21st International Conference on VLSI Design

A Merged Synthesis Technique for Fast Arithmetic Blocks involving Sum-of-Products and Shifters

Sabyasachi Das
Synplicity Inc
Sunnyvale, CA, USA
Email: sabya@synplicity.com

Sunil P. Khatri
Texas A&M University
College Station, TX, USA
Email: sunilkhatri@tamu.edu

***Abstract*— In modern Digital Signal Processing (DSP) and Graphics applications, the arithmetic Sum-of-Products, Shifters and Adders are important modules, contributing a significant amount to the overall delay of the system. A datapath structure consisting of multiple arithmetic sum-of-product, shifter and adder blocks is often found in the timing-critical path of the chip. In this paper, we propose a new operator-level merging technique to synthesize this type of datapath structure. In our approach, we combine the shifting operation with the partial product reduction stage of the sum-of-product blocks. This enables us to implement the functionality of the original design by using only one carry-propagate adder block (instead of two carry-propagate adders). As a result, the timing-critical path of the design gets shortened by a significant percentage and the overall performance of the design improves. Our experimental data shows that the datapath block generated by our approach is significantly faster (13.28% on average) with a modest area penalty (3.24% on average) than the corresponding block generated by a commercially available best-in-class datapath synthesis tool. These improvements were verified on placed-and-routed designs as well.**

I. Introduction

The design complexity and performance requirements of datapath operations implemented in systems on chips has increased considerably over the years. This is especially true in ICs for communication, multimedia and graphic applications, which have highly parallel implementations of signal processing algorithms.

The arithmetic sum-of-products, shifters and adders are some of the most widely used arithmetic datapath operations in modern digital design. The block diagram of the Figure-1 is often seen in modern datapath designs. This design consists of multiple computationally expensive arithmetic sum-of-product (SOP) blocks. The outputs of some of the SOP blocks get shifted by different shift signals, followed by an addition of all the outputs of the shifters, remaining SOPs and some other additive input signals. The critical path of this design goes through an SOP, a shifter and an adder. Since this design requires intensive computations, they incur a significant amount of delay, and therefore tend to be typically found in the timing-critical path of the chip. Developing an efficient architecture for this design structure would reduce the delay of the individual blocks and thereby improve the performance of the IC. Hence there is great interest in generating timing-efficient architecture for this type of datapath structure.

In [1], [2], [3] and [4], the authors presented techniques to

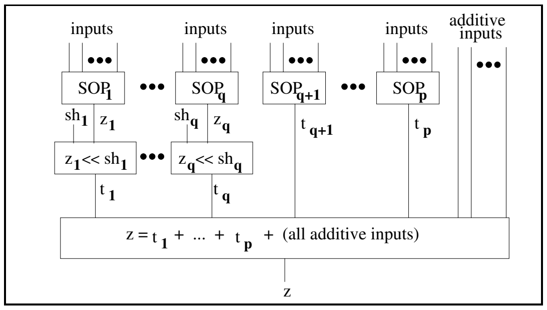

Fig. 1. The generalized block-diagram of our problem statement

emphasize the usefulness of arithmetic sum-of-product (SOP) blocks over a collection of cascaded blocks performing unit operations (like additions, subtractions, multiplications etc). In [5], the critical path delays and hardware complexities of Multiplier-Accumulation units are explored to derive a high performance MAC. In [6], a technique to reduce the partial products in multiplication and sum-of-product units has been proposed. A hybrid compression technique to reduce the delay of SOP has been presented in [7]. The basic architecture for a barrel shifter was proposed in [8]. In [9], a timing-driven decomposition is introduced for fast shifter. The use of dynamic logic for shifter blocks was demonstrated in [10]. Timing-driven layout techniques of shifters were proposed in [11], [12]. A 32-bit rotator/shifter circuit design with short latency was discussed in [13]. Several architectures for performing fast timing-driven two-operand addition are explained in [14], [15] and [18]. A mix of these architectures can be used to synthesize the blocks involving sum-of-products and shifters.

In this paper, we propose an operator-level merging-based technique involving the sum-of-products (SOP) and shifters. In our approach, we combine the shifting operation with the partial product reduction stage of the SOP blocks. This enables us to implement the original design by using only one carry-propagate adder block. As a result, the timing-critical path of the design gets shortened by a significant percentage and the overall performance of the design improves. Our paper addresses a different datapath design issue than what was implemented in the references cited in this section.

