On Digit-Recurrence Division Implementations for Field Programmable Gate Arrays

Marianne E. Louie and Miloš D. Ercegovac
Computer Science Department
University of California, Los Angeles, CA 90024

Abstract

The flexibility of Field Programmable Gate Arrays (FPGAs) can provide arithmetic-intensive programs with the benefits of custom hardware but without the high cost of custom silicon implementations. Efficient mappings are key to fast arithmetic implementations on FPGAs. This paper explores a process for developing such mappings with lookup table based FPGAs. The development process is illustrated with SRT division and the Xilinx XC4010 FPGA. With this mapping process we create a linear sequential array design that avoids the common problem of large fanout delay in the critical path. This approach has a cycle time independent of precision while requiring approximately the same number of logic blocks as a conventional implementation.

1 Introduction

Use of application specific integrated circuits (ASICs) enhances the processing speed of almost every arithmetic intensive application. Unfortunately, high development costs and the long time to fabrication often prevents their creation. Such applications have thus been forced to rely on slower performance combinations of software and off-the-shelf components of standard precisions. Now, the flexibility of field programmable gate arrays (FPGAs) allows the rapid development of high performance custom hardware. FPGAs trade area and some processing speed for flexibility. By selecting arithmetic algorithms suited to the FPGA technology and subsequently applying optimal mapping strategies, high performance FPGA implementations can be developed.

Some existing work in the area of arithmetic processors and FPGAs includes the Flexible Processor Cell [16] and Programmable Active Memories (PAM) [1][2]. Performance estimates for some mathematical applications are given in [2] and an arithmetic processor is proposed in [16], but neither describes the process of mapping specific algorithms. In general there are few results on arithmetic implementations with FPGAs [15].

Current logic synthesis and technology mapping tools (e.g., MIS [12]) optimize and implement the boolean expressions defining a given algorithm. The individual expressions precisely explain the computation of individual bits, but do not provide information about the algorithm as a whole. As a result, these tools cannot identify bits that are unnecessary to implement nor can they make further optimizations based on global information about the algorithm. This paper thus explores the process of developing efficient implementation variations based on information at the algorithm level. Specifically, this study focuses on implementing digit-recurrence division [6] on lookup table based FPGAs. Division is one of the more difficult of the primitive arithmetic operations due to the constraints imposed by iterative steps of quotient digit selection and computations based on that selection. We propose *bit reduction* as a fundamental element of our mapping strategy. Bit reduction lowers the number of binary inputs to combinational logic to permit designs of smaller area and logic depth.

We also create a linear sequential array design in which segments of residuals of successive iterations are computed in parallel. The original concept for the linear sequential array was proposed in [4] without implementation details. By using bit reduction, the implementation becomes simple.

2 Division Overview

In digit-recurrence type division [6][13], the quotient is obtained digit-by-digit in redundant form so that (1) each iteration is processed fast, (2) the quotient digit selection is simplified, and (3) the total error is reduced to less than 2^{-j} at each step j. In this paper we use the SRT division [13, 6] specified below.

Recurrence:

$$w[j+1] = 2w[j] - dq_{j+1}$$

where

$w[j]$ = residual at step j; $\quad w[0]$ = dividend
d = divisor; $\quad 0 \le |w[0]| < d; \quad 1/2 \le d < 1$
$q = \sum_{i=1}^{m} q_i 2^{-i}$

Quotient digit selection function:

$$q_{j+1} = \begin{cases} 1 & \text{if } 0 \le 2\hat{w}[j] \le \frac{3}{2} \\ 0 & \text{if } -\frac{1}{2} = 2\hat{w}[j] \\ -1 & \text{if } -\frac{5}{2} \le 2\hat{w}[j] \le -1 \end{cases}$$

where $\hat{w}[j]$ is a 4-bit estimate of $w[j]$.

1063-6889/93 $03.00 © 1993 IEEE

Configuration	Delay (ns)
Table F or G to Output X or Y	7.0
Tables F and/or G through H to Output X or Y	9.8
Flip Flop Input to Output (Setup and Clock-to-Q)	5.6

Table 1: Throughput Delay of a CLB

3 FPGA Overview

In this study, the division algorithms are mapped to the lookup table based FPGAs of the Xilinx XC4010 [18]. This overview presents the configuration and some of the timing details of the XC4010 speed grade -7 [17]. The faster and more recent speed grade -5 timings are not presented in this overview, but a later section provides both the division timings at speed grade -7 as well as an extrapolated speed grade -5 performance derived with the XACT4000 Design Software.

