ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol.12 (2022) No.1, pp.143-157
DOI:10.33168/JSMS.2022.0111

Multiplier-based International Data Encryption Crypto-Core for Multiphase Encryption Design

Guard Kanda, Kwangki Ryoo

Hanbat National University, Daejeon, South Korea

kkryoo@gmail.com

Abstract. To ensure the continued usage of the International Data Encryption Algorithm (IDEA), current implementations rely on multiphase encryption where it is combined with other algorithms such as ROTation (ROT) and Data Encryption Standard (DES) for maximum security strength. Multiphase encryption implies that there is a tendency for an increase in hardware area and a reduction in overall speed. However, high-speed and reduced area algorithms are much desired. This paper, therefore, proposes an efficient hardware implementation of the IDEA cipher that is based on arithmetic modulo multiplication—one of the main computations of the IDEA—on a novel Vedic multiplier architecture. The increase in efficiency of the IDEA crypto architecture and the reduction in resources utilization is achieved through an enhancement of its structural architecture to utilize a fixed set of resources for all eight (8) identical rounds of computation and the use of a proposed fast and lightweight Vedic hardware multiplier. The proposed hardware modification and resulting architecture are designed using the Xilinx ISE and Vivado tools. The architecture is synthesized using Precision Synthesis Tool (PS) and simulated using Modelsim SE 10.6d and ISIM simulation tools. The proposed IDEA cipher is 100% more efficient when designed based on the Vedic multiplier compared to existing designs. The hardware architecture is implemented on Spartan-6-FGG484 Field Programmable Gate Array (FPGA) using Verilog HDL. Verified results show that the proposed Vedic-based IDEA occupied 212 Slices with the Vedic multiplier only occupying 28 Slices out of the total 212. The proposed architecture operates at a maximum frequency of 253.3 MHz.

Keywords: IDEA, Vedic Mathematics, Xilinx, FPGA, Hardware Multiplier.

1. Introduction

To aid in the transfer of information between parties such that only the intended recipient could access it, the branch of mathematics known as cryptography came into existence. An unintended recipient who may have complete knowledge of the mathematical algorithm that was utilized in encrypting a piece of information or data and the medium through which the information is transferred and not being able to undo the encryption process is the goal of the cryptography field of study. A popular secret key cryptographic algorithm back in the days of 1976 was the Data Encryption Standard, known as DES for short. This crypto algorithm was widely used in securing an array of financial and commercial applications because it had then proved to be resistant to the available cryptanalysis. That notwithstanding, it had a short key length of only 56-bits.

In the year 1990, the IDEA algorithm which is a symmetric block cipher was proposed by two cryptographers X. Lai and J. L. Massey. It initially was referred to as the Proposed Encryption Standard (PES) by the same authors (Lai, Massey, 1990). The name again later evolved to Improved Proposed Encryption Standard (IPES) (Lai et al., 1991) a year later when cryptanalysts Giham and Shamir provided a demonstration of differential cryptanalysis. Due to the shortness of the key size of the Data Encryption Standard (DES), it was estimated that its entire 56-bit long keyspace could be searched within a space of about 22-hours (DES Challenge III, 1999). The IDEA was originally developed and proposed as a replacement to the DES algorithm (NESSIE, 2004). This secret key encryption block cipher algorithm encrypts or decrypts a 64-bit long plaintext or ciphertext respectively, using a 128-bit wide key. This algorithm is highly secured due to the non-usage of substitution boxes and lookup tables that are usually found in block ciphers. Although the IDEA cipher was broken during 2011 based on the meet-in-the-middle attack (Biham et al., 2015) and again in the year 2012 by the narrow-bicliques attack (Khovratovich et al., 2012), the security of the IDEA is however practically not threatened. The structure of the IDEA algorithm is shown in Fig. 1. The IDEA algorithm is based on three main computations: Multiplication modulo $2^{16} + 1$ ($\odot$), Addition modulo 2^{16} ($\boxplus$), and eXclusive-OR ($\oplus$). Although IDEA did not replace the DES as intended, it was eventually incorporated into the Pretty Good Privacy (PGP)—an application software program that offers cryptographic privacy and authentication for data communication such as emails.

The IDEA crypto algorithm operates by taking a 64-bit input block of data and a 128-bit key. The 64-bit long data is broken into 4 blocks each of length 16-bits. This is shown as D1-D4 in Fig. 1. Each of the 4 blocks is passed through 8 identical rounds of arithmetic and logical operations shown in Fig. 1. After the round operations, the resulting data which is represented as **C1'** to **C4'** is again passed through a final phase known as the output transformation round. This phase involves only arithmetic operations. During each of the 8.5 rounds, 6-sub keys are

generated in each round as can be seen represented as **K1** to **K6** in Fig.1.

