High-Level Synthesis for Large Bit-Width Multipliers on FPGAs: A Case Study

Gang Quan, James P. Davis, Siddhaveerasharan Devarkal, and Duncan A. Buell
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208 USA
{gquan, jimdavis, buell}@cse.sc.edu}

ABSTRACT

In this paper, we present the analysis, design and implementation of an estimator to realize large bit width unsigned integer multiplier units. Larger multiplier units are required for cryptography and error correction circuits for more secure and reliable transmissions over highly insecure and/or noisy channels in networking and multimedia applications. The design space for these circuits is very large when integer multiplication on large operands is carried out hierarchically. In this paper, we explore automated synthesis of high bit-width unsigned integer multiplier circuits by defining and validating an estimator function used in search and analysis of the design space of such circuits. We focus on analysis of a hybrid hierarchical multiplier scheme that combines the throughput advantages of parallel multipliers and the resource cost-effectiveness of serial ones. We present an analytical model that rapidly predicts timing and resource usage for selected model candidates. We evaluate the estimator model in the design of a practical application, a 256-bit elliptic curve adder implemented on a Xilinx FPGA fabric. We show that our estimator allows implementation of fast, efficient circuits, where resultant designs provide order-of-magnitude performance improvements when compared with that of software implementations on a high performance computing platform.

Categories and Subject Descriptors
B.2.4 [**Arithmetic and logic structures**]: High-speed arithmetic – *algorithms, cost/performance.*

General Terms
Algorithms, Performance, Design, Experimentation

Keywords: Large-scale Integer Multipliers, High Level Synthesis, Design Exploration, Reconfigurable Computing, FPGA Devices.

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 programmable logic "fabric" consisting of one or more commercially-available field programmable gate arrays (FPGAs) [1][2]. Highly repetitive operations, such as pipelined arithmetic calculations on wide data words, and highly parallel SIMD operations—such as those found in cryptography and error coding applications—have been a good match for RCM-based high-performance computing and mobile computing, applications.

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.

Programmable logic design requires programming of an application using a hardware description language such as VHDL, which is then transformed onto an FPGA substrate using synthesis and layout tools. Since information on the performance and the resource usage required by the design is not determined until after synthesis and place-and-route steps, developers are usually engaged in an iterative design process that can be lengthy and error-prone. Extensive research (such as [3-4]) has been carried out in behavioral level design to leverage the productivity and effectiveness of FPGA design. Also, commercial tools, such as Mentor Graphics' Monet® [12] and Synopsys Behavioral Compiler® [13], are available that can realize efficient multiplication circuits. However, these tools treat arithmetic operations such as multiplication as atomic; thus, they work well for small operand bit-widths that can be performed on monolithic architectures. However, using this scheme for large bit-width operations can lead to extreme resource utilization and/or severely degraded performance—to the point of even making the whole circuit infeasible. In our preliminary experiments, commercial synthesis tools completely failed to synthesize a 256-bit integer multiplier for a Vertex II 6000 FPGA device.

In this paper, we present recent results in the design and implementation of high bit-width integer multipliers using an automated estimator technique. Such multipliers are now required in cryptographic and error correction circuit applications. In supercomputing applications using reconfigurable computing machines, such as the SRC 6-E® [17], these applications are coming into prominence—for example, the requirement of increased key lengths in cryptography, so as to thwart the brute force cracking of shared-key ciphers. However, most conventional microprocessor architectures are not optimized for computation on operands above 64 or 128-bits. Larger bit widths significantly degrade processor performance. It is from this perspective that we are exploring the efficient identification of optimal architectures for large bit width multipliers for implementation in FPGA fabrics that can either be used in supercomputing applications on reconfigurable computing platforms, or for implementation in supporting FPGAs in mobile computing platforms.