The XC4010 consists of a 20x20 grid of configurable logic blocks (CLBs) interconnected horizontally and vertically by programmable routing lines. Several long lines provide a reduced delay of large fanouts for each row and column of interconnects. The minimum interconnect delay, 1.9ns, occurs between adjacent CLBs. Interconnection delays increase with increasing fanout and routing distance. In a simplified diagram, the configuration of each CLB is shown in Figure 1. The delay [17] of each CLB is shown in Table 1.

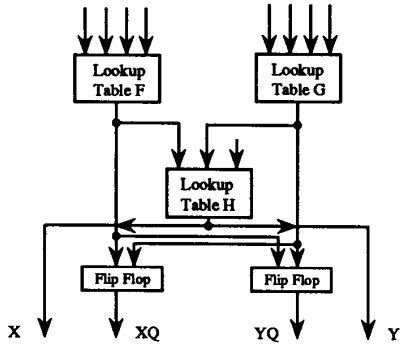


Figure 1: Simplified Diagram of an XC4010 CLB

4 Division and FPGAs

The proper selection of radix and algorithm are critical factors in efficiently matching division to a given set of FPGA characteristics. A higher radix will have greater combinational logic depth but fewer required iterations. Each of the digit-recurrence division algorithms differs primarily in the number of bits required to compute the following basic steps:

(1) select a quotient digit, q_{j+1},

(2) form a multiple of the divisor, dq_{j+1}, and

(3) compute the next residual, $w[j+1]$.

A choice of the best radix for a lookup table based FPGA is determined by comparing the maximum function size of the division algorithm (based on number of input bits) with the function size of a single logic block. A composition of switching functions maps most efficiently into k-input lookup tables when each function has no more than k inputs. With more than k inputs a function requires at least two lookup tables, and a composition of n such functions may require more than $2n$ logic blocks. By this rationale, the best division algorithm for k-input logic blocks is the one with the largest radix having steps of a maximum of k input bits. SRT (radix 2) division is the best choice for the XC4010 under this criteria. SRT requires only four bits for the quotient digit selection function whereas higher radices need at least six bits [6]. SRT also uses three bits for the generation of each bit of the divisor multiple while higher radices use at least five. The computation of $r\hat{w}[j]$ for all radices requires more input bits than the general 5-input lookup table of an XC4010 CLB, but this computation has the fewest number of input bits with radix 2.

The long FPGA interconnection delay for large fanouts also affects the division algorithm. For fanouts of 20 bits, this delay is equivalent to that of a lookup table, and the delay grows dramatically as the fanouts increase. If the implementation performs the basic steps in the given order, the long fanout delay of q_{j+1} will always lie in the critical path. This can be avoided by implementing the division algorithm with quotient digit prediction [5] to produce q_{j+2} during step j. Thus, for small precisions (up to 24 bits on the XC4010), the fanout delay of q_{j+1} is removed from the critical path. As the fanout grows for larger precisions, the scheme presented in Section 5.4 is also needed to prevent the larger fanout delay from entering the critical path.

5 Efficient Arithmetic Mappings

The process of mapping a given algorithm involves developing efficient variations of the logic in the critical path. The proposed mapping process consists of (1) a reduction in the number of bits input to the combinational logic, (2) a decomposition of the logic into subfunctions, and (3) a synthesis into expressions suited to the characteristics of the target FPGA. Since each of the division steps applies to the most significant bits of the residual (those involved in computing a residual estimate), the mapping process focuses on these bits.

5.1 Bit Reduction

Generally, functions that depend on many input bits require more logic blocks in their implementation

and hence have a larger logic depth. A reduction in the number of input bits creates new smaller functions for the original algorithm, permitting a mapping to fewer blocks and reducing the overall logic depth.

For SRT division with carry save adders and quotient digit prediction, the bits of the scaled residual ($2w[j]$) that are involved in quotient digit selection are:

$$\begin{array}{cccccccc} s_{-1} & s_0 & . & s_1 & s_2 & s_3 \\ c_{-1} & c_0 & . & c_1 & c_2 & c_3 \end{array}$$

Fourteen bits are used by the combinational logic to form q_{j+2} in the critical path; these consist of s_i, c_i where $-1 \leq i \leq 3$, d_2, d_3, q^s_{j+1} (sign bit of q_{j+1}), and q^m_{j+1} (magnitude bit of q_{j+1}). An implementation on the XC4010 does not explicitly generate $-q_{j+1}d$ because $s_i + c_i - q_{j+1}d_i$ can be computed in one logic block. However, to simplify the explanations, symbolically let:

$$D_i = \begin{cases} d_i & \text{if } q_{j+1} = -1 \\ 0 & \text{if } q_{j+1} = 0 \\ \bar{d}_i & \text{if } q_{j+1} = 1 \end{cases}$$

An implementation on the XC4010 (three logic levels with delay of two 5-input tables and one 4-input table) is shown in Figure 2 where:

(1) $2b_{i+1} + a_i = s_i + c_i + D_i$

(2) $2e_{-1} + e_0 = \left(\sum_{i=-1}^{0} [(a_i + b_i)2^{-i}] \right) \bmod 4$

(3) $(4f_0 + 2f_1 + f_2)2^{-2} = \sum_{i=1}^{2} [(a_i + b_i)2^{-i}]$

(4) $q_{j+2} = f(e_{-1}, e_0, f_0, f_1, f_2)$

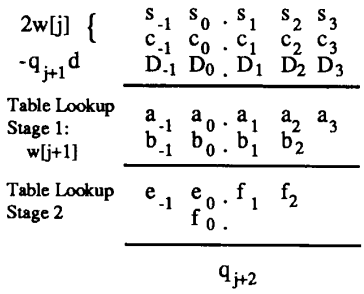


Figure 2: SRT Division with Carry Save Addition and Quotient Digit Prediction

Bit reduction is applied to the implementation to achieve a more efficient design. Bit reduction primarily results from the strategic placement of the registers such that a minimal number of bits either directly enter or generate data used in the critical path. Repositioning the registers is similar to a retiming [9] but at the algorithm level. With this strategy, a reduced number of input bits can be obtained by placing the registers after generating the more compact residual estimate (the carry assimilation of $\sum_{i=-1}^{1}(s_i + c_i)2^{-i}$ of Figure 2) rather than storing all of the sum and carry bits. With this variation, the implementation is not forced to explicitly generate each individual sum and carry bit. The resulting simplified logic not only reduces the area by computing fewer output bits but also reduces the delay by manipulating fewer input bits to obtain the output. The stored residual bits for selection after optimally placing the registers are:

$$\begin{array}{cccccc} \hat{w}_{-1} & \hat{w}_0 & . & \hat{w}_1 & s_2 & s_3 \\ & & . & & c_2 & c_3 \end{array}$$

In this case the critical path utilizes a total of eleven input bits, the above seven bits, d_2, d_3, q^s_{j+1} and q^m_{j+1} to form the estimate of the residual. This implementation consists of three logic levels with the delay of one 5-input and two 4-input lookup tables.

Due to the shifting of the residual in a digit-recurrence algorithm, some bits that are not in the current iteration's critical path will be in the next iteration's critical path. Early processing of those bits will not affect the selection operation in the current step, but it will further reduce the number of bits involved in the next iteration. The early computation of $\sum_{i=2}^{3}[(s_i + c_i)2^{-i}]$ to form $\hat{c}_1\hat{s}_2\hat{s}_3$ results in registers for:

$$\begin{array}{cccccc} \hat{w}_{-1} & \hat{w}_0 & . & \hat{w}_1 & \hat{s}_2 & \hat{s}_3 \\ & & . & \hat{c}_1 & & \end{array}$$

The estimate of the residual thus becomes a function of one fewer bits. This is enough to reduce the delay on an XC4010 implementation by one logic level. In this variation, the above six bits, d_2, d_3, q^s_{j+1} and q^m_{j+1} (for a total of ten bits) are needed to form the estimate of the residual in the critical path.

Further bit reduction can be obtained by eliminating redundant bits. With quotient digit prediction, the redundant bits are uniquely determined by the quotient digit and some bits of the estimate. For SRT the most significant bit, $\hat{w}_{-1}$, can be eliminated because it is deduced from q_{j+1}, $\hat{w}_0$, and $\hat{w}_1$. The resulting reduced inputs are:

$$\begin{array}{cccccc} \hat{w}_0 & . & \hat{w}_1 & \hat{s}_2 & \hat{s}_3 \\ & . & \hat{c}_1 & & \end{array}$$

Although this variation does not further reduce the delay of SRT, it does reduce the implementation area because fewer bits are computed. The logic steps for SRT with bit elimination when implemented on the XC4010 is shown in Figure 3 where:

(1) $m_1 = (\hat{w}_1 + c_1 + D_1) \bmod 2$

(2) $(4k_1 + 2k_2 + k_3)2^{-3} = \sum_{i=2}^{3}[(s_i + D_i)2^{-i}]$

(3) $2p_3 + p_4 = (s_4 + c_4 + D_4)$

(4) $u_4 = \lfloor (s_5 + c_5 + D_5)2^{-1} \rfloor$

(5) $(F_1, F_2) = f(q^s_{j+1}, q^m_{j+1}, \hat{w}_0, \hat{w}_1, c_i)$ representing an encoding as described in Section 5.3.

