digit q_{j-1} (q in the Figure).

In order to avoid an increase in the number of inputs into the CSA structure at the point where n_{j+4} is inserted, what would increase significantly the delay through the CS adder due to the extra level of FAs, we use a combinational logic to combine n and the vector $P = -dQ2^{-4}$ before the CS addition. This approach was proposed originally in [12]. Considering that digit n is represented in two's complement by a pair (n_s, n_v), which represents the value $-2n_s + n_v$, its insertion at the bit position 2^{-4} is done as follows:

$$\begin{array}{ccccccc} n_s & n_s. & n_s & n_s & n_s & n_v & 0 \\ p_s & p_s. & p_s & p_s & p_s & p_s & p_1 \\ \hline x & x. & x & x & x & y & p_1 \end{array}$$

where p_s represents the sign bit of the vector P and p_1 represents its first fractional digit. The variables x and y are generated by the following logic expressions:

$$x = n_s + (n_v)'p_s \qquad \text{and} \qquad y = n_v \oplus p_s$$

For a precision of n bits:

$$M = \lceil \frac{n-6}{5} \rceil + 1 \tag{8}$$

modules are required.

5 Simulation Results

The proposed LSA module designs and the standard on-line design were described using VHDL code, compiled and simulated using Modelsim[1]. A simulation result of the LSA on-line divider is shown in Figure 7 for the input operands $N = (0.011010111011110110)_2 = 0.420822$ and $D = (0.110111111110101111)_2 = 0.874382$, which results in $Q = (0.01111011001101...)_2 = (0.10\bar{1}111\bar{1}11\bar{1}\bar{1}11\bar{1}...)_2$ considering up to 14 fractional digits (the simulation result shows more than 14 digits). The digit -1 is represented as $\bar{1}$. A testbench to apply all possible 6-bit inputs was used to verify the divider operation. The test compares the on-line division quotient with the quotient obtained with a division operation available in a VHDL library. The result of standard and LSA on-line division design were also compared for verification. The LSA design does not increase the on-line delay (what usually happens when pipelining is used).

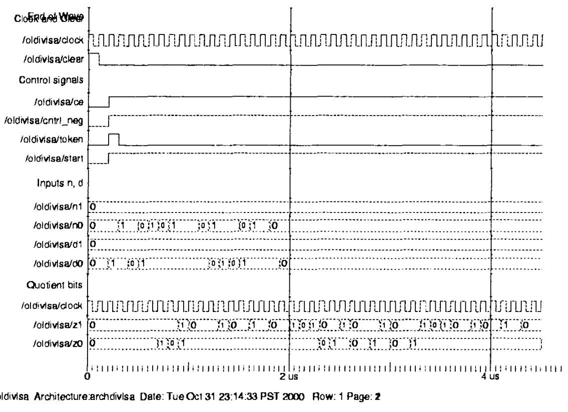
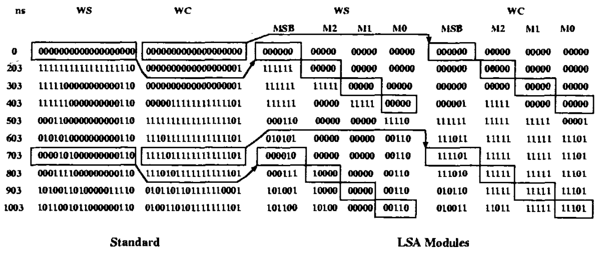


Figure 7. On-line LSA Divider - Simulation trace.

To better understand the calculation of the residue in the LSA organization consider Figure 8 which shows the calculation of the residue in terms of WS and WC both in the standard design and in the LSA design. Two instances of the residue calculation steps have been marked to show how different modules (MSB, M2, M1 and M0) work on the different steps of the residue calculation. The Figure also compares the values of WS and WC obtained using LSA approach with that of the standard approach.