Due to performance and resource constraint issues, large bit-width multiplication operations are usually mapped onto a series of smaller bit-width units for creating partial product terms. The differences between small and large bit-width multiplier units are reasonably well-understood, as reported in [26]. These differ from one other in their performance, resource usage, scalability, flexibility, and so on. The search space for finding optimal application-specific designs for high bit-width multiplier circuits is very large.

Having a fast and effective evaluation method is critical to the success of design space exploration during high-level synthesis. As such, we have constructed an estimator model that rapidly predicts performance and resource usage for a large subset of candidates in a 256-bit design space, based on different performance/resource characteristics. Our Estimator model is parameterized and calibrated with sample data collected from commercial logic synthesis and place-and-route tools for multiplier permutations at the desired bit width. We

then applied this estimation model in the design of 256-bit elliptic curve cryptosystem (ECC) adder circuits over a Galois field *GF(p)* [16], to aid in selection of the most efficient 256-bit multiplier model for use.

In this paper, we show through our experiments that control logic can contribute significantly to the latency and resource usage in large complex multiplier architectures, particularly for those that need to employ heavy resource sharing. Estimation results can deviate significantly from actual results, as has been seen in approaches such as [11], where the impact of control logic is ignored in evaluating architectures. Our model incorporates heuristics to estimate the resource usage in generating the control logic and the data path so as to produce more accurate estimation. Our results indicate that estimator automation can result in high-performance circuits; when compared with a software implementation of the same ECC application on a 32-node Beowulf cluster computer, FPGA implementation using designs selected by our method can achieve an order of magnitude speedup and still meet critical area and device timing constraints.

The paper is organized as follows. In Section 2, we provide background on the application context and architecture assumptions in this domain. In Section 3, we introduce the hybrid multiplier architecture and Estimator models for obtaining timing and resource estimates for the target Xilinx device. In Section 4, we present our design exploration methodology and resource estimation incorporating both control and datapath logic. In Section 5, we validate our Estimator model and present implementation results.

2. THE DESIGN PROBLEM

A requisite target application for large integer operands is the Elliptic Curve Cryptography (ECC) public-key cryptosystem proposed in 1985 [14][15]. ECC rests its security on the discrete logarithm problem over the points on an elliptic curve, which is said to have no sub-exponential solution [14]. It can, therefore, use smaller key sizes and still offer the same level of security provided by other public key cryptosystems such as RSA [16]. Smaller key sizes make the ECC algorithm highly suitable for FPGA implementation [21].

The core of the ECC cryptosystem requires a fixed number of modular multiplication, addition, shift, and squaring operations; the actual number depends on the elliptic curve representation [16]. As a problem to explore large multiplier architectures, our version of ECC uses 14 modular multiplications for a point addition over a finite field GF(p) as shown in Fig. 1. We selected an ECC structure requiring 256-bit operands, as this is one of those recommended by NIST [19].

For the ECC Adder, the modular multiplication requires both a multi-precise multiplication and division, which dominates the resource usage and computation. Since division in hardware is expensive, the Montgomery multiplication scheme [20] is used instead of regular multiplication and modular reduction. By using this method, a modular multiplication is transformed into three ordinary multiplications and two low-cost shift operations. Our ECC Adder thus contains a total of 42 multi-precision multiplication operations.

The SRC-6E [17] is our target reconfigurable computing platform. It is a reconfigurable board consisting of two Multi-Adaptive Processor (MAP) modules, each with two programmable Xilinx Vertex II XC2V 6000 FPGAs, attached to the main microprocessor board, capable of being clocked at 100 MHz. While there are up to four user-programmable FPGAs residing on the same board, each one can be used independently. The source models for execution on the FPGA can either be written completely in a high-level language such as C, or using a combination of C and VHDL/Verilog code implementing hardware macro-functions that speed up compute-intensive functions. HDL code is synthesized using Synplify Pro®, and placed and routed using Xilinx ISE®.