We have organized the rest of the paper as follows: Some preliminary information is given in the Section II. In the Section III, we present the definition of the problem we

1063-9667/08 $25.00 © 2008 IEEE
DOI 10.1109/VLSI.2008.112

are addressing in this paper. In the Section IV, we discuss our proposed approach in detail. The experimental setup is explained in the Section V. The Section VI presents the experimental results. Conclusions are drawn in the Section VII.

II. Preliminaries

In this section, we briefly explain the concepts of an arithmetic *Sum-of-Product* (SOP) and a *Shifter* [18].

An example sum-of-product (SOP) block can be expressed by the following Verilog RTL:

assign $z = a * b + c * d + e * f + g + h$;

In this block, there are three product terms ($a*b$, $c*d$ and $e*f$) and two input sum terms (g and h). In general, the number of product and sum terms are arbitrary. After evaluating the product terms (or performing the multiplication operations for each product term), the results of each product term also get added with the input sum terms to produce the final result of the overall SOP block. A sum-of-product block can have any number (including zero) of product terms or sum terms. As a consequence, an SOP block is quite general. It can be used to implement a multiplier. MAC (multiply-accumulator), adder, subtractor, squarer, chain-of-adders or combinations thereof.

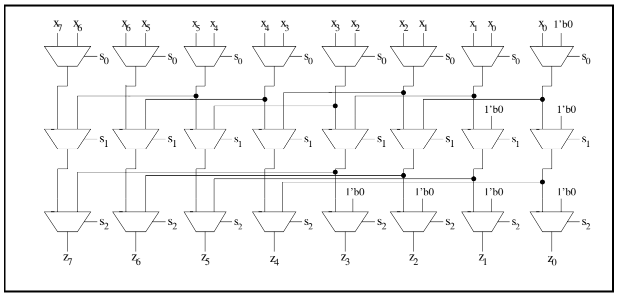

Fig. 2. A Traditional Barrel Shifter with 3-stages

A generalized left-shifter block can be expressed by the Verilog RTL: "assign $z = a << sh$"; where the input *data* signal (a) gets shifted by an input *shift* signal (sh) to produce the output (z) signal. If the data input signal is n-bits wide, then the shift signal is typically $\lceil log_2(n) \rceil$ bits wide. The width of the output (z) is also typically same as the input-width (n).

In a barrel shifter having an n bit wide data signal, the shifter is divided into $\lceil log_2(n) \rceil$ stages, where each stage (i) handles a single shift of 0 or 2^i bits. Each bit of the shift signal controls the functionality of exactly one barrel shifter stage. The input data will be shifted or not shifted by each of the stages in sequence. To implement this, multiplexers (or an equivalent logic circuit constituted using technology library cells) are used in each stage. Figure-2 shows the block-level diagram of a 3-stage barrel shifter. In this diagram, the data input signal (x) is 8-bit wide and the output signal (z) is also 8-bit wide. The shift signal has 3 bits ($\lceil log_2(8) \rceil$ = 3) and the shifter consists of 3 stages only.

III. Problem Definition

In this paper, we propose an operator-level merging-driven technique to synthesize a fast arithmetic block involving sum-of-product (SOP) and shifter blocks. The generalized description of our targeted datapath design block involving SOP and shifter blocks is as follows.

The datapath module consists of p arithmetic sum-of-product (SOP) blocks. Each SOP block takes any number of input vectors. The inputs to each of the p SOPs are arbitrary. The functionality of each of the p SOPs is arbitrary as well. Out of the p outputs of the p SOP blocks, q signals get shifted left by q shifter blocks. The q shift signals (sh_1, sh_2, ..., sh_q) are also arbitrary. Finally, the outputs of q shifters and the outputs of the remaining (p-q) sum-of-products (which do not go through the shifters) and r primary additive input signals (which do not go through any sum-of-products or shifters) get added together by a final adder.

The block diagrams shown in Figure-1 depicts the above-described datapath structure. This type of circuit topology is often seen in modern datapath designs (specially DSP ICs).

One specific example of the above-described generalized datapath design is shown in the Figure-3. In that specific example design, p=2, q=1 and r=2.

IV. Our Approach

Our proposed synthesis approach has four steps. In the following sub-sections, we discuss each step in detail.

A. Generation of Partial Products

For every product term in an SOP block, partial products are generated by performing a bit-wise multiplication between the appropriate bits of the multiplicand and the multiplier. Each partial product is shifted by one or more bits, depending on the bit number of the multiplicand. If PP_i is the i^{th} partial product of the product term $a * b$, and b has n bits, then partial products can be represented by the following expression:

$$PP_i = a * b_i * 2^i \qquad for\ i = 0, 1, ...(n-1)$$

After computing the set of all the partial products corresponding to the product terms, the r additive terms of the SOP expression get included in the set of partial products.

