

Pruning the Design Space for Wide-bit Multipliers: An Estimation Approach for Programmable Logic-based Embedded Systems¹

Gang Quan, James P. Davis and Siddaveerasharan Devarkal

Department of Computer Science and Engineering

University of South Carolina

Columbia, SC 29208 USA

{jimdavis, gquan, devarkal}@cse.sc.edu

Abstract

In this paper, we explore some of the architectural issues associated with architecting high-performance wide-bit multiplier units on programmable logic devices for embedded systems applications. Many applications of reconfigurable computing for arithmetic, such as for DSP and Cryptography, as reported in the literature, have been carried out with relatively small bit-widths, thereby avoiding some of the more difficult design tradeoffs and scheduling control issues associated with larger operand widths. In this paper, we define an Estimator function and present some of the results obtained in exploring the architectural space associated with large-bit width multipliers for cryptography applications. Specifically, we explore topologies for hierarchical, hybrid multiplier architectures with 192, 224, 256, 384 and 521-bit operands, to provide finer grain performance/resource usage tradeoffs during design space exploration. The timing and area cost for the general hybrid multiplier is analytically formulated by exploring the regularity of its internal structure, and then adjusted empirically. Experimental results show that our approach can rapidly predict the timing and area cost for the hierarchical hybrid multiplier with a reasonable accuracy. We discuss the relevance of this method in the context of MAC Layer embedded design of encryption enhancements for 802.11g and 802.11i wireless LAN standards.

1. Introduction

Reconfigurable computing machines (RCM) have become recognized as a viable means for achieving orders of magnitude speedup in compute-intensive applications, through the use of fine-grained parallelism on a customized logic substrate [1, 2, 3]. Algorithms in a wide variety of military and scientific domains--along

¹ The work has been partially supported by the Department of Defense under the Lucite contract #MDA904-98-C-A081.

the architectures necessary for their implementation--are being implemented on a custom-logic "fabric" consisting of one or more commercially-available field programmable gate arrays (FPGAs) [4]. This fabric is generally suited to fine-grained parallelism [1]. Highly repetitive operations, such as pipelined arithmetic calculations on wide data words, and highly parallel SIMD operations--such as those found in image processing and cryptography—have been a good match for RCM-based high-performance computing applications [1, 3].

However, the use of reconfigurable computing in embedded systems design is still fairly new endeavor. The interest in using programmable logic and its inherent logic reconfiguration capabilities is based on a desire to achieve a greater degree of dynamic programmability—typically afforded to embedded systems built around a microprocessor—while also providing a platform for orders-of-magnitude faster computing at lower power consumption levels than is available through conventional instruction-set microprocessor-based embedded systems [3]. An example problem domain where such processing may be required is in the WLAN for ubiquitous computing arena. Here, with the 802.11 protocols for example, we are moving towards processing requirements, coupled with power and device “footprint” requirements, where delivering a solution using conventional embedded systems techniques will be a great challenge. In response to this, there are two primary means to achieve the type of embedded systems performance that may be required of a new generation of embedded computing devices: (1) systems-on-chip (SoC); and, (2) processors coupled with reconfigurable, programmable logic devices. In both cases, we need to be able to consider the architecture and design of high-performance components that can be moved into VLSI logic—either the custom logic of an SoC device, or the programmable logic of an FPGA, both of which might be tightly coupled to a microprocessor or some sort.

In the context of creating this next generation of embedded system, the problem discussed in this paper involves creating high-performance architectures for arithmetic processing units, namely Integer multipliers for use in encryption/decryption according to the new AES standard [5], where the operand width is greater than that typically operated on by embedded microprocessors. It can be easily imagined that the cycle cost of attempting to implement a software algorithm for processing the 128-, 192- and 256-bit operands used in the

AES algorithm using a conventional instruction set will be very costly. This motivates the search for solutions where custom-logic or programmable-logic solutions are to be used for the custom arithmetic processing architecture, where large bit-width b , $b \geq 128$, multiplications and additions dominate.

The process of converging on an optimal multiplier architecture and implementation, in the absence of any specific heuristics, requires iterating through the architecture and design process many times before finally settling on an architecture for the application while using the target device's resources most effectively.

In this paper, we present a method—and results of using this method by creating an automated procedure—for estimating the timing and resource requirements for a range of hybrid, hierarchical, unsigned Integer multipliers. Our objective has been to devise architectures that can be efficiently implemented on programmable logic devices, such that we can realize high-performance, large-bit width arithmetic processing for embedded cryptographic systems applications. In this paper, we examine one aspect of this problem, namely, the analysis of architectures supporting multiplication of operands 192, 224, 256, 384 and 521 bits in width. These bit-widths are recommended according to the NIST standards [6] and have been discussed relative to their use in reconfigurable computing in such sources as [7, 8].

We examine this activity of estimating the performance characteristics of a given hierarchical, hybrid multiplier architecture scheme as an activity that precedes actual design of such a unit. It is the premise of this paper that considerable savings in the development cost of such a model can be achieved through the use of this estimation procedure. Furthermore, the procedure creates estimates that have a very low percentage error over actual measures generated through the use of synthesis and layout tools, indicating the clear benefit of using this approach.