(6) $(4\hat{c}''_1 + 2\hat{s}''_2 + \hat{s}''_3)2^{-4} = k_3 2^{-3} + u_4 2^{-4} + \sum_{k=3}^{4} p_k 2^{-k}$

(7) $\hat{w}''_0 = (m_1 + k_1) \bmod 2$

(8) $\hat{w}''_1 = k_2$

(9) $q^s_{j+2} = f(F_1, F_2, k_1, k_2)$

(10) $q^m_{j+2} = f(F_1, F_2, k_1, k_2)$

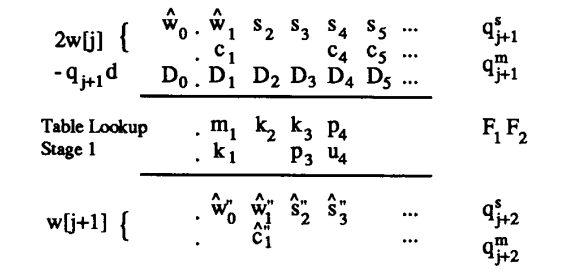


Figure 3: SRT Division after Bit Reduction

5.2 Building the Dependency Graph

After completing bit reduction, mapping to a specific FPGA begins. Boolean minimization and technology mapping tools for lookup table based FPGAs (e.g., MIS-PGA[12], Chortle[7], and Dagmap[3]) require a decomposition of the boolean expressions into k-input (or smaller) functions where k is the number of inputs to the CLBs. Typically, these tools prefer *and-or-not* networks with gate fan-in of two because (1) techniques for minimizing boolean algebra are well-known, and (2) smaller functions allow greater possibilities of grouping several expressions into a single lookup table. A disadvantage of decomposition into *and*, *or*, *not* logic is that common reconvergent operations may not be detected causing a design to be inefficient in area. Reconvergent operations decompose into several parallel functions, the results of which are reconverged into a single output. An example is *xor* implemented as $g(f_1(a,b), f_2(a,b))$. Since the *xor* operation is fundamental to arithmetic algorithms, current minimization and mapping tools perform poorly with arithmetic. Current minimization tools are designed to handle random logic rather than the regular logic functions of arithmetic. Hence, the tools do not take advantage of known reconvergent operations and algorithmic structure.

Unlike typical mappers, the mapping methodology that we use does not create a directed acyclic graph (DAG) of two-input *and-or-not* gates. Rather, it creates a DAG describing the relationships among function variables. An example DAG of function variables for a 3-bit carry propagate addition is shown in Figure 4. Each node in the graph represents a function which may be any boolean expression. The node variables list the function inputs. The node's output corresponds to the computed result after applying the function to the inputs. The output variable is then labeled and used as an input to its successor node. This proposed DAG creates a more compact tree representation and reveals the structure of the arithmetic algorithm for an efficient mapping. Hence, a simple reconvergent function is kept in the same logic block.

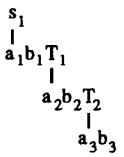


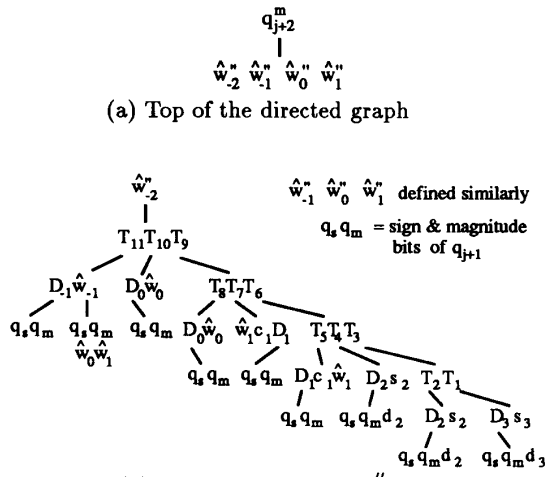
Figure 4: Directed graph for sum bit s_1 of $\sum_{i=1}^{3}(a_i + b_i)2^{-i}$

A lookup table can accommodate in the same area and delay either a complex switching function of many nonminimizable minterms/maxterms or a simple boolean expression; the only limiting factor is the number of inputs. The proposed dependency graph more closely matches this feature of lookup tables in that both complex and simple boolean expressions of the same number of inputs can be represented with the same simple graph structure. In arithmetic algorithms, the relationships among small groups of variables tend to form complex expressions. Thus, the simplified dependency graph quickly presents small groups of variables that are already known to be interrelated with a more complex function. With DAGs of two-input *and-or-not* gates, the relationship within these groups of variables would not be as obvious.

The details for generating the dependency graph are presented in [10]. For the purpose of mapping division, the directed graph (without the corresponding boolean expressions) for the magnitude bit of q_{j+2} is shown in Figure 5. This dependency graph is used to determine the encoding scheme of F_1, F_2 as shown earlier in Figure 3.