For an SOP with the expression z = a * b + c (where a, b and c each are n-bits wide), there will be a total of n+1 partial products. Out of these, n partial products will be generated by the product term a * b. The remaining single partial product will be the sum-term c. Similarly, for a more complex SOP with the expression z = a * b + c * d + e + f + g (where each input signal is n-bits wide), there will be a total of $2n$+3 partial products.

B. Shifting of Partial Products

As mentioned in the description of the datapath structure targeted in this paper, q of the p outputs of the sum-of-products need to be shifted. In the traditional synthesis approach, all the q SOP blocks get computed and then the q output values get shifted before being fed to the final adder module. In this case, the critical path traverses one SOP block (which in turn has a carry-propagate adder in it), one shifter block and one final carry propagate adder block. It is well-known that the carry propagate adder is one of the most delay-consuming arithmetic operations. Our experimental analysis suggests that the carry propagate adder inside a multiplier (which is one of the most heavily used sum-of-products block), typically contributes about 30% to the total delay of the multiplier [16]. Hence any reduction in the number and structure of the adder stages in our targeted datapath design plays a key role in determining the critical path delay.

In our approach, we implement the shifting of the q SOP outputs by performing the shift operation on *each* of the partial products corresponding to the q sum-of-product blocks. This eliminates the need for carry propagate adders in the q SOP blocks, leading to the improvement of about 30% delay of the individual SOP blocks. Note that, to achieve this improvement in speed of the individual SOP blocks, we require more shifter modules compared to the traditional approach.

To implement a fast shifter, we use the decomposition of the barrel shifter approach [9]. We describe this technique briefly, for completeness. In this approach, two or three stages of the shifter are merged into a single stage whenever feasible. If two stages are merged, the newly created stage is called a *dual merged* stage. On the other hand, if three stages are merged, then the new stage is called a *triple merged* stage.

In the case of *dual merged* stages, let us assume that the stages corresponding to the i^{th} bit and the j^{th} bit of the shift signal s were merged, where $0 \leq i < n$, $0 \leq j < n$ and $i \neq j$. Note that i and j do not require to be two consecutive bits of the shift signal. The newly created *dual merged* stage will perform one of the following four operations:

1) no shifting operation (if s_i=0 and s_j=0)
2) shift by 2^i bits (if s_i=1 and s_j=0)
3) shift by 2^j bits (if s_i=0 and s_j=1)
4) shift by (2^i+2^j) bits (if s_i=1 and s_j=1).

The functionality of each bit-slice of the *dual merged* stage for the left shifter is as follows (data signal is denoted as x):

$$out_q = \overline{\overline{(t_1)} \wedge \overline{(t_2)} \wedge \overline{(t_3)} \wedge \overline{(t_4)}}. \qquad \text{for } 0 \leq q < n.$$

$$\begin{aligned} \text{where } t_1 &= x_q \wedge \overline{s_i} \wedge \overline{s_j} \\ t_2 &= x_{(q-2^i)} \wedge s_i \wedge \overline{s_j} \\ t_3 &= x_{(q-2^j)} \wedge \overline{s_i} \wedge s_j \\ t_4 &= x_{(q-2^i-2^j)} \wedge s_i \wedge s_j \end{aligned}$$

In a similar manner, one can formulate the output equation of each bitslice for *triple merged* stages as well. Let us assume that the stages corresponding to the i^{th} bit, j^{th} bit and the k^{th} bit of the shift signal s are merged, where $0 \leq i < n$, $0 \leq j < n$, $0 \leq k < n$, $i \neq j$, $j \neq k$ and $k \neq i$. The functionality of each bit-slice of *triple merged* stage for a left shifter is as follows:

$$out_q = \overline{\overline{(t_1)} \wedge \overline{(t_2)} \wedge \overline{(t_3)} \wedge \overline{(t_4)} \wedge \overline{(t_5)} \wedge \overline{(t_6)} \wedge \overline{(t_7)} \wedge \overline{(t_8)}}. \quad \text{for } 0 \leq q < n.$$