The paper is organized as follows. In Section 2, we discuss the large-bit width multiplication problem, and consider what has been reported in the literature to date. We also discuss the impetus of devising an Estimator tool, namely the combinatorial explosion of possible design candidates. In Section 3, we present, in detail, the primary architecture “patterns” we adopted for exploring the most plausible architecture solutions for this class of Integer multiplication application. In Section 4, we present the theoretical analysis for the Estimation

capability itself, whereupon we subject the resultant model to comparative validation against actual implementation data. Finally, we sum up the ramifications of our estimation method and tools in the context of the aforementioned embedded systems problem.

2. The Problem of Large Multipliers

We seek to articulate a set of specialized heuristics through an “estimator” method that can guide us in the selection and use of different multiplier architectures in varying applications of crypto-arithmetic processing. First, we need to understand why it is infeasible to simply multiply two large operands together using monolithic units. Second, we need to understand why the number of possible architectures to solve this problem is very large.

2.1 Monolithic versus Hierarchical Multiplication

We need to select a basic set of candidate structures for constructing large-width multiplier data paths. On first consideration, it might be viewed as acceptable to simply define a “monolithic” (i.e., flat) multiplier that simply takes two large operand word lengths, and multiplies the numbers to generate partial products, then add up the partials as one would do using the “pencil and paper” method. However, it is impractical to construct such monolithic circuits of very large bit-widths [7, 9, 10], for the obvious reasons that: (1) a monolithic circuit of such bit-widths is likely to be non-optimal, and difficult to optimize, for specific applications; and, (2) a monolithic circuit is likely to be difficult to synthesize into a circuit.

Therefore, we must model our different multiplier schemes as smaller, more basic units, from which we construct larger units. These units are organized hierarchically, in that a large bit-width unit contains some number of units that execute smaller multiply and add operations (either concurrently or in a pipelined fashion) on a sub-range of the larger operands’ bits. The results of the smaller multiplication operations are recombined according to the topology of the given architecture. As discussed in [9], a hierarchical design approach allows

better management of design complexity by exploiting the regularity of large numbers of identical structures, and also affords a better chance that a usable and efficient circuit can be synthesized.

Having modeled basic multiplier topologies, we subject these units to analysis in order to devise the analytics of the Estimator. As pointed out in Chen and Mead [9], what might be considered an efficient or optimal algorithm in a mathematical or software sense may not be so from the standpoint of its VLSI architecture because of physical cost considerations. Therefore, we must subject each model to cost evaluation in order to identify and select an optimal design for our intended application. For synthesized circuits, this is generally done in terms of area, speed and power characteristics. We will, for the moment, defer considering the power budget issue and focus on the issues of area and speed. Area is important, as we need to be able to assess the ability of a multiplier unit to scale to size, while also continuing to “fit” on the FPGA device or custom logic substrate. Otherwise, we incur performance penalties by having to go “off chip” to complete the arithmetic function. Speed is important for obvious reasons, in that minimizing latency (in the form of worst-case delay and maximum clock rate through the unit) affects the overall computation throughput.

2.2 Circumscribing a Large Design Search Space

We are using unsigned Integer arithmetic of the following data path widths: 128, 192, 224, 256, 384 and 521 bits in width. (Note that the 521-bit computation is rounded up to 544 bits, as this allows use of a standard unit configuration for this purpose.) We assume at present that the architectures for each of the bit widths may dictate a different—or even more than one—multiplier architecture for use in a given reconfigurable component of an embedded application. Therefore, to appropriately account for this, and select the best candidate, we must analyze larger multiplier architectures using different basic multiplier unit configurations, and evaluate the characteristics of each configuration.

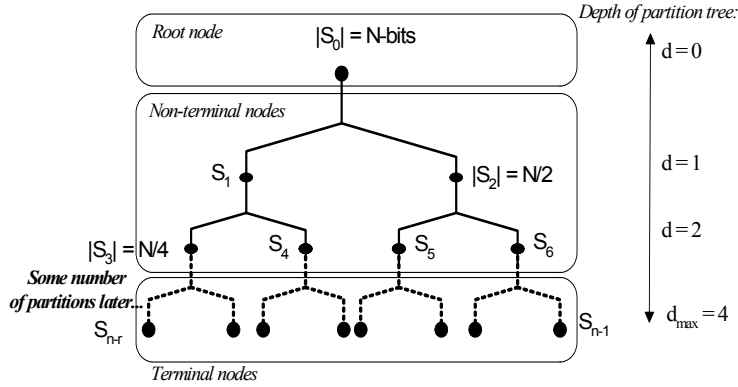


Fig..1. Characterizing the Partition Space.

One of the problems is that, when considering the most efficient scheme for large bit-width multiplication, we quickly realize that the set of candidate topologies is very large. We can easily see the size of this design space by considering the hierarchical partitioning of a large bit-width multiplier into a compositional tree of some depth. Such a tree might be depicted as shown in Fig 1.