The key to realizing high-performance ECC Adder execution on the SRC-6E platform is in the design of the 256-bit integer multiplication unit. To avoid a time consuming ad-hoc iterative design process, system programmers need to conduct fast and accurate prediction of the performance and resource outcomes for different design alternatives. As such, extensive work is reported (e.g. [7],[8],[10],[11]) that provides evaluation metrics and corresponding estimators for applications expressed using data flow graph (DFG) methods or high-level language such as C. Unfortunately, these approaches cannot be readily applied to our design problem, as they treat the high bit-width multiplication simply as an atomic operation. As stated earlier, a 256-bit multiplier cannot be synthesized successfully when specified as a single, monolithic arithmetic operator targeted to FPGA devices using current EDA tools. Even if such a multiplier could be synthesized, it would be timing and resource inefficient if all the multiplications were required to use or share the same type of constituent multipliers. Considering the large design space, the critical

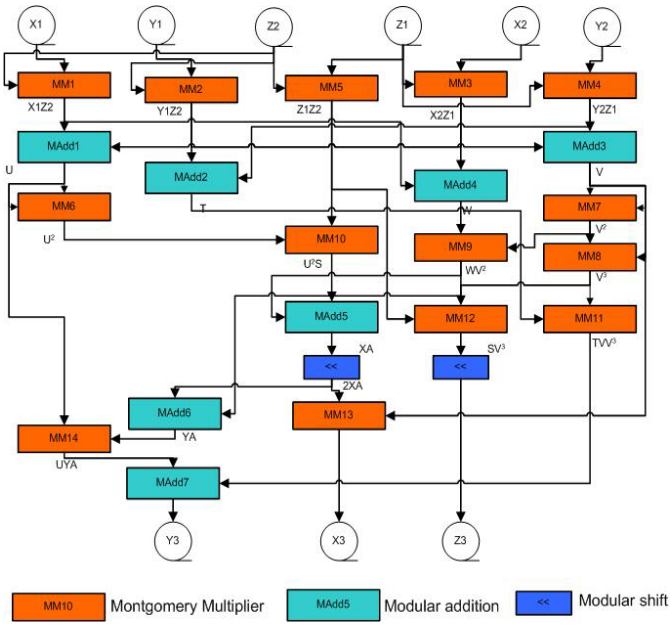

Figure 1 Data flow graph for elliptic curve addition: P3(x3,y3,z3)=P1(x1, y1, z1) + P2(x2,y2,z2)

issues for facilitating design-assist are: (1) making performance and resource tradeoffs in selecting the multiplier architecture; and, (2) determining how the multiplication operations will share the multiplier resources in the larger computation. We discuss these points later in this paper.

In what follows, we first present our development of the estimation model, and tune the parameters of the model using sampling points. We then apply our estimation model to prune the search space and select optimal targets for the ECC Adder application.

3. ESTIMATION MODEL

We construct an estimation model for timing and resource usage evaluation in order to identify and select an optimal design for our intended application. Resource usage is critical, as we need to "fit" the design on the target FPGA device; otherwise, we incur performance penalties by having to go "off chip" to complete the intended computation. 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. Large bit-width multiplication forms the bulk of the computation in our application. We first present the most prevalent architecture topologies for large bit-width multipliers and then develop the corresponding estimation models.

3.1 Large Bit-Width Multiplier Architectures

When constructing large-width multiplier data paths, a first-cut approach is to specify a "monolithic" (i.e., flat) multiplier that takes two large word operands and multiplies them to generate partial products, and to add the partial products as one would do using the "pencil and paper" method employed with operands of 8, 16, or 32 bits. However, it is impractical to construct such monolithic circuits of very large bit-widths, say, greater than 64 bits, for the 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 architecture is likely to be difficult to synthesize onto a fixed-size device. Therefore, we must model our different multiplier schemes as a hierarchical decomposition of smaller, more basic units, from which we construct the larger units. These units are organized hierarchically, in that a large bit-width unit contains some number of units that execute smaller MUL 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 given algorithm being used.