$$\begin{aligned} \text{where } t_1 &= x_q \wedge \overline{s_i} \wedge \overline{s_j} \wedge \overline{s_k} \\ t_2 &= x_{(q-2^i)} \wedge s_i \wedge \overline{s_j} \wedge \overline{s_k} \\ t_3 &= x_{(q-2^j)} \wedge \overline{s_i} \wedge s_j \wedge \overline{s_k} \\ t_4 &= x_{(q-2^k)} \wedge \overline{s_i} \wedge \overline{s_j} \wedge s_k \\ t_5 &= x_{(q-2^i-2^j)} \wedge s_i \wedge s_j \wedge \overline{s_k} \\ t_6 &= x_{(q-2^i-2^k)} \wedge s_i \wedge \overline{s_j} \wedge s_k \\ t_7 &= x_{(q-2^j-2^k)} \wedge \overline{s_i} \wedge s_j \wedge s_k \\ t_8 &= x_{(q-2^i-2^j-2^k)} \wedge s_i \wedge s_j \wedge s_k \end{aligned}$$

A general-purpose technology mapper should be able to identify the most efficient implementation of the traditional *unmerged* stage, *dual-merged* stage and *triple merged* stage of a shifter. The best possible delays of these three types of stages are denoted as Del_1, Del_2 and Del_3.

In addition to the design of the merged stages, the technique for identification of the *mergeable* stages plays a key role to determine the performance of the shifter architecture. Without an efficient algorithm to identify the *mergeable stages*, the design of merged stages would not be useful.

In the approach of [9], the following timing-driven analysis was done to find two or three stages for merging: Assume that the earliest arriving three shift signals are s_i, s_j and s_k. Let ts_i be the arrival time of the shift signal s_i. For the signals s_i and s_j, if a *dual merged* stage is constructed, then the output of the dual merged stage will be available at time

$$T_{dual} = ts_j + Del_2.$$

On the other hand, if two individual stages are constructed in cascade, then the output of the second stage will be available at time

$$T_{single2} = \text{Max}\ ((ts_i + Del_1),\ ts_j) + Del_1.$$

Similarly, for the signals s_i, s_j and s_k, if a *triple merged* stage is constructed, then the output of the triple merged stage will be available at time

$$T_{triple} = ts_k + Del_3.$$

On the other hand, if three individual cascaded stages are constructed, then the output of the third stage will be available at time

$$T_{single3} = \text{Max}\ (T_{single2},\ ts_k) + Del_1.$$

Now, if (T_{triple}<$T_{single3}$) and (T_{triple}<(T_{dual}+(Del_2/2)), then the three stages (i, j, and k) of the shifter are chosen as the *mergeable* stages. If the above conditions are not true and if ($T_{dual} < T_{single2}$), then the two stages (i and j) are selected as the *mergeable* stages. If both the above conditions are false (which means, ($T_{single2} \leq T_{dual}$) and ($T_{single3} \leq T_{triple}$)), then i is not included into any merging combination and one single stage is utilized for the stage i. Next, the same analysis is performed with the three stages corresponding to the next three earliest arriving bits. This analysis and identification of *mergeable* stages continues until all the stages are analyzed. At the end of this algorithm, the list of all the *mergeable stages* is determined in this manner.

In terms of the execution of the flow, the mergeable stages are first identified. Once the configurations of all the *dual merged*, *triple merged* and *unmerged* stages are identified, then the merged stages as well as unmerged single-stages in the netlist are implemented with proper connectivity.

Note that during technology mapping in our approach, the mapper sizes the output of any node based on their load capacitance. Also, the delay analysis for each configuration considers actual capacitance of the output node, using a load-dependent delay model. Also, note that any of our nodes inside the shifter block do not have high fanouts.

C. Reduction of Partial Products

In our approach, all the shifted partial products (corresponding to the q sum-of-products), the non-shifted partial products (corresponding to the remaining p-q sum-of-products) and the r additive terms are fed to a partial product reduction tree. The purpose of this partial product reduction tree is to perform column-wise reduction of the elements in each bitslice, such that each bitslice finally consists of 2 elements or less.

If all the sum-of-products in the Figure-1 are multipliers and each signal is n-bits wide, then a total of (pn+r) partial products will be fed to this partial product reduction tree. On the other hand, if all the sum-of-products in the Figure-1 are multiply-accumulators (MAC) and each signal is n-bits wide, then a total of (pn+p+r) partial products will be fed to this partial product reduction tree.