- Let ∂ be the *depth* of a partition tree, as shown in Fig. 1, where $1 \leq \partial \leq 4$ for our maximum MUL width of 544 bits.
- Let $G = R \cup NT \cup T$ the union of tree node possibilities, where $|G| = n = 29$.
- Let C be a set representing the ordered sequence of non-repeating configuration units, where

$$\forall c, (c \in C) \subseteq (c \in G)$$

$$|C| = r = 2, 3, 4, 5, \text{ where } r = \partial + 1.$$

- Let P be the permutation of the set G of MUL architectures by type and width, taken r elements at a time, forming a candidate sequence set C , such that the following is true.

$$\sum_{\mathbf{r}} P(n,r) = n! / (n - r)!$$

From the circumscribing of the search space in terms of its permutation equation, we see that this is a really large number of possible hierarchically defined wide-bit MUL architectures. It also illustrates why we simply cannot devote the time—in a development project—on exploring many of these possible candidates, as the time to generate post layout timing and area values for each would be too prohibitive. In addition, given

the wide range of possible results of these multiplier units, it is not feasible to simply pick one at random without some *a priori* understanding of its speed/area tradeoff. Note that the compositional tree structure shown in Fig. 1 is valid for only a given hierarchical topology; different factoring of the multiplication into smaller units will result in different trees (not necessarily binary ones).

3. Architectural Treatment

Multiplier architectures can be grouped into two different categories: *non-hierarchical* or *hierarchical*. A non-hierarchical multiplier constructs the product directly from the inputs monolithically, while a hierarchical multiplier builds upon sub-multipliers and derives the product from the partial products generated from the sub-multipliers. The multiplicands of hierarchical multipliers are divided into different blocks. The partial products computed from these sub-blocks are integrated to form the complete product. Good scalability, ease of parallelism exploitation, high regularity structure, complexity management, etc, are among the advantages of the hierarchical multipliers. In this paper, we focus on two basic architecture “patterns” for large bit-width multiplication. We then take both of these and combine them into a number of possible hybrid configurations.

3.1 Divide and Conquer multiplier

The first multiplier architecture pattern that is part of our model is the Divide and Conquer scheme, discussed in references such as [10]. There is a “naïve” version [10], and also an optimized version [11]. We discuss both versions.

According to the naïve Divide-and-Conquer (DC) approach, assume that A and B are n-bit numbers and can be evenly divided into two halves, i.e., A_H and A_L , B_H and B_L , respectively. Let

$$a_0 = A_H \times B_H, a_1 = A_H \times B_L, a_2 = A_L \times B_H, a_3 = A_L \times B_L,$$

Then

$$A \times B = 2^n a_0 + 2^{n/2} (a_1 + a_2) + a_3.$$

Note that, the four partial products, i.e., a_0, a_1, a_2, a_3 , can be computed in parallel. In order to perform this calculation, four $(n/2\text{-bit})$ multipliers are required.

To reduce the resource requirements, the Karatsuba-Ofman Algorithm (KOA) [11] reduces the number of sub- multipliers by replacing the multiplication operations with several additions. That is, let

$$a_0 = A_H \times B_H, a_1 = (A_H + A_L) \times (B_H + B_L), a_2 = A_L \times B_L.$$

Then, the product can be computed as

$$A \times B = 2^n a_0 + 2^{n/2} (a_2 - a_1 - a_0) + a_2.$$

In this case, only two $n/2\text{-bit}$ and one $(n/2+1)\text{-bit}$ multiplication operations are necessary. Moreover, in order to use the same multiplier for all the multiplications, the KOA algorithm can be implemented in a slightly different way as shown below. Let

$$a_0 = A_H \times B_H, a_1 = (A_H - A_L) \times (B_H - B_L), a_2 = A_L \times B_L. \quad (1)$$

Then,

$$A \times B = 2^n a_0 + 2^{n/2} (a_2 + a_0 - a_1) + a_2. \quad (2)$$

Fig. 2 shows the implementation of a 192-bit multiplier with KOA scheme.

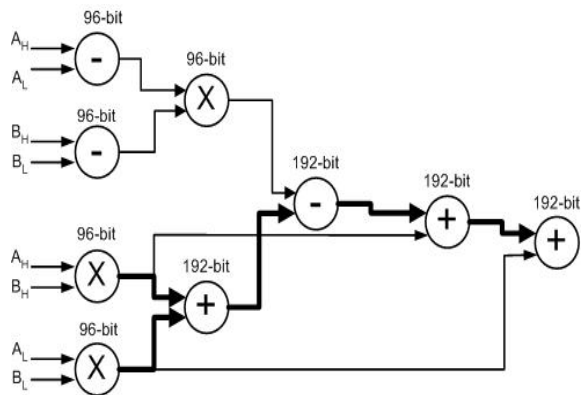


Fig. 2. KOA-style 192-bit multiplier (the thick lines represent the critical path).

