EFFICIENT ADDERS TO SPEEDUP MODULAR MULTIPLICATION FOR CRYPTOGRAPHY

Adnan Abdul-Aziz Gutub and Hassan A. Tahhan

Computer Engineering Department, KFUPM, Dhahran, SAUDI ARABIA

ABSTRACT

Modular multiplication is an essential operation in many cryptography arithmetic operations. This work serves the modular multiplication algorithms focusing on improving their underlying binary adders. Different known adders have been considered and studied. The carry-save adder, carry-lookahead adder and carry-skip adder showed interesting features and trade-offs. The adders VHDL implementations gave some more beneficial details promising for improved crypto designs.

***Keywords**: Cryptography hardware, Modular multiplication, Binary adders in cryptosystems.*

1. INTRODUCTION

Modular multiplication is the fundamental arithmetic operation in most cryptographic systems. The crypto operands size needed to achieve secured systems is usually very large; which makes the modular multiplication operation too lengthy. Several modular multiplication algorithms have been proposed to solve this problem based on various computer arithmetic principles where all depend heavily on the binary adders as their basic building blocks. Most multiplication approaches perform operations through repeated number of additions, which makes our focus of this paper. Parts of this work have been presented earlier in [1].

This work discusses different types of adders and compares between them and their effect in selected modular multiplication hardware. Hardware components studied in this work were modeled using VHDL and synthesized. ModelSim Se 5.6a was used to design and test the VHDL code. The Xilinx ISE 5.2i tool was used for synthesis.

2. MODULAR MULTIPLICATION

Modular multiplication is defined as the computation of $P=A\times B \bmod M$, which are represented using n bits. In the literature, various algorithms for modular multiplication have been proposed with different hardware implementations. A straightforward way to perform the modular multiplication operation is to do the multiplication first and then divide the result by the modulus. This method is too expensive in terms of time and area requirements. An $N\times N$ bit multiplier is required followed by a $2N\times N$ bit divider used to computer the remainder of the division process. Since the multiplication result of $A\times B$ is not important to the result of the modular multiplication, doing the reduction during the multiplication step is more suitable and also more efficient for the modular multiplication problem [2]. This work applies to the basic common algorithms for modular multiplication of good performance so it might serve as an introduction to this widely evolving field.

Most related multiplication algorithms are described in the literature using binary representation, so that direct mapping with the hardware can be induced from it. Our binary adders research is concerned with two implemented algorithms: interleaving multiplication and reduction [3], and Montgomery's method [4]. The multipliers proposed in references [3] and [4] are chosen since they have efficient performance and similar hardware structure. Details of both algorithms can be found in the mentioned references. The two implementations were modeled using VHDL and synthesized using the Xilinx synthesis tool (ISE Series 5.2i). Summarize of the differences between the two algorithms are found in [1].

In general, the classical method in reference [3] takes slightly larger amount of hardware than the Montgomery's method in reference [4]. In addition, the multiplication step of the classical method takes slightly more time than Montgomery's method. Moreover, it has a correction step at the end of the algorithm that will complicate the hardware for the final summation circuit.

However, a full-custom design of the sign-estimation logic needed by the classical method will reduce the latency to its minimum. The carry-save logic and the sign-estimation logic are both of three logic levels. This means that parallel execution of the two logics will take about the same time as one individual carry-save stage. Hence, the multiplication steps of two algorithms are expected to have the same latency.

By considering the pre and post-calculations needed by both algorithms, we see that Montgomery's method needs much more expensive calculations. Reference [5] shows how practically pre and post-calculations of Montgomery's method take very long time over the multiplication step. These calculations doubled the amount of time required by Montgomery's method by more than 20 times. In addition, the hardware was

complicated. Thus, carry save implementations of the classical method of interleaving multiplication and reduction has the potential to be one of the most effective solutions in terms of time and hardware requirements. Further improvements by researchers on the classical method would lead to a high speed modular multiplier which is scalable and regular by its nature. The reader is referred to [1] for more remarks on the implementation by Koc in [3], which is a promising hardware design for modular multiplication. The adders in all modular multipliers play a curtail role, which if improved the overall process performance will improve. These adders study will be the focus of the rest of the paper.