To perform the reduction of partial products, we use the technique presented in [7]. This technique uses the concept of *counters*. A *(p:q) counter* is a functional block, which adds p single-bit inputs and produces q single-bit outputs; where p and q satisfy the following equation: $q = \lfloor log_2 p + 1 \rfloor$

To reduce the partial products, we use the concept of (4:3) counter. This is defined as a functional block which accepts 4 single-bit signals in the i^{th} bitslice and transform them into 3 different single-bit output signals (one for the i^{th} bitslice; one for the $(i+1)^{th}$ bitslice and the third one for the $(i+2)^{th}$ bitslice). Let us assume that there is a (4:3) counter in bitslice$_i$, which takes 4 signals (a_i, b_i, c_i and d_i) as inputs and produces 3 outputs (x_{i+2} for the bitslice$_{i+2}$, x_{i+1} for the bitslice$_{i+1}$ and x_i for the bitslice$_i$). The functionality of the (4:3) counter is as follows:

- $x_i = (a_i \oplus b_i) \odot (c_i \odot d_i)$
- $x_{i+1} = \overline{((a_i \wedge b_i) \odot (c_i \wedge d_i)) \wedge \overline{((a_i \oplus b_i) \wedge (c_i \oplus d_i))}}$
- $x_{i+2} = a_i \wedge b_i \wedge c_i \wedge d_i$

The (3:2) counter is widely used in column-compression schemes. A (3:2) counter accepts 3 inputs signals (a_i, b_i and c_i) belonging to the i^{th} column (bitslice) in the partial-products and would produce 1 output signal (x_i) for the i^{th} column (bitslice) and 1 output signal (x_{i+1}) for the $(i+1)^{th}$ bitslice. The functionality of the (3:2) counter is:

- $x_i = a_i \oplus b_i \oplus c_i$
- $x_{i+1} = \overline{\overline{(a_i \wedge \ b_i)} \wedge \overline{(b_i \wedge \ c_i)} \wedge \overline{(c_i \wedge \ a_i)}}$

Similarly, the functionality of a (2:2) counter is:

- $x_i = a_i \oplus b_i$
- $x_{i+1} = (a_i \wedge \ b_i)$

In our approach, we use a timing-driven algorithm to design the partial product reduction tree by using a combination of (4:3), (3:2) and (2:2) counters. The key idea in our partial product reduction approach is to find the opportunity to use the (4:3) counters. Let us consider that a_i, b_i, c_i and d_i are four input signals (sorted in the ascending order of the arrival time) in bitslice$_i$. Our algorithm would use the following scheme to determine the type of counter to be instantiated:

- If a_i and b_i arrive at least a 2-input EXOR gate-delay before signal c_i arrives; then instantiate a (2:2) counter.
- If the above condition fails and a_i, b_i, c_i arrive at least two 2-input EXOR gate-delay before the signal d_i arrives; then instantiate a (3:2) counter.
- If both the above-mentioned conditions fail; then instantiate a (4:3) counter.

In other words, our algorithm instantiates a (4:3) counter, if the arrival times of all four signals at the bitslice$_i$ are *reasonably close* to each other. We continue to perform the reduction in all the bitslices until each of the bitslices contain ≤ 2 elements. With an instantiation of the (4:3) counter, four elements are reduced in every bit. In addition, due to the simple circuitry needed to generate x_{i+2} in a (4:3) counter; the arrival time of the signal (x_{i+2}) at the input of bitslice$_{i+2}$ is also low. This reduces timing-skew of the signals at the output of the reduction tree.

D. Computation of the Final Sum

After performing the column-wise reduction of the partial products, each column in the reduced element-set consists of a maximum of 2 elements. Hence, after the partial product reduction step, we effectively transform all the partial products into two operands. To produce the final output (z) of our targeted datapath design, the 2 addends of all the columns have to be added by a final carry propagate adder circuit. Therefore, a 2-operand addition is required to compute the final result of the arithmetic block.

To obtain faster performance, we need to use a fast addition technique. In high-frequency datapath designs, adders with parallel prefix computation methodologies [14] are very popular. The hybrid adder described in [16] exploits the skewed pattern of the input arrival-times and can be very effective for the sum-of-product computation. In our approach, we use a hybrid adder which consists of three subadders. The bit-width of each of the subadders in the hybrid adder are computed by using the approach discussed in [17]. The internal topology of our hybrid adder is as follows:

- *Ripple-Carry* for the few bits near the least significant bit (LSB).
- A fast *Kogge-Stone* adder for *several* bits in the middle.
- A carry-select adder based on the *Kogge-Stone* architecture for the *remaining* bits near the most significant bit (MSB).

V. Experimental Setup

We have implemented our proposed approach in the C++ programming language. The experiments were performed with datapath RTL designs written in Verilog hardware description language. For all our experiments, we used a Linux workstation (RedHat 7.1) with dual-2.2GHz processors and 4GB memory.