Vedic mathematics is a primitive technique that was utilized in the era of Vedas. The term 'Vedas' generally refers to the repository of all knowledge from which Vedic mathematics was reconstructed. This technique comprises a maximum of 16 rules used for varying arithmetic calculations and can be applied to and covers almost every branch of Mathematics (Kent, Sharma, 2015) Urdhva Tiriyagbhyam—vertical and crosswise—acts as the basic and most generalized rule for the quick and simple Vedic multiplication implementation (Sona, Somasundram, 2020). This general and basic rule is the vertical and crosswise technique.

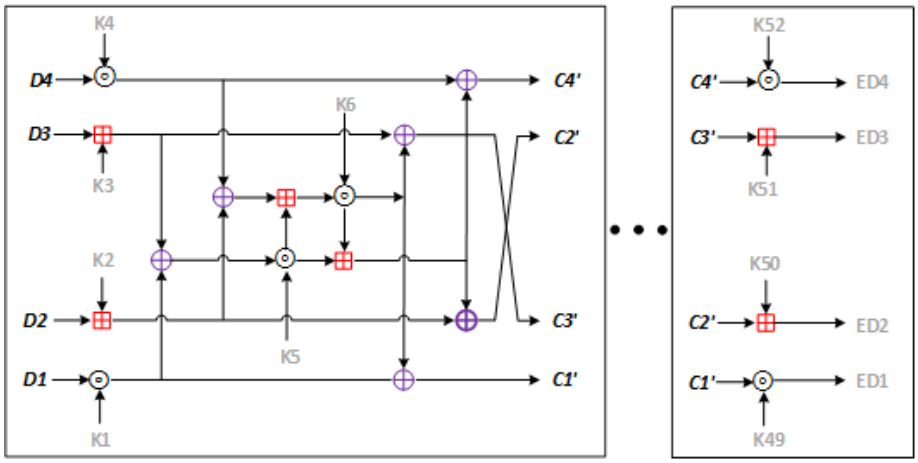

Fig. 1: Round and Output Transformation of an IDEA Algorithm.

2. Related Literature and Background Study

Several implementations of the IDEA crypto algorithm have been demonstrated both in literature and in practice. In the year 2000, (Leong et al., 2000) presented a novel bit-serial architecture that performed the $2^{16}+1$ modulo multiplication. This architecture meant that the data was processed in a bitwise fashion, either starting with the least significant bit or most significant bit and the intercommunication between operators was executed by way is multiplexors over a single bit wire. In their design the primitive operators—XOR and Addition modulo $2^{16}+1$ had a latency of 1 cycle whereas the modulo multiplication had a latency of 35 cycles due to the bit-serial nature of the design. The authors also doubled the throughput of the bit-serial by duplicating the logic of Lyon's serial-parallel multiplier. This allowed for a two-stage pipeline of the algorithm whereby two consecutive multiplication results. However, this approach meant that the design required several latches which causes a lot of timing issues and also increased the area to an extent although the resulting architecture ensured that the proposed design occupied a minimal hardware area. The deeply pipelined architecture proposed allowed the design to operate at a speed of 125MHz on the Xilinx VirtexXCV300-6 FPGA.

Not all, (Modugu et al., 2009), presented a novel multiplexor-based compressor, and modulo carry-lookahead adder were used as the building blocks to implement 2^n+1 multiplier design. In that research, it was asserted that CMOS implementation of a MUX had better performance in terms of power and delay compared to an XOR. Using the output of the multiplexor and its complement resulted in compressors with efficient performance. From this research, it was observed that the use of a carry-save adder improved the propagation delay of the multiplier and such a technique was adopted in this research. The same authors (Modugu et al., 2010), again presented a similar approach by using compressors to implement the $2^{16}+1$ modulo multiplier along with sparse tree-based carry adders with inverted ends. Furthermore, the authors (Ledda et al., 2019) proposed the combined use of the middle square method to generate Pseudo-Random Numbers (PRN) that are used during the key generation phase. During the encryption phase, the original key is XORed with the plaintext before it is divided into two halves of 64-bits each. After each parallel round of computation, the texts are combined and a cyclic left shift of 25 bits is performed. This improves the randomness and diffusion by focusing on enhancement to the number of rounds, the shift operations, bit augmentation, and adjustment during the algorithm's operation.