[1]All CAD tools used in this work are Mentor Graphics tools



ns	WS	WC	WS MSB	WS M2	WS M1	WS M0	WC MSB	WC M2	WC M1	WC M0
0	0000000000000000000	0000000000000000000	000000	00000	00000	00000	000000	00000	00000	00000
203	1111111111111111110	0000000000000000001	111111	00000	00000	00000	000000	00000	00000	00000
303	1111100000000000110	0000000000000000001	111111	11111	00000	00000	000000	00000	00000	00000
403	1111110000000000110	0000011111111111101	111111	00000	11111	00000	000001	11111	00000	00000
503	0001100000000000110	1111111111111111101	000110	00000	00000	11110	111111	11111	11111	00001
603	0101010000000000110	1110111111111111101	010101	00000	00000	00110	111011	11111	11111	11101
703	0000101000000000110	1111011111111111101	000010	00000	00000	00110	111101	11111	11111	11101
803	0001111000000000110	1110101111111111101	000111	10000	00000	00110	111010	11111	11111	11101
903	1010011010000011110	0101101101111110001	101001	10000	00000	00110	010110	11111	11111	11101
1003	1011001011000000110	0100110101111111101	101100	10100	00000	00110	010011	11011	11111	11101

Standard

LSA Modules

Figure 8. Residue calculation steps in Standard and LSA on-line dividers

Precision	Standard		LSA	
	TClock (ns)	Area (gates)	TClock (ns)	Area (gates)
32	20.8	2412	22.9	2715
64	28.1	4877	23.0	5888

Table 1. Synthesis Results for $0.5\mu m$ CMOS ASIC technology

6 Experimental Results

The VHDL code for the LSA on-line divider was processed by Leonardo synthesis tool to generate the ASIC implementation of this design. The EDIF netlist was generated setting the technology to $0.5\mu m$ CMOS. This EDIF netlist was then converted into EDDM format and imported to the IC station where it was floorplanned, auto routed and extracted to generate SDF netlist. After filtering SDF netlist through RCD it was submitted to Velocity to perform Static Timing Analysis. The result of the synthesis process applied to two values of the operand precision is shown in Table 1.

The same VHDL code was also synthesized using CAD tools for Xilinx 4000 series of FPGAs. The results are shown in Table 2.

It was not our objective to excessively control the implementation of the designed modules but rather have some results to show the benefits of this design approach. Minimum constraints were imposed to the synthesis tools, for both implementation cases.

The results obtained in these experiments confirm our earlier statement that the LSA on-line divider is almost insensitive to the variation on the precision of the operands. For both FPGA and ASIC technologies, the design kept approximately the same clock cycle time for 32 and 64 bits of precision, while the standard implementation increased by 31-35% with the increase in precision. The decrease in the clock period with the

Precision	Standard TClock (ns)	LSA TClock (ns)
32	16.0	12.5
64	21.1	9.6

Table 2. Synthesis Results for Xilinx 4000 series FPGA

increase in precision obtained for the FPGA implementation (LSA - 64 bits) shows that the CAD tool was able take advantage of the LSA organization to generate a better performance circuit for this technology. It also shows that the 32-bit design for FPGAs could be further improved.

For small precision values, the use of standard on-line design is more adequate than the LSA on-line. The LSA organization consumes more area than the standard design, 21% more area when the operand's precision is 64 bits. In fact, the precision of the LSA organization for 64 bits is in fact 66 bits, given that $n = 5(M-1)+6$, where M is the number of modules used, according to Eq. 8. So, the extra area for the LSA also accounts for these two extra bits, and should be a little bit less if exactly 64 bits of precision were considered in the design.

The reader should consider that these results were obtained without any manual intervention in the synthesis process. We believe that more performance can be extracted from this design. In particular, if an optimal module is created, it can be made available as a library cell which may then be used to create on-line dividers of any desired precision. It would be very suitable for VLSI design since the modules are fixed in size and have only local connections.