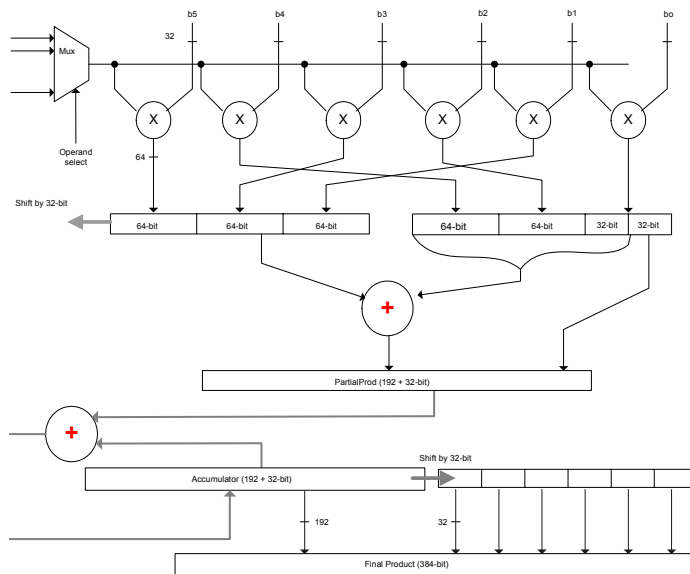


Fig. 3. Broadcast-style 192-bit multiplier

Even though the basic 2-way DC method can, theoretically, be easily extended for m-way partitioning, the practical control overhead (including interconnect and state machine) will likely outweigh the potential performance benefits [6].

The naïve DC strategy and KOA algorithm achieve high computation performance by exploiting parallelism in computing the partial products. However, such a high performance is not achieved without extra cost in terms of circuit area of the required components.

Note that, the number of base multipliers required by the naïve DC strategy and KOA algorithm increases exponentially with the number of hierarchical levels. For example, to implement a 256 by 256 multiplier, as many as 256 16x16 base multipliers are needed if the naïve DC strategy is applied, or 81 are needed for the KOA algorithm. The excessive resource requirement can easily consume the hardware resources on an FPGA (such as the hardware multipliers available on certain devices), which makes it impossible to synthesize the high bit-width multiplier on a single FPGA.

3.2 Broadcast multiplier

An effective and flexible design strategy for trading-off performance and resource is to apply the Broadcast (BC) multiplier pattern for large operand widths, proposed by Buell, *et al* [7], as follows:

Let A and B be n -bit integers. Assume that the available resources can be used to realize k multipliers, each of which can compute the p -bit multiplication, where

$$p = \left\lceil \frac{n}{k} \right\rceil.$$

Then, we can partition both A and B into k blocks such that

$$A = \{A_{(k-1)} \wedge A_1 A_0\}, B = \{B_{(k-1)} \wedge B_1 B_0\},$$

and $|A_i| = |B_i| = p$. Then at the first cycle, we can use the k multipliers to compute the partial products $A_i B_0, i = 0, \dots, (k-1)$, simultaneously. After these products are appropriately accumulated, we can then use all the multipliers to compute another round of partial products $A_i B_1, i = 0, \dots, (k-1)$ in parallel, and accumulated to the previous results. In general, we have

$$A \times B = \sum_{i=0}^{k-1} ((A_{(k-1)} \wedge A_1 A_0) \times B_i \gg (i \times p)).$$

A BC multiplier is, in fact, a sequential block-based shift-and-add multiplier. It contains k iterations of partial product generation and accumulation steps. The computation efficiency of the Broadcast multiplier comes from the fact that multiplication cost for generating the partial products usually dominates the accumulation cost. Since a Broadcast multiplier has k independent sub-multipliers, the computations of the k sub-products can be computed in parallel during each sub-operand pass.

In Fig. 3, we depict the second of the two candidate wide-bit architecture patterns, namely, the Broadcast (BC) scheme [7]. In this Broadcast scheme, we break up a 192-bit radix into six separate 32-bit multiplier units, whose outputs are fed into 64-bit pipeline registers, according to a “perfect shuffle” permutation scheme [12]. The partial product registers are grouped into 3-units, where each group is treated as a unit for purposes of shifting partial product data. The leftmost 3-units form a 192-bit shift register, on which a 32-bit SHL operation occurs after each multiplication cycle.

The data stored in these registers is then clocked into three 64-bit adder units, organized into an “exchange permutation” scheme [15] whose outputs are fed into a second stage of pipelined registers. The following stage adds the results of these three registers, grouped as a 3-unit, with an “accumulator” register 3-unit consisting of three additional 64-bit pipelined registers. During this final addition, the contents of the permuted partial product addition registers are added to the running value stored in the accumulator. Between each accumulator add, the contents of the 3-unit accumulator are right-shifted 32-bits into an additional 3-unit register group. After six such accumulator-adds and 32-bit shifts, a complete final product is contained within the running 384-bits of 64-bit registers organized into two 3-unit registers. A final product register, consisting of six 64-bit registers, stores the product for latching by another stage of the application.

3.3 Hardware Multiplier Resources

The Xilinx Vertex-II® FPGA comes with embedded multiplier blocks that can do signed multiplication for inputs of up to 18-bits wide, and unsigned up to 17-bits wide. These multiplier blocks can either be registered or non-registered. The non-registered version is referred to as MULT18x18 and the registered block as MULT18x18s, but both instantiate the same block. When the result is not registered, MULT18x18 is instantiated and when the result is registered MULT18x18s is instantiated. The multiplier block also has an internal register that can be used if one needs to run the multiplier at higher speeds. To instantiate the internal register the product has to be registered twice—the first register is realized as an internal register and the second as the output register.