To collect different data-points regarding the quality of results for the different types of datapath designs involving SOP and shifter blocks in the timing-critical portion of the design, we used the following variations:

- Multiple types of designs of different expressions and input bit-widths:

  - Our first design has one SOP block (multiply-accumulator) driving a shifter. That means, q=1. The three Inputs to the SOP block are 16-bit wide. In addition, there is another SOP block (multiplier), whose output directly drives the adder. That means, p=2. The two inputs to the SOP block are 16-bit wide. Finally, there are two 16-bits wide additive signals which get fed to the final adder. That means, r=2. Figure-3 shows a block-diagram of this example. We referred to this design as Des-q1-p2-r2.
  - Using this notation, the next design is referred to as Des-q3-p7-r6.
  - We also experimented with the following 3 designs: Des-q2-p3-r0, Des-q4-p9-r5 and Des-q1-p6-r3.

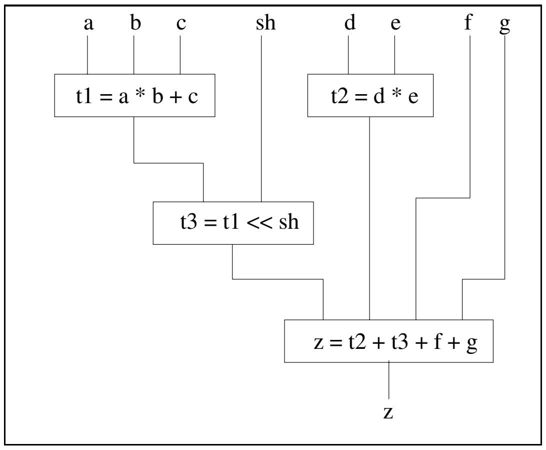

Fig. 3. The block-diagram of one design (Des-q1-p2-r2)

- The different technologies and libraries we used are:
  - Two industrial libraries (L_1 and L_2) for a 0.13μ technology.
  - Two industrial libraries (L_3 and L_4) for a 0.09μ technology.
- Different input arrival time constraints:
  To facilitate the explanation, let us assume that there are four inputs to the datapath structure of the Figure-1 and each of the four input signals is n-bit wide. We have used the following types of input arrival time constraints:
  - All input bits of all the signals arrive at the same time. We refer to this constraint as Type-A. If we denote $Arr(a_i)$ as the arrival time of the bit a_i and if k is a constant number, then this Type-A constraint can be represented as:

$$\begin{array}{ll} Arr(a_i) = k; & 0 \leq i < n \\ Arr(b_i) = k; & 0 \leq i < n \\ Arr(c_i) = k; & 0 \leq i < n \\ Arr(d_i) = k; & 0 \leq i < n \end{array}$$

    This category represents the actual timing situations if the SOP blocks are placed immediately after a register-bank or the primary inputs of the design are fed to the SOP blocks.
  - Different input bits arrive at different times. We refer to this category of timing constraints as Type-B. We believe that this category represents the actual timing situations for many sum-of-product blocks in real-life designs. Assuming that k is a constant number and δ is the delay of the fastest 2-input AND-gate in the given technology library, the following are some specific examples of the Type-B timing constraints. Here we have explained the arrival times for signal a_i. Similar expressions for arrival times of all the bits of signals b, c and d can be written as well.

1) $Arr(a_i) = i * k * \delta; \quad 0 \leq i < n$
2) $Arr(a_i) = i^2 k\delta; \quad 0 \leq i < n$
3) $$\begin{array}{ll} Arr(a_i) = 0; & 0 \leq i < \lceil n/2 \rceil \\ Arr(a_i) = k\delta; & \lceil n/2 \rceil \leq i < n \end{array}$$
4) $$\begin{array}{ll} Arr(a_i) = 0; & 0 \leq i < \lceil n/4 \rceil \\ Arr(a_i) = k\delta; & \lceil n/4 \rceil \leq i < \lceil n/2 \rceil \\ Arr(a_i) = 2k\delta; & \lceil n/2 \rceil \leq i < \lceil 3n/4 \rceil \\ Arr(a_i) = 3k\delta; & \lceil 3n/4 \rceil \leq i < n \end{array}$$
5) $$\begin{array}{ll} Arr(a_i) = 0; & 0 \leq i < \lceil n/4 \rceil \\ Arr(a_i) = ik\delta; & \lceil n/4 \rceil \leq i < \lceil n/2 \rceil \\ Arr(a_i) = 2ik\delta; & \lceil n/2 \rceil \leq i < \lceil 3n/4 \rceil \\ Arr(a_i) = 3ik\delta; & \lceil 3n/4 \rceil \leq i < n \end{array}$$