3. BINARY ADDERS

Binary addition is one of the most important operations in modular multiplication as well as all digital computer systems. Statistics showed that more than 70% of the instructions perform additions in the data path of RISC machines [7]. In modular multiplication hardware, binary addition is considered the bottleneck [1]. Thus, the cryptography computation time is considerably affected by the speed of adders.

In the literature, there exist many types of adders with different time and space complexities. The focus in this work is on some practicable and commonly used binary adders which will be discussed and compared. For the sake of completeness, the ripple-carry adder and the carry-save adder are presented first. Then, the carry-lookahead adder and the carry-skip adder are studied.

3.1. Ripple-Carry Adder (RCA)

The ripple-carry adder consists of a sequence of cascaded Full Adders (FA) by which each FA computes the i_{th} bit of the result according to the following logical relations:

$$s_i = a_i \oplus b_i \oplus c_i\ ,\ c_{i+1} = a_i\, b_i + (a_i + b_i)\, c_i\ ,$$

where $i = 0, 1, \ldots, n\text{-}1$

Obviously, both the logic complexity and the worst case computation time are O(n). The ripple-carry adder required the optimum space and has the worst computation time among its category of parallel adders. However, the carry propagation process takes on average O(log n) time to be completed [7, 8]. The carry-completion sensing adder is an asynchronous adder designed based on this fact.

3.2. Carry-Save Adder (CSA)

A carry-save adder adds three *n-bit* numbers and produces the result without performing carry propagation by saving the carry. The result will be in an *n-bit* redundant format represented using two bit vectors, sum and carry. The total delay of a carry-save adder equals to the delay of a single full adder cell. In addition, the carry save adder requires n times the area of a full adder cell. Thus, the carry-save adder takes O(1) time and O(n) space. A carry-save adder design is shown in Figure 1.

According to reference [7], there are basically two disadvantages of the carry-save adders. The carry-save adders do not add two numbers and produce a single output; instead, they add three inputs and produce two outputs such that the sum of the outputs equals to the sum of the inputs. Moreover, the sign-detection is complex. Unless the addition on the outputs is performed in full length, the correct sign of the sum-carry pair may never be determined.

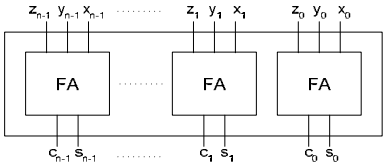

Figure 1. Carry-Save Adder

3.3. Carry-Lookahead Adder (CLA)

The carry-lookahead adder computes the sum as follows:

$$s_i = a_i \oplus b_i \oplus c_i\ ,\ p_i = a_i \oplus b_i\ ,\ g_i = a_i\, b_i\ ,\ c_{i+1} = g_i + p_i\, c_i$$

By expanding further the equation of the carry we get:

$$c_{i+1} = g_i + p_i\, g_{i-1} + p_i\ p_{i-1}\, g_{i-2} + p_i\ p_{i-1} \ldots p_1\, g_0 + p_i\ p_{i-1} \ldots p_1\ p_0\, c_0$$

A carry-lookahead adder hardware may be designed as shown in Figure 2.

The carry-lookahead logic consists of two logic levels, AND gates followed by an OR gate, for each c_i. When the adder inputs are loaded in parallel, all g_i and p_i will be generated at the same time. The carry-lookahead logic allows carry for each bit to be computed independently. Ideally, the carry signals c_i will be produced through two-stage logic at about the same time, which means that the adder will have a constant time complexity. However, it is impractical to build a two-stage full large-size carry-lookahead adder because of the practical limitations on fan-in and fan-out, irregular structure, and many long wires [7, 9].