3.4 The Hybrid Multiplier Scheme

Compared with the Divide-and-Conquer scheme, the Broadcast multiplier has an advantage, in that it is more flexible in terms of determination of the bit-width and number of sub-multipliers to be implemented in the hardware, thereby achieving better resource usage. However, since it can only make use of partial

parallelism in the multiplication, it cannot achieve the computation performance of the Divide-and-Conquer schemes.

As explained before, the KOA and Broadcast schemes represent two end choices for the hierarchical multiplier design. The KOA approach can better exploit the parallelism in the partial product generation but requires greater resource usage. The Broadcast, on the other hand, is more resource conservative, but with lower computation performance. As resource and delay tradeoffs are critical for the high level synthesis of new designs, it is desirable that we take the advantages afforded by each scheme by making proper tradeoffs in the design. To achieve this trade-off, we incorporate these two models into a single design pattern to achieve this goal in the design of high bit-width multipliers.

A *hybrid multiplier* is the hierarchical multiplier that adopts different hierarchical implementation strategies at different hierarchical levels. In our study, we employ two hierarchical multiplier structures, i.e., KOA and broadcast, in the design, since the variations of combinations of these two techniques can representatively provide a relatively large set of multiplier architecture with different area/speed characteristics.

For ease of our presentation, we use an integer list, i.e. $M_n = \{m_1, m_2, \dots, m_N\}$, to represent a n by n hybrid multiplier with N hierarchical levels. Each element in the list, i.e., $m_i > 0$, represent the multiplication scheme adopted at the specific level, i.e., level i . The KOA scheme is applied at the i th level if $m_i = 1$, or the broadcast technique with k multiplication units is used if $m_i = k > 1$. After hierarchically decomposed, the hybrid multipliers need a set of monolithic base multipliers.

We can take advantage of the high performance and resource efficiency of the built-in hardware multipliers in many Xilinx Virtex-II[®] chips. Therefore, we assume the multiplications with bit-width less than 18 bit are conducted with these hardware multipliers.

For example, a 192-bit hybrid multiplier, $M_{192} = \{1, 1, 3\}$, has three abstraction levels. At the first level, KOA scheme is adopted which requires three 96-bit multipliers. For each of the 96-bit multipliers (the 2nd level), the KOA scheme is adopted again requiring three 48-bit multipliers each. For each of the 48-bit

multipliers (the 3rd level), the broadcast scheme with three 16-bit multipliers is used, and finally, totally 27 built-in hardware multipliers are used as the base multipliers.

4. Analysis of the Estimator

Accurate area and delay characteristics of different multiplier architectures are crucial for the high level synthesis of new design. In regard to this, designers can collect these technical parameters through the synthesis and layout process with the EDA (Electronic Design Automation) tools. Unfortunately, this strategy is time consuming and can only be applied for a small set of different multipliers, which cannot be effectively used in the primary synthesis of a new design. In this section, we estimate the performance characteristics of a given multiplier architecture scheme *analytically*, by taking advantage of the regularity in the hybrid multipliers. As demonstrated in next section, the performance metrics predicted with our analytical method has a very low percentage error over actual results generated through synthesis and layout.

4.1 Resource and delay estimation for KOA and BC multipliers

Both KOA and BC multipliers are hierarchical multipliers having strong regularity. Their resource usage and timing can be formulated recursively from their implementation schemes.

Specifically, for a n-bit width KOA multiplier, according to equation (1) and (2), the resource usage ($S_{KOA}(n)$) can be formulated as

$$S_{KOA}(n) = 3S\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4 * S_{ADD}(n) + 4 * S_{ADD}\left(\left\lceil \frac{n}{2} \right\rceil\right) + S_{koactl}(n) \quad (3)$$

where $S(n)$, $S_{ADD}(n)$, $S_{koactl}(n)$ represent the resource usage for the n-bit multiplier, n-bit adder, and the corresponding control logic, respectively. Note that four $\lceil n/2 \rceil$ adders are added in formula (3), which are used to compute a_i . Even though according to (1), only two adders/subtractors are needed for (A_H-A_L) and (B_H-B_L) , it would require a *signed* integer multiplier to compute a_i . This makes it difficult to further decompose this multiplier with the unsigned multiplication schemes as we discussed before. To solve this problem, we use two

subtractors each to compute the absolute values for (A_H-A_L) and (B_H-B_L) , and their sign bits are used to determine if the absolute value of a_i should be added to or subtracted from the sum of (a_2+a_0) later in computing the final product.

Analyzing the critical path from Fig. 2 can readily derive the delay estimation. Specifically, the delay ($T_{KOA}(n)$) for a n-bit width KOA multiplier can be formulated as

$$T_{KOA}(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4T_{add}(n) \quad (4)$$

where $T(n)$ is the delay for the n-bit multiplier, and $T_{add}(n)$ is the delay for a n-bit adder.

The area and timing cost for a BC type multiplier can be analyzed the same way. As explained in section 2.2, an n-bit Broadcast multiplier with k sub-sections requires total k multipliers and two adders, one for computing the partial product in each round and the other for the accumulating the partial product. Therefore, the resource usage (denoted as $S_{BC}(n, k)$) can be formulated as follows.

$$S_{BC}(n, k) = kS(p) + S_{ADD}(n) + S_{ADD}(n+p) + S_{bcctl}(n, k) \quad (5)$$

where $p = \lceil n/k \rceil$ and $S_{BCCTL}(n, k)$ represents the control logic resource usage for this strategy since the complexity of the control depends not only on the number of bits but also how many sections the operands are split.