7 Conclusion

The use of LSA organization in the radix-2 on-line modules was first proposed in [8] for on-line multiplication and continues in this work for on-line division. For the best of the authors' knowledge there is no similar work done by other researchers. The experimental results obtained for both ASIC and FPGA technologies confirmed the effectiveness of the approach to overcome the excessive delay imposed by large fanout values in the standard on-line design. It is appropriate to handle large operand's precision. If necessary, LSA modules may span over multiple chips since the LSA organization provides the means to avoid having the off-chip communication delay affect the computation cycle time. The discussion of this feature can be found in [5]. The LSA on-line divisor is suitable for VLSI.

Radix-4 division implementations are common for conventional division, but not to on-line division. An algorithm was presented in [12]. It is expected that larger complexity of the radix-4 design will result in a longer cycle time, what may reduce the contribution of the fan-out delay on the overall clock cycle. The advantages obtained with the LSA organization may be less noticiable, but they will be there. This subject must be further investigated.

Acknowledgments

The authors would like to thank the International Student Department of UAS Offenburg for partially supporting this project.

References

[1] J. Bruguera and T. Lang. 2-D DCT Using On-Line Arithmetic. In *IEEE Int. Conference on Acoustics Speech and Signal Processing*, pages 3275–3278, Detroit; MI, May 1995.

[2] M. D. Ercegovac. On-line Arithmetic: An Overview. In *Real Time Processing VII*, volume 495. SPIE, 1984.

[3] M. D. Ercegovac and T. Lang. On-line Scheme for Computing Rotation Factors. *Journal of Parallel and Distributed Computing*, (5):209–227, 1998.

[4] H. T. Kung. Why Systolic Architectures? *Computer*, 1982.

[5] M. Louie. *Variable Precision Arithmetic with Lookup Table Based Field Programmable Gate Arrays.* PhD thesis, UCLA, 1994.

[6] M. E. Louie and M. D. Ercegovac. On Digit-Recurrence Division Implementations for Field Programmable Gate Arrays. In *Proc. of the* 11^{th} *Symposium on Computer Arithmetic*, pages 202–209, Canada, June 29- July 2 1993.

[7] A. F. Tenca and M. D. Ercegovac. Design of high-radix digit slices for online computations. In *Proc.of the SPIE Int. Conference on High-speed computing, digital signal processing, and filtering using reconfigurable logic*, pages 14–25, Boston; MA, Nov. 1996.

[8] A. F. Tenca, M. D. Ercegovac, and M. E. Louie. Fast On-line Multiplication Units using LSA Organization. In Frnaklin T. Luk, editor, *SPIE - Advanced Signal Processing Algorithms, Architectures, and Implementations IX*, pages 74–83, Denver, Colorado, 19-21 July 1999.

[9] A. Tisserand and M. Dimmler. FPGA implementation of real-time digital controllers using on-line arithmetic. In *Lecture Notes in Computer Science*, number 1304, pages 472–481. Springer, 1997.

[10] K. S. Trivedi and M. D. Ercegovac. On-line Algorithms for Division and Multiplication. *IEEE Trans. on Computers*, C-26(7):681–687, 1977.

[11] P. K.-G. Tu. *On-line Arithmetic Algorithms for Efficient Implementation.* PhD thesis, University of California, Los Angeles, Sept 1990.

[12] P. K.-G. Tu and M. D. Ercegovac. A Radix-4 On-line Division Algorithm. In *IEEE 8th Symposium on Computer Arithmetic*, pages 181–187, 1987.

[13] P. K.-G. Tu and M. D. Ercegovac. Design of On-line Division Unit. In *IEEE 9th Symposium on Computer Artihmetic*, pages 42–49, 1989.

[14] D. M. Tullsen. A Very Large Scale Integration Implementation of an On-line Arithmetic Unit. Master's thesis, Univeristy of California, Los Angeles, 1986.