Such an 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. In this paper, we focus on two basic hierarchical architecture topologies for large bit-width multiplication. We then take both of these and combine them into a number of possible hybrid configurations.

3.1.1 Divide and Conquer Multiplier

The first multiplier topology in our model is the Divide and Conquer scheme. There is a "naïve" version [26] and an optimized version [22]. 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-bit multipliers are required.

To reduce the resource requirements, the Karatsuba-Ofman Algorithm (KOA) [22] 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-bit and one (n/2+1)-bit multiplication operations are necessary. Moreover, in order to use the same bit-width multiplier for all multiplications, the KOA algorithm can be implemented in a slightly different way as shown below. Let

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

Then

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

and only three n/2-bit multipliers are required. Even though the basic 2-way method can theoretically be extended into an m-way method, the practical control overhead (including interconnect and control logic) will likely outweigh the potential performance benefits [22].

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 achieved with excessive resource usage in terms of circuit area and required components, which can easily consume the hardware resources and makes it impossible to synthesize the high bit-width multiplier on a single FPGA.

3.1.2 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 in [21] 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 $p = \lceil n/k \rceil$. Then, we can partition both A and B into *k* blocks such that

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

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

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

The BC multiplier is, in fact, a sequential block-based shift-and-add multiplier. The computation efficiency of the BC multiplier comes from the fact that multiplication cost for generating the partial products usually dominates the accumulation cost. Since a BC multiplier has *k* independent sub-multipliers, the computations of the *k* sub-products can be computed in parallel during each sub-operand pass. Compared with the KOA scheme, the BC multiplier has an advantage in that it is more flexible in terms of selecting the sub-unit's 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.

3.1.3 The Hybrid Multiplier Scheme

As explained before, the KOA and BC 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 BC, 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 topology to achieve the goal of designing 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 BC as introduced before, in the design, since the variations of combinations of these two techniques can provide a relatively large number of candidate multiplier architectures with different area/speed characteristics.

For ease of our presentation, we use an integer list, i.e. $M_n=\{m_1, m_2, ..., m_N\}$, to represent a n by n hybrid multiplier with N hierarchical levels. Each element in the list, i.e., $m_i>0$, represents the multiplication scheme adopted at the specific level, i.e., level i. The KOA scheme is applied at the ith level if $m_i=1$, or the BC 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. 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. Therefore, we assume the multiplications with bit-width less than 17 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 therefore, totally 27 built-in hardware multipliers are used as the base multipliers.

3.2 Hybrid Multiplier Estimator

As the hybrid multipliers are constructed hierarchically, and the implementation strategies adopted at different levels are independent of each other. We can formulate both the resource usage and delay for the hybrid multiplier analytically.

Specifically, given a hybrid multiplier $M_n=\{m_1, m_2, ..., m_N\}$, the area cost for M_n (i.e., $S(M_n)$)can be recursively estimated as $S(M_n) = S_1(n)$, and

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

where $S_i(n)$ is the area requirement at level i, $S_{ADD}(n)$ is the area cost for n-bit adders, and $S_{koactl}(n)$ and $S_{bcctl}(n)$ are the logic control area overhead corresponding to the n-bit KOA and BC scheme and will be identified later. Similarly, the execution cycles for M_n (i.e. $T(M_n)$) can be predicted as $T(M_n) = T_1(n)$, and

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

where $T_i(n)$ is the number of execution cycles at level i, and $T_{ADD}(n)$ is the timing cost for n-bit adders. The details for the development of equation (1) and (2) are omitted due to the page limit. Interesting readers can refer to [24] for more technical details.

Equation (1) and (2) 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. Based on our target platform and after conducting a series of experiments, 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 such that the adder units can be reused. 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 makes the resource usage for adder sub-units a trivial problem.