The delay for a BC multiplier can be estimated from Fig. 3. To obtain the final production, the BC multiplier needs to go through k rounds, with each round containing the time for k multiplication which can be done in parallel, and two additions. Therefore, the delay ($T_{BC}(n)$) for a n-bit BC multiplier can be formulated as follows.

$$T_{BC}(n) = k(T(p) + T_{add}(n) + T_{add}(n+p)) \quad (6)$$

4.2 Resource and delay estimation for the hybrid multipliers

As the hybrid multipliers are constructed hierarchically, and the implementation strategies adopted at different levels are independent of each other. We can estimate both the resource usage and delay for the hybrid multiplier based on our analytical results as shown before.

Specifically, given a hybrid multiplier $M_n = \{m_1, m_2, \dots, m_N\}$, we have $S(M_n) = S_i(n)$ and $T(M_n) = T_i(n)$, where

$$S_i(n) = \begin{cases} 3S_{(i+1)}\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4*S_{ADD}(n) + 4*S_{ADD}\left(\left\lceil \frac{n}{2} \right\rceil\right) + S_{koact}(n), & \text{if } m_i = 1 \\ kS_{(i+1)}\left(\left\lceil \frac{n}{k} \right\rceil\right) + S_{ADD}(n) + S_{ADD}\left(n + \left\lceil \frac{n}{k} \right\rceil\right) + S_{bcctl}(n, k), & \text{if } m_i = k > 1 \end{cases} \quad (7)$$

and

$$T_i(n) = \begin{cases} T_{i+1}\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4T_{add}(n), & \text{if } m_i = 1 \\ k(T_{i+1}\left(\left\lceil \frac{n}{k} \right\rceil\right) + T_{add}(n) + T_{add}\left(n + \left\lceil \frac{n}{k} \right\rceil\right)), & \text{if } m_i = k > 1 \end{cases} \quad (8)$$

Equation (7) and (8) provide an analytical method to estimate the resource usage and delay amount for a hybrid multiplier. However, to get the concrete values, we still need to measure the resource usage for the adders and control logic, as well as the timing for the adders, which depends on the actual platform as well as the clock rate of the design.

4.3 Functioning of the Estimator

We target our high bit width multiplier design on a Xilinx Vertex-II[®] FPGA chip running at 100Mhz. For simplicity, we simply let the synthesis tool to take care of the adders. After conducting a series of experiment, we found that adders with bit width no more than 64 bits can be operated at the 10 ns as required. For adders with more than 64 bits, we simply use multiple cycles to compute the addition. We also found that the number of slices consumed by an adder is always half the operand bit-width by the synthesis tools (Xilinx ISE 6.1i), which make the resource usage for adder a trival problem.

Now the problem becomes how to estimate the resource usage for the control logic. Menn *et. al* [13] presented a method for estimating the resource usage based on the number of states in the design. Their method is more suitable in the controller design where there is strong correlation between the complexity of the control and number of states. Due to the high regularity of the hierarchical structure, it is conceivable that the complexity of control logic varies more significantly with the size of input rather than the complexity (i.e., the number of states) of the control. We therefore assume a linear relationship between the usage of the control logic and the the input size. Specifically, we have

$$S_{koactl}(n) = \alpha_1 n + \beta_1 \quad (9)$$

and

$$S_{bcctl}(n, k) = \alpha_2 n + \beta_2 k + \gamma_2. \quad (10)$$

To identify the parameters, i.e., α_1 , β_1 , α_2 , β_2 and γ_2 , we apply the traditional linear regression methods [14]. We randomly choose a set of five multipliers i.e. 24, 32, 48, 64 and 96-bit of each type as our sample models. These models are then designed, simulated to check for functional correctness, and synthesized to obtain the actual resource usage, i.e., slice usage in FPGA. These numbers are used as inputs in the linear regression model to compute the parameters. Through our experiments, we obtained parameter values as $\alpha_1=5.28$, $\beta_1=-14.26$, $\alpha_2=1.92$, $\beta_2=-3.03$ and $\gamma_2=17.4$.

The Estimator for hybrid model is a C routine that substitutes these parameters in the analytical equations for KOA and BC topologies, recursively, to estimate the area cost for any size hybrid multiplier. The C routine also estimates the timing cost, in number of clock cycles, by recursively analyzing the timing cost of sub-units.

5. Validation Experiments

In this section, we present experimental results evaluating the effectiveness of our approach. We target our design for 256 bit multipliers at a Xilinx Vertex-II[®] FPGA chip [15] (with total 33,792 slices and 144 built-in hardware multipliers), running at 100Mhz. From our initial experimental studies, the commercial tools,

namely the XST[®] tools (part of the Xilinx ISE[®] tool set), fail to synthesize such a large design due to the excessive resources requirements (slices and/or built-in Virtex-I[®] 18x18 hardware multipliers).