Penumetcha et al, in their research (Penumetcha et al., 2015), presented two design approaches—RTL looping and pipelining—of the crypto core implementation. The first approach which is the RTL looping technique used the same set of resources from the first round of computation multiple times through multiplexors from control logic (unit). Through this approach, the latency for each round is 10 cycles with an overall latency of 86 cycles. For the pipeline approach, a pipeline of depth 9 stages where the results from one computation are fed to the next independent blocks sage. This implied that a total of 9stages x 9 operations (81 cycles) were required. The two results presented by (Penumetcha et al., 2015), had throughputs of 764.59 Mbps and 73.45 Mbps for the pipeline and RTL loop respectively. It was observed that although most of the aforementioned designs obtained moderate to high throughput, they were not true efficient when measured to the area cost. Based on this, the architecture of the proposed Vedic multiplier-based IDEA crypto algorithm is designed to improve efficiency.

3. Proposed Architecture

3.1. Proposed Modification to the IDEA Cipher

The design rationale behind the IDEA cipher is to have a blend of mathematical operations that work on the 16-bit sub-blocks of the input data to enhance diffusion which relates to how a small variation in the key will lead to a large variation in the ciphertext and confusion which relates to how a small change in the input message will result in a huge variation in the ciphertext. The IDEA crypto algorithm is composed of 8-rounds of computation with a final round known as the half-round

that performs output transformation of the resulting data is shown in Fig.1 and consists of two modulo multiplication and eXclusive-OR operations. Each round of the computation is composed of a 16-bit block bitwise XOR module, a 16-bit modulo 2^{16} addition module and $2^{16}+1$ modulo multiplication of 16-bit values. This shows that the computation generally requires a 16-bit word size or Datapath.

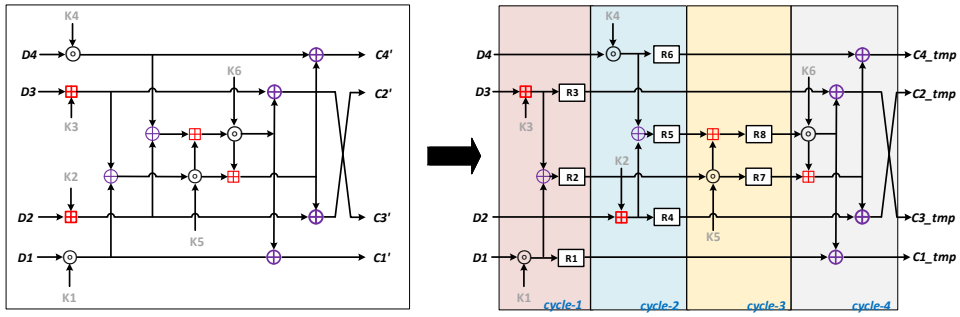

Fig. 2: (a) Original IDEA Round (b) Proposed Resource-Sharing IDEA Round

In the transformation of output stage also, two of the previous operations—modulo addition and multiplication are performed. Required by the IDEA algorithm for its computation is the generation of partial keys. A total of 52 sets of partial keys, each of length 16-bit and which is derived from an initial key of length 128-bits is performed during encryption or decryption. As shown in Table 1, the partial keys are obtained by performing a 25-bit cyclic shift of the key to the right in each round. The partial key $\boldsymbol{K_{i\_j}}$ where "i" represents the cyclic shift—one (1) for the initial key and "j" represents the 16-bit part selection of the resultant key, 1 for bits 127 to 112 and 8 for bits locations 15 to 0. Similarly, for decryption, the partials keys that are utilized are the multiplicative and additive inverses of the corresponding resulting keys after the shift in reverse order. For this research, the main aim is to enhance the architecture to obtain the highest possible throughput for the IDEA cipher in an FPGA. To achieve this, the design was drastically repartition. With the ordinary implementation of the algorithm, a total of 34 modulo additions and 34 modulo multiplications are required for a standard 8-round and a half-round IDEA algorithm. Hardware multipliers are very critical in hardware design as they directly impact the operation speed, the area occupied and power consumption of any hardware system they are found. This implies that a serial implementation of the IDEA core would mean that the hardware area for the crypto core will be exponentially high—having 34 different multipliers.

Table 1: IDEA Encryption-Round Subkey Generation

Round	Partial Keys Generation	Key ID.