Now the problem becomes how to estimate the resource usage for the control logic. Menn et al. [18] presented a method for estimating the resource usage based on the number of states in the design. Their method is more suitable in the control oriented 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 input size. Specifically, we assume

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

and

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

To identify the parameters, i.e., α_1, β_1, α_2, β_2 and γ_2 , we applied traditional regression methods [14]. We randomly chose a set of five multipliers, i.e. 24, 32, 48, 64 and 96-bits, as our sample models. These models were then specified in VHDL, simulated to check for functional correctness, and synthesized to obtain the actual resource usage, (i.e., slice usage) in the target FPGA. These numbers were used as inputs in the regression model to compute the parameters. Through our experiments, we obtained parameter values as α_1=5.28, β_1=-14.26, α_2=1.92, β_2=-3.03 and γ_2=17.4. As demonstrated later, the performance metrics predicted with our analytical method have a very low percentage error over actual results generated through synthesis and layout.

4. DESIGN SPACE EXPLORATION AND SYSTEM CONTROL ESTIMATION

Having modeled timing and resource usage for high bit-width multipliers, the most critical processing element in ECC adder, we started the design space exploration. The process starts with identification of the potential multipliers that can be used in our benchmark design. This can be achieved by exploring the permutations of hybrid multipliers and obtaining a Pareto-optimal [23] set of candidates. That is, for any two given multipliers, no given one is clearly better than the other in terms of both resource usage and computational latency. Pareto-optimal solutions play a significant role for making the design tradeoffs.

After identifying the candidate multipliers, we continued with the design space exploration including allocation, scheduling, and binding. If N is the maximal number of multipliers that can fit in one FPGA chip and M is the size of the Pareto-optimal set, the there exists maximal M^N ways of selecting types and numbers of candidates in the final design. The Pareto-optimal set that we identified is small—with only total 9 choices; therefore it was reasonable to explore all possible allocation results (types and numbers of different processing units) via an exhaustive search. For data path scheduling and binding, we simply adopted the traditional list scheduling scheme [25].

With the output from list scheduling, we can estimate the computation cost for a design alternative; estimating resource usage, however, needs more careful analysis. Note that, using estimation methods presented in other work, we are able to gauge the resource usage only for the data path in our design. This is usually tolerable for small designs employing simple control schemes. However, our large bit-width multipliers require considerable resource sharing, as the number of multipliers that can be put into one single FPGA is quite limited. Recall that 42 large bit-width multiplications are required in one ECC addition operation. This implies extensive resource sharing in our design, requiring a more intricate control regime for pipelining. The resources consumed by this control logic can be comparable to, or even that of, the processing units in the data path; the control resources must

be considered to accurately predict the total resource usage in the candidate designs.

Intuitively, the resource usage by the control logic is more tightly coupled with the control step rather than the bit-width of the operators. Let the number of total control steps (available from list scheduling output) be x and let $X=\lceil \log_2 x \rceil$. Suggested by the work in [9][18], we assume a quadratic relationship between the resource requirement by control logic and value of X. Therefore, the resource usage for control (i.e, $S_{ctl}(X)$) is estimated using

$$S_{ctrl} = K_1 X^2 + K_2 X + K_3. \quad (3)$$

Again we use regression to determine the parameters. We synthesized three versions of ECC adders (44-bit, 96-bit, and 128-bit) and collected their resource usage. Their datapath resource usages are estimated as introduced above, and the differences are treated as the resource consumed by the control logic. From our experiment results, we obtained the values as K_1=2811.03, K_2=-50038.78, and K_3=222243.20. These parameters and equation (3) are then used to estimate the resource usage for the system control and added to the data path resource usage to estimate the overall resource usage for a design alternative.