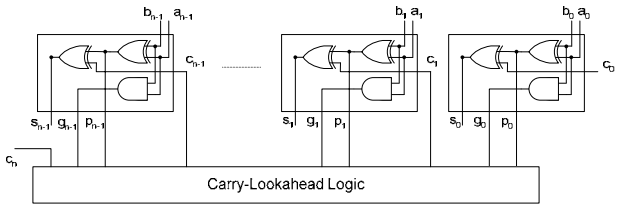

Figure 2. Carry-Lookahead Adder

In practice two approaches are used to implement the CLA: the block carry-lookahead adder and the complete carry-lookahead adder [7]. In the first implementation, small (4-bit or 8-bit) carry-lookahead logic cells with sections generate and propagate functions are built, and then they are stacked to build larger carry-lookahead adders. In complete carry-lookahead logic, the adder is built for the given operand size but in a way that allow the use of parallel prefix circuits. One well-known adder

of this type is the Brent-Kung adder [10]. The total delay of the carry-lookahead adder is O(log *n*) which can be significantly lees than the carry propagate adder. There is a penalty paid for this gain: the area increases. The carry-lookahead adders require O(*n* log *n*) area. Is seems that a carry-lookahead adder larger than 256 bits is not cost effective. Even by employing block carry-lookahead approach, a carry-lookahead adder with 1024 bits seems not feasible or cost effective. [7]

3.4. Carry-Skip Adder (CSK)

The carry-skip adder [11, 12, 13, 14] was invented for decimal arithmetic operations by Babbage in the 1800's, and become quite popular in mechanical adding machines later that century. Modern interest in carry-skip adders only began in the early 1960's by Lehman and Burla [8].

The carry-skip adder is and an improvement over the ripple-carry adder. By grouping the ripple cells together into blocks, it makes the carry signal available to the blocks further down the carry chain, earlier. The primary carry c_i coming into a black can go out of it unchanged if and only if, a_i and b_i are exclusive–or of each other. This means that corresponding bits of both operands within a block should be dissimilar. If $a_i = b_i = 1$, then the block generates a carry without waiting for the incoming carry signal. And the generated carry will be used by blocks beyond this block in the carry chain. If $a_i = b_i = 0$, then the block does not generate a carry and will absorb any carry coming into it.

By ANDing all ($a_i \oplus b_i$) of a block, the skip signal will be generated to select between the incoming carry and the generated carry using a 2×1 multiplexer as shown in Figure 3. However, reference [13] presented a more simplified skip logic that requires less area as Figure 4.

If the adder input is assumed to be loaded in parallel, then the skip signal of all blocks will be ready at about the same time. The last FA stage of a block will generate a carry, if any, before arrival of the input carry c_i. When the input carry arrives, it needs to pass through two logic gates only so that the output carry c_{i+1} will stabilize.

In order to count for the overall delay, we need to look at the longest path delay of the carry-skip adder. The longest path is the path that passes through the skip logic plus the un-skipped FA stages at the two ends of the adder as shown in Figure 5.

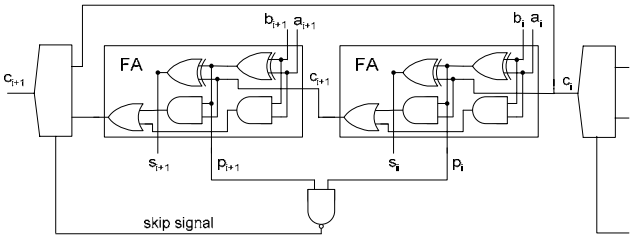

Figure 3. Using multiplexers in the carry-skip logic.

Note that the first skipped block needs to have the same size as the un-skipped block prior to it so that all the first multiplexer's inputs arrive simultaneously. Subsequent blocks can have larger size so that the carry will skip more bits and the adder speed will be increased. In this case, the adder is called one-level carry-skip adder with variable block sizes. The adder speed can be improved even more by using a multilevel skip structure; the skip logic determines whether a carry entering one block may skip the next group of blocks. However, the main design problem with the adder is working out how best to group the "skips" [12].