VI. Experimental Results

We compared our approach against a commercially available datapath synthesis tool which is considered to be the best-in-class solution. The synthesis tool generates arithmetic-optimized architectures for all the arithmetic blocks (like sum-of-products, shifters, adders) and then it performs general-purpose operations like technology-independent optimizations, constant propagation, redundancy removal, technology mapping, timing-driven optimization, area-driven optimization, incremental optimization etc. While running the synthesis tool, we turned on all the above-mentioned optimizations. In the Table-I, we report the worst-case delay and the total area results obtained for the datapath block from the commercial synthesis tool and from our approach. In this table, we report 20 sets of data-points involving different combinations of datapath blocks and technology libraries.

If we compute the average of all the 20 data-points presented in the Table-I, then our approach results in about 13.28% faster implementation of the datapath block, with a 3.24% area penalty compared to the netlist generated by the commercial datapath synthesis tool. State-of-the-art designs have very strict timing goals, hence most designers would be willing to accept a 13.28% delay improvement at the expense of a 3.24% area penalty of the datapath block only.

To keep the size of the Table-I relatively brief, we do not report the results for different types of Type-B timing constraints. Note that the results in each of the combinations which are not reported here also support our conclusion that the proposed approach produces significantly faster netlist.

			Worst-case Delay (ps)			Area (μ^2)		
Design Name	Technology Library	Timing Constraint	Commercial Tool	Our Approach	(%) Improvement	Commercial Tool	Our Approach	(%) Penalty
Des-q1-p2-r2	Lib_1	Type-A	1572	1396	11.19%	6358	6469	1.76%
Des-q3-p7-r6	Lib_1	Type-A	1749	1523	12.92%	22413	23316	4.03%
Des-q2-p3-r0	Lib_1	Type-A	1527	1351	11.53%	9862	10125	2.67%
Des-q4-p9-r5	Lib_1	Type-A	1681	1447	13.87%	29571	31318	5.91%
Des-q1-p6-r3	Lib_1	Type-A	1603	1419	11.47%	20247	20814	2.80%
Des-q1-p2-r2	Lib_2	Type-A	1243	1068	14.08%	7416	7523	1.45%
Des-q3-p7-r6	Lib_2	Type-A	1419	1254	11.63%	25539	26514	3.82%
Des-q2-p3-r0	Lib_2	Type-A	1207	1045	13.42%	11282	11649	3.26%
Des-q4-p9-r5	Lib_2	Type-A	1356	1172	13.56%	34753	36480	4.97%
Des-q1-p6-r3	Lib_2	Type-A	1285	1093	14.94%	23927	24346	1.75%
Des-q1-p2-r2	Lib_3	Type-A	1847	1582	14.35%	4672	4779	2.31%
Des-q3-p7-r6	Lib_3	Type-A	2169	1814	16.37%	16461	17031	3.46%
Des-q2-p3-r0	Lib_3	Type-A	1785	1539	13.78%	7518	7786	3.58%
Des-q4-p9-r5	Lib_3	Type-A	2031	1763	13.19%	22085	23257	5.32%
Des-q1-p6-r3	Lib_3	Type-A	1914	1627	14.99%	14359	14652	2.04%
Des-q1-p2-r2	Lib_4	Type-A	1094	961	12.16%	6892	7019	1.85%
Des-q3-p7-r6	Lib_4	Type-A	1256	1089	13.29%	23641	24521	3.71%
Des-q2-p3-r0	Lib_4	Type-A	1061	913	13.95%	10367	10694	3.16%
Des-q4-p9-r5	Lib_4	Type-A	1183	1027	13.19%	32753	34286	4.69%
Des-q1-p6-r3	Lib_4	Type-A	1128	996	11.70%	21536	21863	1.52%
Average					13.28%			3.24%

TABLE I

AREA AND DELAY COMPARISON OF BLOCKS GENERATED BY A COMMERCIAL SYNTHESIS TOOL AND BY OUR APPROACH

To verify the correlation of the post-synthesis experimental data of the Table-I with the post place-and-route data, we performed placement and routing on Des-q1-p2-r2 and Des-q3-p7-r6. For these two testcases, the average improvement in the post-routing worst case delay of the datapath design generated by our proposed approach is 12% compared with the worst delay of the corresponding block generated by the commercial datapath synthesis tool (with the average 4% post-roting area penalty). The individual results for these testcases correlate closely with the post-synthesis numbers reported in the Table-I. These post-routing data confirm our conclusion about significant timing improvement of the netlist produced by using our approach (with a modest area penalty).