5.3 Mapping the Dependency Graph

After the dependency graph is complete, the node expressions are synthesized into new expressions suited to the FPGA block. Our mapping strategy treats the directed graph as an unbalanced tree. Mapping is performed from the bottom up, and tree balancing (creation of parallel functions, each of lesser depth) is performed as needed.



(a) Top of the directed graph

(b) Directed graph for $\hat{w}''_{-2}$

Figure 5: Graph for the Magnitude Bit of q_{j+2}

We employ a combination of three techniques — bin packing [8], Roth-Karp decomposition [14], and multiplexing. Bin packing treats each expression as a package, where the lookup table is viewed as a bin. The maximum number of connected packages (subexpression nodes) are packed into the same bin (lookup table). If several packages share variables, the variable is counted as occupying only one input space in the bin. Bin packing is especially efficient for arithmetic because a single variable tends to affect only local sets of interconnected nodes.

Roth-Karp decomposition [14] separates a single expression of many variables into a composition of expressions, each dependent on a subset of the original variables as illustrated by:

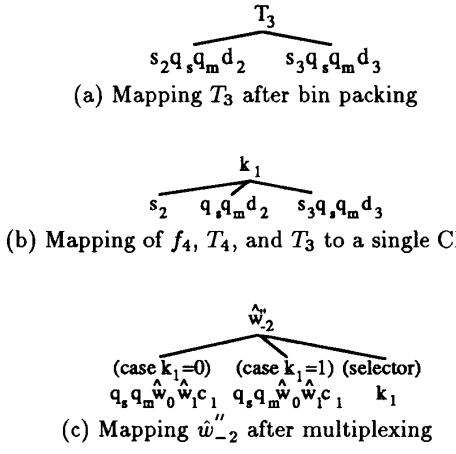
$$f(a_0, a_1, a_2, a_3, \ldots, a_n) = g(h_1(a_i, a_j, a_k), h_2(a_i, a_j, a_k), a_l) \qquad l \neq i, j, k$$

where f, g, h_1 and h_2 are function definitions. Many logic synthesis and technology mapping tools such as [12] and [3] use Roth-Karp decomposition.

Roth-Karp decomposition can be viewed as a coding of a subset of the input variables. In the mapping of the dependency graph, Roth-Karp decomposition is employed when the variables of one or more input nodes are subsets of another input node. The decomposition attempts to create a code for the inputs, thus reducing the number of input nodes and the required number of logic blocks. The path delay may also be reduced due to the fewer number of inputs. The logic to generate the coding scheme is accommodated in the logic function of the involved nodes.

We propose multiplexing as an additional approach towards reducing the graph depth. Multiplexing is primarily useful for unbalanced directed graphs. This often occurs, for example, in carry-propagate addition. In multiplexing, a single variable having one or more ancestor nodes is separated from the graph. Two cases then arise for the remaining skewed leg of the graph: (1) the computation if the separated variable has a value of 1 and (2) the computation if the separated variable has a value of 0. Both cases are computed simultaneously with the evaluation of the actual variable, and a selection/multiplexing of the correct case is performed afterward. The cost of implementation is (1) logic repetition for the multiple cases and (2) the extra logic step of multiplexing at the end. However, for some arithmetic functions that are very serial in nature and for which reduced logic depth is a key goal, multiplexing offers a solution.

The algorithm details for applying the three techniques are presented in [10]. A summary of the mapping of $\hat{w}''_{-2}$ (dependency graph of Fig. 5) is shown in Figure 6. By bin packing, T_3 is mapped to a single CLB (Fig. 6a). Since T_2 and T_4 have identical input variables, we then attempt to map T_4 and $T_3 = f_3(T_2, T_1)$ together. By Roth-Karp decomposition, $T_6 = f_6(T_5, f_4(T_4, T_3))$ where k_1 (of Fig. 3) $= f_4(T_4, T_3) = T_4$ *or* T_3. Again by Roth-Karp decomposition, k_1 including T_4 and T_3 can be mapped to one CLB (Fig. 6b). The remaining intermediate variables, T_5 through T_{11}, use only six different inputs among which is k_1. Hence multiplexing reduces the delay to two CLBs (Fig. 6c). The mapping of $\hat{w}''_{-1}$, $\hat{w}''_0$, and $\hat{w}''_1$ is performed similarly to that of $\hat{w}''_{-2}$.