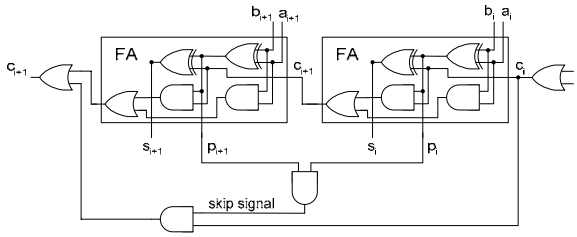

Figure 4. More simplified carry-skip logic.

In the literature, there exist many proposals for optimum design of carry-skip adders. Based on some assumptions and some input variables in addition to the desired size, the proposed algorithms decide on the optimum size of each block and some times the number of skip levels. For more details, please refer to the mentioned references at the beginning of this section.

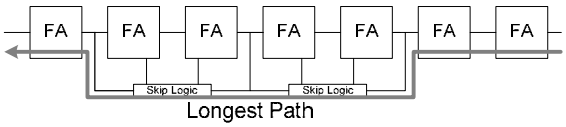

Figure 5. The longest path delay in carry-skip adders.

The carry-skip adder has a simple and regular structure that requires an area in the order of O(*n*) which is hardly larger then the area required by the ripple-carry adder. The time complexity of the carry-skip adder is bounded between $O(\sqrt{n})$ and $\Omega(\log n)$. An equal-block-size one-level carry-skip adder will have a time complexity of $O(\sqrt{n})$. However, a more optimized multi-level carry-skip adder will have less time latency reported in reference [15] to be O(log *n*).

4. COMPARISON

This section compares two fast speed adders described above: the carry-skip adder and carry-lookahead adder. By relaying on some recently published research, the two adders will be compared in terms of time, area and power.

In order to have a fair comparison, we claim that the adders need to be design at the VLSI level. Using FPGAs to implement and compare both adders will give results that are most probably inconsistent with results obtained from a practical implementation. In addition, it has been shown that optimizing the carry-skip adder is highly

dependent on the time delay deference between the skip logic and the propagate logic. Thus, optimizing the carry-skip adder on FPGAs is difficult and may not lead to an optimum time delay. This explains why implementing the carry-skip adder on FPGAs as in reference [16] results in a time delay that is not much better that the delay of the ripple-carry adder.

In reference [15], the two adders were designed using the CMOS technology and compared. A 32-bit carry-skip adder was better than a 32-bit carry-lookahead adder in terms of time, area and power. A carry-skip adder, that has multi-level skip logic, was compared with a conventional carry-lookahead adder. The carry-skip adder was 14 % faster. However, if the adder size is increased to 64 bits, the carry-lookahead adder starts to have slight improvement in time over the carry-skip adder.

The carry-skip adders have the potential for reduced power dissipation because they requires only propagate signals, in contrast with the carry-lookahead adders that require both propagate and generate signals [15]. Moreover, the carry-skip adders require a linear area that is hardly larger then the area required by the ripple-carry adder. This means much lower power consumption than the carry-lookahead adders. Reference [15] reports that the carry-skip adder's power dissipation was 58 % of that of the carry-lookahead adder.

If one-level carry-skip adder is used, as in [9], then 64-bit carry-skip adder is 38% slower than 64-bit carry-lookahead adder. However, the carry-skip adder is still better than the carry-lookahead adder in the average power consumption by 33% and in chip area by 32%.

The results presented here matches with the theoretical analysis presented before. A full-optimized carry-skip adder is comparable in speed with a conventional carry-lookahead adder since they are of the same complexity class, O(log *n*). However, the carry-skip adder is much better than the carry-lookahead adder in terms of area and power consumption.