Our delay improvement is consistent across multiple types of designs, technology libraries and arrival time constraints. This underscores the strength of our approach. Since this type of datapath structure is frequently used in modern digital design, we believe that the timing-critical portions of many real-life designs can significantly benefit from our approach.

VII. CONCLUSION

In this paper, we have presented a new approach to implement a faster datapath block involving arithmetic sum-of-products, shifters and a final adder. Our approach would be very useful when the critical path of the design goes through such a block. Our approach to generate the timing-efficient architecture for this block works seamlessly with different types of datapath blocks, arrival timing constraints and across different technology domains (0.13μ, 0.09μ). The experimental results indicate that our implementation of the datapath block is significantly faster (with a modest area penalty) than the datapath block generated by a commercially available best-in-class datapath synthesis tool.

REFERENCES

[1] T. Kim, W. Jao, S. Jjiang. "Circuit optimization using carry-save-adder cells," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-17*, pp. 974–984, 1998.

[2] A. Mathur, S. Saluja. "Improved merging of datapath operators using information content and required precision analysis," in *Proceedings of the* 38^{th} *conference on Design Automation*, pp. 462-467, 2001.

[3] A. K. Verma, P. Ienne. "Improved Use of the Carry-Save Representation for the Synthesis of Complex Arithmetic Circuits," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pp. 791-798, 2004.

[4] A. Fayed, W. Elgharbawy, M. Bayoumi, "A data merging technique for high-speed low-power multiply accumulate units," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 145–148, 2004.

[5] L. Chen, O. T. C. Wang, Y. C. Ma. "A multiplication accumulation computation unit with optimized compressors and minimized switching activities," in *IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 6118–6121, 2005.

[6] C. S. Wallace, "A suggestion for a fast multiplier," in *IEEE Transactions on Electronic Computers*, EC-13(2):14-17, 1964.

[7] S. Das, S. P. Khatri, "A Timing-Driven Hybrid-Compression Algorithm for Faster Sum-of-Products", in *Proceedings of International Conference on Circuits, Signals and Systems*, 2007.

[8] R. S. Lim, "A Barrel Switch Design," in *Computer Design*, pp. 76-78, 1972.

[9] S. Das, S. P. Khatri, "Timing-Driven Decomposition of a Fast Barrel Shifter", in *Proceedings of IEEE MidWest Symposium on Circuits and Systems*, 2007.

[10] R. Rafati, S. M. Fakhraie, K. C. Smith, "A 16-Bit Barrel-Shifter Implemented in Data-Driven Dynamic Logic (D^3L)," in *IEEE Transactions on Circuits and Systems I*, vol 53, issue 10, pp. 2194-2202. 2006.

[11] P. M. Seidel, K. Fazel, "Two dimensional folding strategies for improved layouts of cyclic shifters," in *Proceedings of the IEEE Computer society Annual Symposium on VLSI*, pp. 277-278, 2004.

[12] M. A. Hillebrand, T. Schurger, P. M. Seidel, "How to half wire lengths in the layout of cyclic shifters," in *Proceedings of the IEEE International Conference on VLSI Design*, pp. 339-344, 2001.

[13] A. P. Singh, M. Barany, D. J. Deleganes, "A mixed signal rotator/shifter for 8GHz Intel/spl reg/ Pentium/spl reg/ 4 integer core," in *Proceedings of the Symposium on VLSI Circuits*, pp. 394-397, 2004.

[14] P. M. Kogge, H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," in *IEEE Transactions on Computers*, C-22(8):783-91, 1973.

[15] R. P. Brent, H. T. Kung, "A regular layout for parallel adders," in *IEEE Transactions on Computers*, C-31(3):260-64, 1982.

[16] P. F. Stelling, V. G. Oklobdzija, "Design strategies for the final adder in a parallel multiplier," in 29^{th} *Asilomar Conference on Signals, Systems and Computers*, pp. 591-595, vol. 1, 1995

[17] S. Das, S. P. Khatri, "Generation of the Optimal Bit-Width Topology of the Fast Hybrid Adder in a Parallel Multiplier", in *Proceedings of International Conference on Integrated Circuit Design and Technology*, 2007.

[18] M. D. Ercegovac, T. Lang, "Digital Arithmetic," *The Morgan Kaufmann Series in Computer Architecture and Design*, 2003