5. VALIDATION AND EXPERIMENTAL RESULTS

In this section, we validate the accuracy of our estimation model and present our final design results for 255-bit ECC Adder. Our first set of experiments evaluates the accuracy of our estimation model for hybrid multipliers. Based on our tuned hybrid multiplier model presented in Section 3.2, we searched and found 9 different "short-list" 256-bit hybrid multiplier candidates in the Pareto-optimal set derived from thousands of possible alternatives. We created these 9 multipliers in VHDL and synthesized them to obtain the actual timing and area cost data. The results are compared with the estimated results in Table 1.

As shown in Table 1, the estimated delays (from formula (2)) exactly match to the actual results as captured from simulation (using Mentor's ModelSim SE 5.7d). This is because equation (2) can precisely determine the timing characteristics of a hybrid multiplier as long as those for the base multipliers and adders are precisely characterized. Moreover, as shown in Table 1, the relative deviations of the estimate results to actual results are very small, with an average error (3.49%) less than 5%. These results demonstrate that 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.

The essential goal of our project is to design and implement 256-bit ECC Adder on a Virtex-II® FPGA fabric. We exploited the design space, as introduced in Section 4, locating one candidate, and implemented it on the Xilinx Vertex-II XC2V6000 FPGA (with total 33,792 slices and 144 built-in hardware multipliers), running at 100 MHz. Table 2 shows the resource usage of our design.

As shown in Table 2, the total area-cost difference between the estimated result and the actual one is around 20%. Comparing against the component results in Table 1, we believe the difference derives from the estimation error for the control path. Since the large bit-width multipliers have high resource-cost and must be intensively shared, the control path consumes significant amount of slices. As also shown in Table 2, it is evident that using the estimated resource requirements for the data path components to predict the whole design can be problematic—the actual design requires 61% more resources as the predicted results.

Table 1 Comparison of the estimated results to the actual timing and resource cost obtained from the synthesized results

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

Finally, we compared the performance of the resultant design against a benchmark software implementation of 256-bit ECC Adder, as shown in Table 3. The software version was implemented on a 32-node Beowulf cluster, with each node using a 933MHz Pentium III and 1GB of memory. The application was programmed [21] in C, using GNU Multi-precision Library (GMP) routines for multi-precise integer arithmetic. The timing, in μsec, shown in Table 3, is for one elliptic curve point addition. As indicated in Table 3, we realize an 8X speedup in performance from our ECC Adder over its implementation on a high performance software platform.

6. SUMMARY

The extreme flexibility and increasing capacity of FPGAs makes them an excellent choice for custom computing applications, and at the same time imposes tremendous design challenges to developers. While there have been extensive research published for optimizing FPGA-based designs, many developers still resort to high-level global performance optimization by hand, due largely to the limited optimization methods supported in available EDA tools for designs of this type.

To study the limitation of the current EDA tools and examine our behavioral synthesis methodology in practical application design, we presented an empirical study of designing and implementing a 256-bit ECC Adder over GF(p) on an FPGA fabric. As large bit-width multiplication is critical component in our application, we proposed a hierarchical architecture for hybrid multiplier circuits in which we can

Table 2 Resource Usage for 255-bit ECC Adder

Multipliers		Slices			MULT 18x18
#	Types	Area est.	Area act.	Datapath only est.	
9	Same ($M_{256}\{4\ 4\}$)	31059	25860	16084	144

Table 3 Comparison of software and hardware implementation of ECC Adder over *GF(p)*

Bit-width	Timing (μsec)		Speed-up
	Software	FPGA	
256	196.72	24.56	8.01

combine the performance advantages of parallel multipliers and the resource cost-effectiveness of serial ones. To facilitate the behavioral synthesis process, we built an analytical model that can rapidly predict the timing and resource values for a large subset of hybrid and hierarchical integer multiplier topologies. The estimation model was parameterized and calibrated with sample data collected using logic synthesis and place-and-route tools. As shown in our experiments, the resultant multiplier model can estimate the timing precisely and resource usage with error less than 5%.