256-bit Hybrid multiplier	Cycles		Slices		
	Actual	Estimate	Actual	Estimate	% error
M ₂₅₆ {1 1 1 1}	38	38	17564	17665	0.58
M ₂₅₆ {1 1 1 2}	43	43	12917	13507	4.57
M ₂₅₆ {1 1 2 1}	50	50	11891	12274	3.22
M ₂₅₆ {1 1 4}	54	54	7054	7360	4.34
M ₂₅₆ {1 4 1}	73	73	6374	6610	3.70
M ₂₅₆ {1 2 4}	87	87	4385	4684	6.82
M ₂₅₆ {2 1 4}	98	98	4086	4240	3.77
M ₂₅₆ {2 4 1}	136	136	3649	3740	2.49
M ₂₅₆ {4 4}	156	156	1615	1584	1.92

Table 1. Comparing Estimated to Actual.

For ease of our experiments, we assume that the multiplicand is split into no more than 6 sub-units. Therefore, a 256-bit multiplier can have a maximum of four hierarchy levels, giving us a total of $6^4=1296$ possible candidate combinations. It would be extremely time consuming, if not practical possible, to evaluate all these different designs. However, we can estimate each of the design exhaustively with our estimators within one second. Among these design alternatives, we then prune the space by reserving only the Pareto-Optimal solutions [16]. That is, for any two given designs under consideration, no given one is clearly better than the other in terms of both resource usage and computation delay. Pareto-Optimal solutions play a significant role for making the design tradeoffs during the high level synthesis.

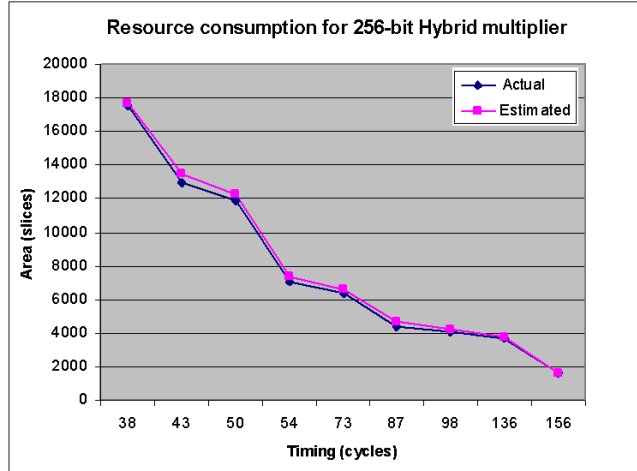


Fig. 4. Comparing Estimator versus Synthesis & Layout.

Fig. 4 graphs the Area-Delay curve for a range of possible architecture choices considered in this paper, with different area-time trade offs. For example, the multipliers can take as few as 38 cycles, but consumes as many as 17665 (estimated) slices. Also, it can consume much less resources (as low as 1615) but require as many as 156 cycles. In addition, even though there are potentially a large number of possible combinations in the hierarchical multiplier design, only nine of them are possible design choices. Without our estimation techniques, to find such a set can be quite challenging.

Our first set of experiments evaluates the accuracy when applying our estimator to project the resource and delay characteristics of a hybrid multiplier. Our second set of experiments present a set of design alternatives for the 256 bit multipliers that have different resource usage and delay characteristics.

To validate the accuracy of estimated results we create the requisite designs in VHDL and synthesize these multipliers to obtain the actual area cost for comparison with the estimated results. These are presented in Table 1 and plotted in Fig. 4 on an Area-Delay curve.

It is interesting to note that, in Table 4, the estimated delay exactly matches to the actual results as captured from simulation (using Mentor’s ModelSim SE® 5.7d). This is because equation (8) can precisely determine the timing characteristics of a hybrid multipliers as long as that for the base multipliers and adders are precisely captured. Moreover, as shown in Table 4, the relative deviations of the estimate results to the actual results are under 7%. Overall, our experiments show that by exploiting the regularity of the hierarchical multipliers, our

hybrid multiplier model and related analytical area/performance estimations can be effectively used in automatic design exploration for implementing efficient high bit-width multipliers.

6. Conclusions and Further Work

In this paper, we have presented a means to devise a set of candidate architectures for large operand-width Integer multipliers, which would be suitable for crypto-arithmetic applications. We briefly discussed how such architectures will become more important as embedded systems designers seek to move more stream-oriented functionality into custom or programmable logic as part of SoC or more loosely-coupled embedded systems design platforms. We presented the architecture of the basic patterns for achieving flexible and efficient large multipliers, and also outlined a method for estimating the location of a given candidate architecture in the continuum as represented by the Area-Delay tradeoff curve. Our hybrid model can represent a large spectrum of high bit width multipliers with different area/time trade off. Furthermore, our analytical Estimator model makes design exploration among candidates in this large solution space possible and very practical. Finally, our experimental results clearly indicate that our techniques are reasonably precise—thus making this method practical for use in actual architecture analysis of embedded systems, constructed either using the SoC method or the more loosely-coupled programmable logic approach.