1	K1_1, K1_2, K1_3, K1_4, K1_5, K1_6	K1~ K6
2	K1_7, K1_8, K2_1, K2_2, K2_3, K2_4	K7~K12
3	K2_5, K2_6, K2_7, K2_8, K3_1, K3_2	K13~K18
4	K3_3, K3_4, K3_5, K3_6, K3_7, K3_8	K19~K24
5	K4_1, K4_2, K4_3, K4_4, K4_5, K4_6	K25~0
6	K4_7, K4_8, K5_1, K5_2, K5_3, K5_4	K31~K36
7	K5_5, K5_6, K5_7, K5_8, K6_1, K6_2	K37~K42
8	K6_3, K6_4, K6_5, K6_6, K6_7, K6_8	K43~K48
9	K7_1, K7_2, K7_3, K7_4	K49~K52

The first approach to increasing the speed of the IDEA core was to perform a pipeline approach. Since all the subsequent seven (7) rounds possess the same order and structure of computation, a single module round that is repeated for all the eight (8) rounds was designed, registering the results from each complete round and feeding it back into the single module. As seen in Fig. 2(b), four registers (D-Flipflop) of size 16-bits each were used to hold the intermediate results labeled ***C1'***, ***C2'***, ***C3'***, ***C4'*** of a single round computation. This approach drastically reduced the amount of area required for the implementation of the IDEA crypto core.

Although the pipeline design approach in the paragraph above reduced the hardware area and increase the speed of execution, it, however, did not attain the best performance since there was the requirement for a large number of hardware resources. A typical implementation of the IDEA crypto-core will be integrated with other cores such as UART, on-chip bus, DMA, and Synthesizable processors. Therefore, an implementation that requires minimal resources is much desired. To further increase the performance, the proposed modification to the IDEA crypto algorithm shown in Fig. 2(b) was presented.

The proposed modification introduced additional registers—eight of them—to the original algorithm shown in Fig. 2(a) by tweaking and rescheduling s single around into four independent sections. Each section—labeled cycle-1 to cycle-4—was defined after careful examination and breaking down or regrouping of the mathematical computations into smaller blocks that allowed the crypto core multiple utilization of the computationally intensive and hardware constrained resources—a design approach that is known as resource sharing. Fig. 2(b) further shows that the proposed reordering was performed such that each round of computation will be performed in four sequential computations, each lasting for a period of one cycle. The Datapath of the proposed architecture was therefore designed such that a single addition modulo 2^{16} (⊞) and multiplication modulo $2^{16}+1$ (⊙) computations were utilized in addition to the eight (8) temporal registers

and multiplexers.

3.2. Proposed 16x16-bit Vedic Multiplier

Aside from addition or multiplication, research has shown that multiplication is the next most utilized mathematical operation (Wilmshurst, 2010) that is required in computer arithmetic. Due to its computational complexity, several hardware processors do have dedicated multipliers and hence multiply instructions for that matter. In its simplest form, the general approach to the execution of a multiplication operation is the classic elementary school repeated shift and add method. Since hardware multipliers can dictate and determine the performance and throughput of the applications they are found in, the use of an efficient multiplier architecture is highly desired when designing any multiplier required hardware architecture. To again increase the performance of the IDEA crypto core, the Vedic multiplier was designed and implemented in the crypto core.

As depicted in Fig. 3, a decimal computation of the numbers ***123x456*** is demonstrated based on the Vedic rules. Beginning with STEP 1, the rightmost digits of the multiplicand and the multiplier are vertically multiplied—***3x6***—which results in 18. Since this is the starting point, there is no carry from any previous step and so carry is 0. The carry and the result summed together. The rightmost digit is used as the first partial product and the leftmost digit—which is 1—is kept as a "carry" for the next step—STEP 2. In STEP 2, we move crosswise and perform computation where the rightmost two digits for the multiplicand—2 and 3—and that of the multiplier—5 and 6—are multiplied and summed in a crosswise approach. That is the result of (***6x2) + (5x3)*** being added to the "carry" from STEP 1 to obtain 28. Similarly, the next digit of the partial product is the rightmost digit of the result—which is 8—which is placed to the left of the previous partial product. The leftmost digit of the result—which is **2**—becomes the "carry" for the next step. STEP 3 uses the next 3 right-most digits, ***123*** for the multiplicand and ***456*** for the multiplier, and makes a vertical and crosswise multiplication. This multiplication becomes [***(1x6) + (5x2) + (4x3)***] which is subsequently added to the "carry" from the previous step to get a result of **30** where the **0** is added to the leftmost side of the partial product and the **3** used as the "carry" for the next step. At the end of STEP 5, the final partial product can be built giving the final product of **56088**. From Fig. 3, the algorithm can be generalized to any ***NxN*** decimal number. A maximum number of ***2n-1*** partial products are required to generate the final sum.