Based on this calibration, we further developed an estimation model—predicting the resource usage required by both data path components and control logic—to prune the search space and locate the Pareto-optimal design choice for our intended application, which is then programmed with VHDL and realized on the FPGA fabric. Benchmarked against a software implementation on a high performance computing platform, the resultant implementation from using our technique still achieves an 8X improvement in overall computation speed while also fitting on the Virtex-II device.

On the downside, we observed that the set of heuristics identified thus far will not allow us to make reasonable estimates of the entire application working off the constituent estimates of the individual components—indicating that the cost is non-linear and has some additional control component that is not covered under our current control-oriented estimation heuristic.

Our work demonstrates the feasibility of using automated methods for design-assist in creating a class of large bit-width integer multiplier circuits. We believe this method is extensible to a wider range of possible topologies. We also believe that accuracy in estimation—when moving from the multiplier components to the whole application—can be improved through identification of additional heuristics, which is the subject of future research.

7. REFERENCES

[1] S. Hauck, "The roles of FPGAs in reprogrammable systems," Proceedings of the IEEE, Vol. 86, 1998, pp. 615-639.

[2] 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.

[3] R. P. Dick and N. K. Jha, "CORDS: Hardware-software Co-synthesis of Reconfigurable Real-Time Distributed Embedded Systems", ICCAD98, 62-69, 1998.

[4] N. Shenoy and A. Choudhary and P. Banerjee, "An Algorithm for Synthesis of Large Time-Constrained Heterogeneous Adaptive Systems", *ACM Trans. DAES*. Vol 6, No. 2, 207-225, 2001.

[5] S.C. Goldstein and M. Budiu, "The DIL Language and Compiler Manual," Carnegie Mellon University, www.ece.cmu.edu/research/piperench/dil.ps, 1999

[6] J. Hammes, R. Rinker, W. Bohm, and W. Jajjar, "Compiling a High-Level Language to Reconfigurable Systems," *CASES*, 1999.

[7] B. So, P. Diniz and M. Hall, "Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration", *DAC*, 2003

[8] B. So, M. Hall and P. Diniz, "A Compiler Approach to Fast Hardware Design Space Exploration in FPGA-based Systems," PLDI, 2002.

[9] C. Brandolese and W. Fornaciari and F. Salice, "An Area Estimation Methodology for FPGA Based Designs at SystemC-Level", DAC, 2004.

[10] A. Nayak *et al*, "Accurate Area and Delay Estimation for FPGA", *DATE*, 2002

[11] D. Kulkarni *et al*, "Fast Area Estimation to Support Compiler Optimizations in FPGA-based Reconfigurable Systems," *FCCM*, 2002.

[12] Mentor Graphics Inc. *Monet™ User's Manual*, 2002

[13] "CoCentric SystemC Compiler, Behavioral User and Modelling Guide," Synopsys,http://www.synopsys.com

[14] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203-209, 1987.

[15] V. S. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology: Proceedings of Crypto '85*, LNCS 218, pp. 417-426, New York: Springer-Verlag, 1986.

[16] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, October 1996.

[17] SRC High Performance Computer www.srccomp.com/default.htm, 2004

[18] C. Menn *et al,* "Control Estimation for FPGA Target Architectures During High-Level Synthesis," *ISSS*, 2002

[19] 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.

[20] Peter L. Montgomery. "Modular multiplication without trial division". *Mathematics of Computation*, 44(170):519--521, 1985

[21] D.A. Buell, J.P. Davis, and G. Quan, "Reconfigurable Computing Applied to Problems in Communications Security", in *Proceedings MAPLD-02*, Laurel, MD, 2002.

[22] D.E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, 1998.

[23] U. Junker, "Preference-based search and multi-criteria optimization", *8th National Conference on AI*, 2002

[24] U. Junker**, "**Design space exploration of Elliptic curve arithmetic on a reconfigurable platform," MS Thesis, 2004

[25] G.D. Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, 1994

[26] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, 2000