The relevance of the problem is readily apparent, if one considers the complexity of implementing the AES encryption algorithm. As discussed in [7], the inventory of arithmetic processing units to carry out the necessary mathematical operations (finite field arithmetic over a Galois field $GF(p)$ for some prime number p) comprises of 14 multiplications (of which 2 are squarings), 7 additions and 2 bit-shift operations. Each of these multiplication operations must be performed modulo some prime number p , in order to reduce the bit-width of the resultant product [7]. As such, we have been working with Montgomery multiplier architectures in the construction of elliptic curve addition units [7]. According to the Montgomery algorithm [20], each modular multiplier requires 3 base multipliers of the same operand width, on which the modular reduction takes place [7]. So, for the AES encryption algorithm that is part of the IEEE 802.11i standard extensions,

given an operand bit width of 192-bits (which we have been using as an example in this paper), we would require 42 multipliers, each of which must multiply 192x192 bit Integers. As we postulated in this paper, there are a number of candidate architectures for realizing this operation using custom logic or programmable logic as the substrate on which these computations take place—for which we could subdivide these expensive operations. However, it should be clear from the discussion that performing these operations strictly out of an embedded processor will be very costly, and might be problematic, given the performance requirements of the IEEE 803.11g WLAN standard [20] (having an upper limit throughput defined as 56 Mbits per second, operating in the 2.4 GHz frequency spectrum). This provides strong motivation for considering alternate architectures for embedded systems for wireless LAN implementation using SoC and/or programmable logic techniques.

We have not considered the impact of the use of a higher degree of parallelism that is possible in the construction of the constituent wide Adder units that would likely improve performance. Consideration of this part of the problem is being investigated at present, so there are no results to present in context of this paper. Thus, in future work, we will be incorporating a greater spectrum of wide-bit adder architecture models (for example, incorporating parallelism inherent in Wallace tree adder schemes [10], increasing the level of parallelism in the multiplication supported in systolic structures [17, 18], or in formulating our combinational logic primitives in alternate Boolean representations, as in [19]) in the overall formulation of multiplier estimates. In addition, we are extending the set of basic architecture patterns, incorporating other multiplier architectures, such as aforementioned tree structured and systolic architectures, into our analytical model and estimation method.

7. References

- [1] D.A. Buell, J.M. Arnold, and W.J. Kleinfelder (eds.), *Splash-2: FPGAs in a Custom Computing Machine*, IEEE Computer Society Press, Los Alamitos, CA, 1996.
- [2] S. Hauck, “The roles of FPGAs in reprogrammable systems,” *Proceedings of the IEEE*, Vol. 86, 1998, pp. 615-639.
- [3] K. Compton and S. Hauck, “Reconfigurable computing: A survey of systems and software,” *ACM Computing Surveys*, Vol. 34, No. 2, June 2002, pp. 171-210.

- [4] DeHon, A., and Wawrzynek, *The case of reconfigurable processors*. Berkeley Reconfigurable Architectures Systems, and Software. University of California, Berkeley. <http://citeseer.nj.nec.com/dehon97case.html>.
- [5] Daemen, J., and V. Rijmen. *The Design of Rijndael: AES- The Advanced Encryption Standard (Information Security and Cryptography)*. Springer Verlag, Berlin, 2001.
- [6] National Institute of Standards and Technology (NIST), *Recommended Elliptic Curves for Federal Government Use*, found at <http://csrc.nist.gov/csrc/fedstandards.html>, July 1999.
- [7] D.A. Buell, J.P. Davis, and G. Quan, "Reconfigurable Computing Applied to Problems in Communications Security", in *Proceedings MAPLD-02*, Laurel, MD, 2002.
- [8] Chodowicz, P., P. Khuon, and K. Gaj, "Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining", *Proceedings ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, Monterey, California, February 11-13, 2001.
- [9] Chen, M. C., and C. A. Mead, "Concurrent Algorithms as Space-Time Recursion Equations", in Kung, S. Y., H. J. Whitehouse, and T. Kailath (eds.), *VLSI and Modern Signal Processing*, Prentice-Hall Publishers, 1985.
- [10] Parhami, B., *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 2000.
- [11] D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley, 1998.
- [12] S.Y. Kung, *VLSI Array Processors*, Prentice Hall Publishers, Englewood Cliffs, NJ, 1988.
- [13] C. Menn and O. Bringmann and W. Rosenstiel. Controller estimation for FPGA target architectures during high-level synthesis. *International Symposium on System Synthesis (ISSS)*, 56-61, 2002.
- [14] Weiss, S. M., and N. Indurkha, *Predictive data mining: A practical guide*. New York: Morgan-Kaufman Publishers, 1997.
- [15] Xilinx, Inc., *Vertex-II 1.5V Field Programmable Gate Arrays: Advance Product Specification*, DS031-1, DS031-2, and DS031-3, September 22, 2002.
- [16] Keeny, R., and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, NY, 1976.
- [17] S.Y. Kung, *VLSI Array Processors*, Prentice Hall Publishers, Englewood Cliffs, NJ, 1988.
- [18] Fagin, B., "Large Integer Multiplication on Massively Parallel Processors", *Proceedings FMPC-90*, 1990.
- [19] Ciriani, V., F. Luccio, and L. Pagli, "Synthesis of Integer Multipliers in Sum of Pseudoproducts Form", *Integration: The VLSI Journal*, 36, 2003, pp. 103-119.
- [20] IEEE. 802.11g: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, IEEE Computer Society, 2003, available at <http://grouper.ieee.org/groups/802/11/>.