5. CONCLUSION

This work studied the binary adders within modular multiplication hardware for crypto systems of large operand sizes. Time-area analyses of several VHDL implementations have been considered. It has been noted that carry-save adders give the maximum speedup in computing the partial products of the modula multiplications since they have constant time complexity. However, full-length addition on the sum-carry pair needs to be carried out at the last iteration, which can be assumed as a drawback. This final addition must be performed through dedicated binary adder. Two other binary adders were also considered, i.e. the carry-lookahead adder and the carry-skip adder. It has been shown that the two adders can be of a comparable speed. However, the carry-skip adders require smaller area and consume much less power than the carry-lookahead adders showing promising indications.

ACKNOWLEDGMENT

Thanks to King Fahd University of Petroleum & Minerals (KFUPM) for supporting this work.

REFERENCES

[1] Gutub, A. and Tahhan, H., "Improving Cryptographic Architectures by Adopting Efficient Adders in their Modular Multiplication Hardware", *9th Annual Gulf Internet Symposium*, Khobar, Saudi Arabia, October 13-15, 2003.

[2] Mekhallalati, M.C., Ibrahim, M.K., and Ashur, A.S., "Radix Modular Multiplication Algorithm", *Journal of Circuits and Systems, and Computers*, 6(5): 547-567, 1996.

[3] Koc, C. K. and Hung, C.Y., "Fast Algorithm For Modular Rreduction", *IEE Proceedings: Computers and Digital Techniques*, 145(4): 265-271, 1998.

[4] Kwon, Taek-Won et. al., "Two Implementation Methods of a 1024-bit RSA Cryptoprocessor Based on Modified Montgomery Algorithm", *IEEE International Symposium On Circuits and Systems (ISCAS)*, pp. 650–653, 2001.

[5] Satoh, A., and Kohji T., "A Scalable Dual-Field Elliptic Curve Cryptographic Processor", *IEEE Transactions on Computers*, 52(4): 449-460, 2003.

[6] Cheng, Fu-Chiung, Stephen Unger and Michael Theobald, "Self-Timed Carry-Lookahead Adders", *IEEE Transactions on Computers*, 49(7): 659-672, 2000.

[7] Koc, C.K., "RSA Hardware Implementation", *RSA Laboratories*, RSA Data Security, Inc. 1996.

[8] Lehman, M., and Burla, N., "Skip Technique for High Speed Carry Propagation in Binary Arithmetic Unites", *IRE Transactions on Electronic Computers*, 10: 691-698, 1961.

[9] Nagendra, C., Irwin, M., and Owens, R., "Area Time Power Tradeoffs in Parallel Adders", *IEEE Transactions on Circuits and Systems*, 43(10): 689-702, 1996.

[10] Brent, R., and Kung, H., "A Regular Layout for Parallel Adders", *IEEE Transactions on Computers*, C-31:260-264, 1982.

[11] Kantabutra, V., "Designing Optimum One-Level Carry-Skip Adders", *IEEE Transactions on Computers*, 42(6):759-764, June 1993.

[12] Burges, N., "Accelerated Carry-Skip Adders with Low Hardware Cost", *Conference on Signals, Systems and Computers*, 1: 852-856, 2001.

[13] Goel, A., and Bapat, P., "A New Time-Position Algorithm for the Modeling of Multilevel Carry Skip Adders in VHDL", *Canadian Conference on Electrical and Computer Engineering*, pp. 158-61, 1996.

[14] Turrini, S., "Optimal Group Distribution in Carry-Skip Adders", *WRL Research Report 89/2*, February 1989.

[15] Gayles, Eric S., Owens, R., and Irwin, M., "Low Power Circuit Techniques for Fast Carry Skip Adders", *Midwest Symposium On Circuits and Systems*, pp. 87-90, Aug. 1996.

[16] Xing, Shanzhen and Willlam, W. Yu, "FPGA Adders: Performance Evaluation and Optimal Design", *IEEE Design & Test OF Computers*, pp. 24-29, Jan–March 1998.