(a) Mapping T_3 after bin packing

(b) Mapping of f_4, T_4, and T_3 to a single CLB

(c) Mapping $\hat{w}''_{-2}$ after multiplexing

Figure 6: Mapping Process for $\hat{w}''_{-2}$ to the XC4010

Next consider the mapping of the magnitude bit of q_{j+2} where (1) q_{j+2} is a function of $\hat{w}''_{-2}$, $\hat{w}''_{-1}$, $\hat{w}''_0$, and $\hat{w}''_1$, and (2) $\hat{w}''_{-2}$, $\hat{w}''_{-1}$, and $\hat{w}''_0$ are each functions of $\hat{w}_0$, $\hat{w}_1$, q_s, q_m, c_1, and k_1. Since each $\hat{w}''_i$ is

derived from the same input variables, Roth-Karp decomposition can find a boolean expression to directly code q_{j+2} from $\hat{w}_0$, $\hat{w}_1$, q_s, q_m, c_1, and k_1 instead of computing $\hat{w}''_{-2}$, $\hat{w}''_{-1}$, and $\hat{w}''_0$, and $\hat{w}''_1$ individually. The final mapping of the magnitude bit of q_{j+2} is shown in Figure 7. The resulting expression uses a conditional encoding of $\hat{w}''_{-2}$, $\hat{w}''_{-1}$, and $\hat{w}''_0$ where k_1 and $k_2 = \hat{w}''_1$ provide the value of the condition. The coding scheme, where (h_{-1}, h_0, h_1) corresponds to $c_1 2^{-1} + \sum_{i=-1}^{1}[(\hat{w}_i - q_{j+1}d_i)2^{-i}]$, is:

$F_1F_2 = f(q^s_{j+1}, q^m_{j+1}, \hat{w}_0, \hat{w}_1, c_1) =$

00 if $h_{-1}h_0h_1 = 000$ or 001
(where $q_{j+2} = 1$;
$h_{-1}h_0h_1 \neq 010$ or 011)

01 if $h_{-1}h_0h_1 = 111$
(where $q_{j+2} = 0$ if $k_1 = 0$, $k_2 = 1$;
$q_{j+2} = 1$ if $k_1 = 1$;
$q_{j+2} = -1$ otherwise)

10 if $h_{-1}h_0h_1 = 110$
(where $q_{j+2} = 0$ if $k_1 = 1$, $k_2 = 1$;
$q_{j+2} = -1$ otherwise)

11 if $h_{-1}h_0h_1 = 100$ or 101
(where $q_{j+2} = -1$)

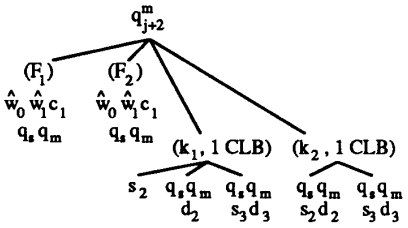


Figure 7: Mapping of the Magnitude Bit of q_{j+2} to the XC4010 in 2 logic levels

5.4 A Linear Sequential Array (LSA) Design

As the precision of the division implementation grows, the fanout delay of q_{j+1} also grows and eventually becomes part of the critical path. For the SRT division variation on the XC4010, the fanout delay of q_{j+1} enters the critical path with precisions that are larger than 24 bits. A linear sequential array (LSA) [4] (Fig. 8) provides pipelining rather than the large fanout broadcasting of q_{j+1}. The LSA requires a comparable amount of hardware as a non-array implementation. For our division variation, it actually reduces the area by 5% for the lower weighted bits that are not involved in quotient digit selection.

Each module of the LSA design encompasses a segment of four weights in the bits of the residual. The modules may be attached to the division design at any point where the sum and carry bits of the residual are

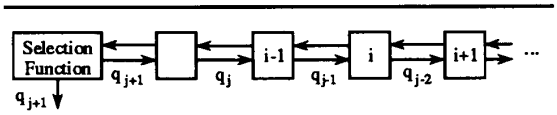


Figure 8: The Linear Sequential Array Organization

computed and latched. Hence, the LSA implementation serves as an extension and/or replacement of the lower weighted residual bits of the existing design. Figure 9 shows the existing design described in terms of conventional 4-bit modules while Figure 10 illustrates their replacement with LSA modules.

Bit reduction in the form of strategic register positioning easily derives the LSA implementation. In this case, progressively delaying the computation of successive residual bits enables the pipelined distribution of the quotient digit. We begin the new design by repositioning the least significant bits of the existing residual. The objective is to maximally delay the generation of these bits while allowing the registers for the more significant bits to remain the same. The variable c_{i-1} is delayed along with s_i and c_i because c_{i-1} and s_i are generated from the same input variables. The precision of the LSA is then gradually extended by strategically positioning the inputs used for computing each of the delayed bits. Again the objective is to find the maximally delayed time of computation without altering the generation of the existing bits. This process continues until the desired extended precision is obtained. The resulting configuration for the LSA is shown in Figure 11 where the superscript on each variable denotes its computational step. The LSA also explicitly computes each D_i in Fig. 11.

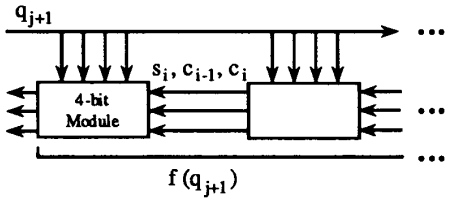


Figure 9: Conventional Modules

An implementation of the linear sequential array requires two logic levels, each of a 4-input lookup table delay. Fanout of data between stages of the pipeline is also small and locally distributed to achieve low interconnection times. As a result, the delay of the LSA is always less than the computation delay of the most significant bits of the residual (which requires two logic levels – one 5-input and one 4-input lookup table). Therefore, due to its pipelined nature, LSA stages and hence precision may be added freely without increasing the overall throughput delay of the division design.

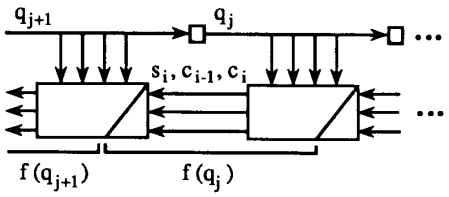


Figure 10: LSA Modules

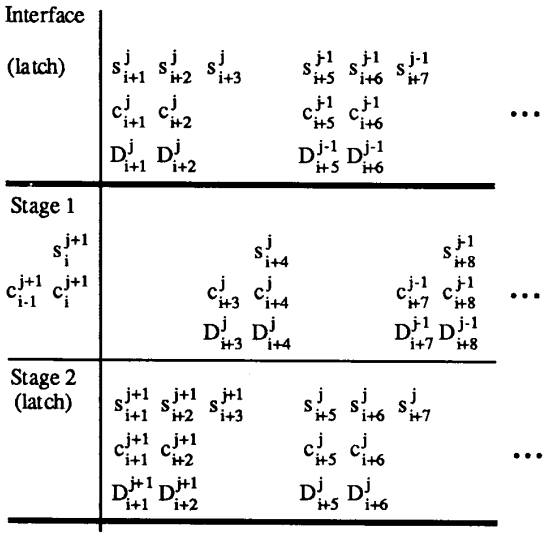


Figure 11: The LSA Design

Thus high precision implementations will experience the same cycle time as lower precision designs. The LSA is also applicable to other technologies and has an iteration delay of approximately two full adders.

Regarding area, every 4 bits of added precision in the LSA requires 9.5 CLBs for implementation. This includes overhead for the operand registers as well as initialization logic. For the SRT variation without the linear sequential array, the lower weighted bits of the residual require 10 CLBs for every 4 bits of precision, including initialization overhead and operand registers.

5.5 Implementation Results

The SRT division variation of Figure 3 is implemented with the XC4010 speed grade -5 and -7 (preliminary notes). The speed grade -7 implementations have been simulated with the Xilinx software and Viewlogic's Workview while the speed grade -5 timings were extrapolated with the Xilinx software. Table 2 shows the area and timing details of a single precision (24 bit mantissa only) implementation and compares these results to a baseline estimate and an earlier SRT division design [11] that uses the XC4010 fast carry propagate adder (CPA) hardware [18]. Neither implementation incorporates the LSA technique because the fanout delay of q_{j+2} does not enter the critical path for 24-bit mantissas. The division design using the XC4010 fast CPA also incorporates some bit reduction optimizations. However, this implementation explicitly computes $2\hat{w}[j+1]$ with the fast CPA. In comparing the two division designs, the timing of the new variation represents a 28% reduction in delay over the previous SRT design. Although a standalone CPA has the fastest implementation with the special CPA feature, allowing the CPA algorithm step to be mapped with other logic steps into lookup tables can result in a more efficient design.

Division Variation	No. of CLBs	Speed Grade -5 Cycle Time	Speed Grade -7 Cycle Time
With Bit Reduction	52	19.0 ns	28.5 ns
With Fast CPA [11]	48	26.6 ns	39.6 ns
Baseline Estimate	42	36.0 ns	54.8 ns

Table 2: SRT Variations for Single Precision Mantissas

The baseline estimate represents an approximate design with quotient digit prediction but no further optimizations. The logic steps include:

(1) computation of $q_{j+1}d$,

(2) carry save addition to form $w[j+1]$,

(3) generation of $2\hat{w}[j+1]$ with a 4-bit CPA using the XC4010 fast carry propagate adder hardware, and

(4) quotient digit selection.

Comparing the baseline estimate with the our new variation shows that bit reduction optimizations can improve a design's performance by almost twice.

The LSA scheme allows the designs to be extended to any precision while maintaining the same cycle time. This was verified in the SRT variation with bit reduction where the original design was extended from 20 to 32 bits of precision. The new design was simulated with both the Xilinx software and Viewlogic's Workview.

6 Summary and Conclusion

Generating an efficient lookup table based FPGA implementation for arithmetic requires (1) the selection of an algorithm suited to the target technology,

(2) the creation of a suitable variation of that algorithm for the target characteristics, and (3) an efficient mapping approach. A well-matched algorithm is recognized by the simple decomposition of its intermediate steps into expressions of k variables or less, where k is the number of inputs in the lookup tables. Bit reduction then enables a more efficiently mapped variation of that algorithm. We describe the arithmetic structure in terms of a dependency graph to provide a clear and simple description of the relationships among variables. For technology mapping, we utilize a combination of bin packing, Roth-Karp decomposition, and multiplexing. Additionally, use of bit reduction techniques allows the simple development of a linear sequential array so that the large fanout of the quotient digit is removed from the critical path. Our design method has been used to obtain an improvement of about 28% in step delay compared to an earlier design that utilizes some optimizations. It is estimated that our optimizations can create a design having almost twice the performance as a baseline design, where the baseline does not feature the proposed bit count reductions and enhancements in technology mapping.

Acknowledgements

This work has been supported in part by the State of California MICRO Grant "Field-Programmable Application-Specific Processor Arrays" and Xilinx, Inc. We also thank Jason Cong for his comments on technology mapping.

References

[1] P. Bertin, et. al., "Introduction to Programmable Active Memories," DEC Research Report No. 3, Paris Research Laboratory, 1989.

[2] P. Bertin, et. al., "Programmable Active Memories: A Performance Assessment," Proc. First Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays, Feb. 16-18, 1992, Berkeley, CA.

[3] J. Cong, A. Kahng, K. Chen, P. Trajmar, "Graph Based FPGA Technology Mapping for Delay Optimization," Proc. First Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays, Feb. 16-18, 1992, Berkeley, CA., pp. 77-82.

[4] M.D. Ercegovac, "On-Line Arithmetic: An Overview," Proc. SPIE Vol. 495 – Real Time Signal Processing VII, 1984, pp.86-92.

[5] M.D. Ercegovac, T. Lang, "A Division Algorithm with Prediction of Quotient Digits," Proc. 7th IEEE Symposium on Computer Arithmetic, 1985, pp. 51-56.

[6] M.D. Ercegovac, T. Lang, *Digit-Recurrence Algorithms and Implementations for Division and Square Root*, to be published by Kluwer Academic Publishers, 1993.

[7] R. Francis, J. Rose, K. Chung, "Chortle: A Technology Mapping Program for Lookup Table Based Field Programmable Gate Arrays," Proc. 27th ACM/IEEE Design Automation Conference, 1990, pp. 613-619.

[8] R. Francis, J. Rose, Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," Proc. 28th ACM/IEEE Design Automation Conference, June 17-21, 1991, San Francisco, CA, pp. 227-233.

[9] C. Leiserson, F. Rose, J. Saxe, "Optimizing Synchronous Circuitry by Retiming," Third Caltech Conference on VLSI, 1983.

[10] M.E. Louie, "Computer Arithmetic and Field Programmable Gate Arrays," Ph.D. Thesis in progress.

[11] M.E. Louie, M.D. Ercegovac, "Mapping Division Algorithms to Field Programmable Gate Arrays," Proc. 26th Asilomar Conference on Signals, Systems, and Computers, Oct. 26-28, 1992, Pacific Grove, CA, pp. 371-375.

[12] R. Murgai, N. Shenoy, R. Brayton, A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Lookup Up Architectures," Proc. 1991 IEEE Int'l Conference on Computer-Aided Design, Nov. 11-14, 1991, Santa Clara, CA, pp. 564-567.

[13] J.E. Robertson,"A New Class of Digital Division Methods," IRE Trans. on Electronic Computers, Sept. 1958, pp 88-92.

[14] J. Roth, R. Karp, "Minimization Over Boolean Graphs," IBM Journal, April 1962, pp. 227-238.

[15] M. Shand, P. Bertin, J. Vuillemin, "Hardware Speedups in Long Integer Multiplication," Computer Architecture News, 19(1):106-114, 1991.

[16] A. Wolfe, J.P. Shen, "Flexible Processors: A Promising Application-Specific Processor Design Approach," Proc. of the 21st Annual Workshop on Microprogramming and Microarchitecture, Nov.30-Dec.2, 1988, San Diego, CA, pp.30-39.

[17] Xilinx XACT4000 Design Implementation Software.

[18] Xilinx, "XC4000 Logic Cell Array Family - Technical Data," San Jose, 1990.