For a 3x3 digit number, a maximum total of 5 partial products represented as steps are required to obtain the resulting product. Fig. 4 illustrates the same procedure but in the binary domain which shows the computation of ***2x3*** ≡ ***(10x11 = 110)***. A 2-bit by 2-bit number multiplication requires six (6) AND gates and two (2) XOR gates. In designing the 16x16 bit Vedic multiplier, we implemented the bottom-up approach. The 2x2 bit was used in designing a 4x4-bit Vedic multiplier.

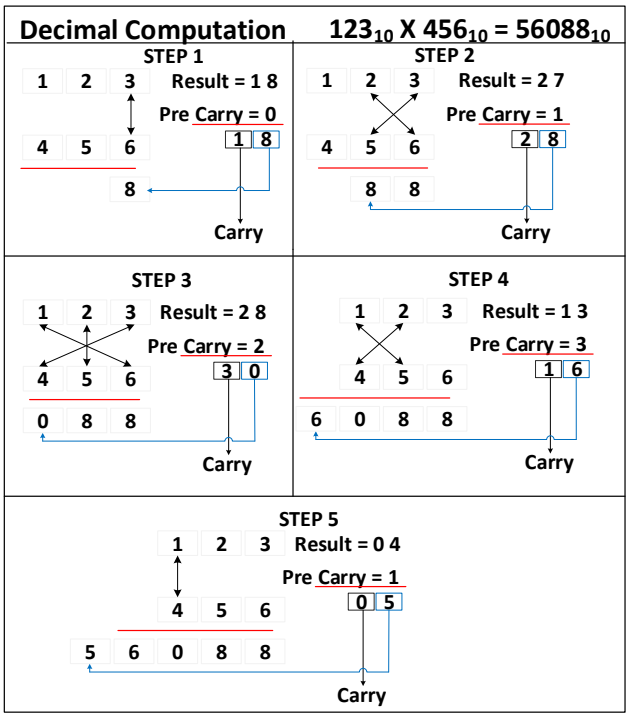

Fig. 3: Vedic Multiplication Steps in Decimal Domain

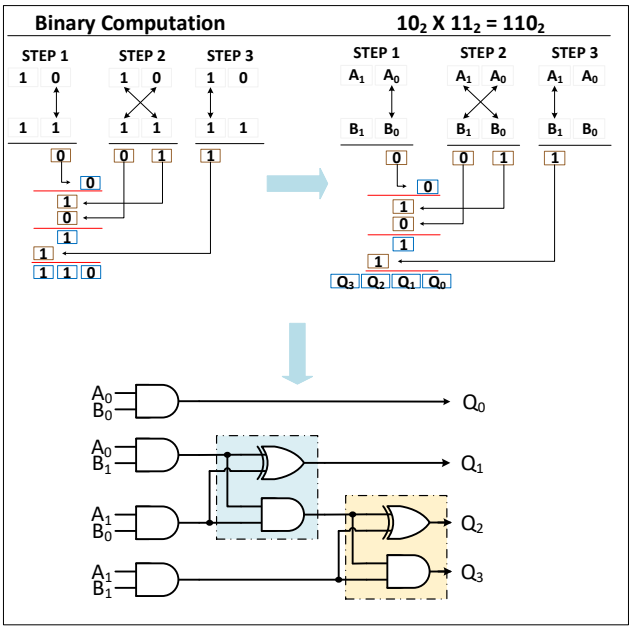

Fig. 4: Vedic Multiplication Steps in Binary Domain with Logic Schematic

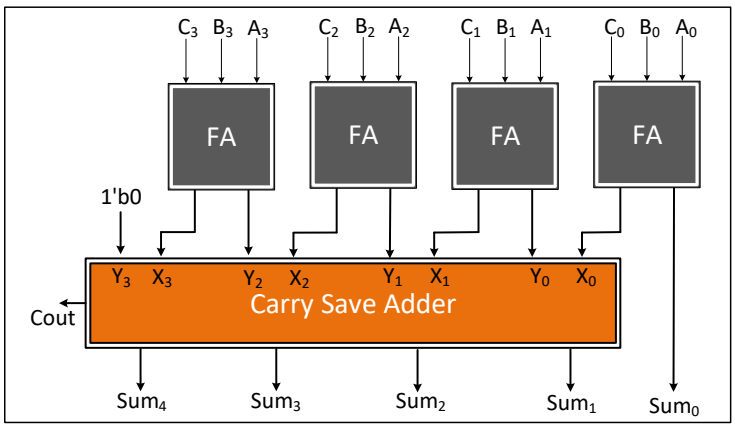

Fig. 5: Vedic Multiplication Steps in Decimal Domain

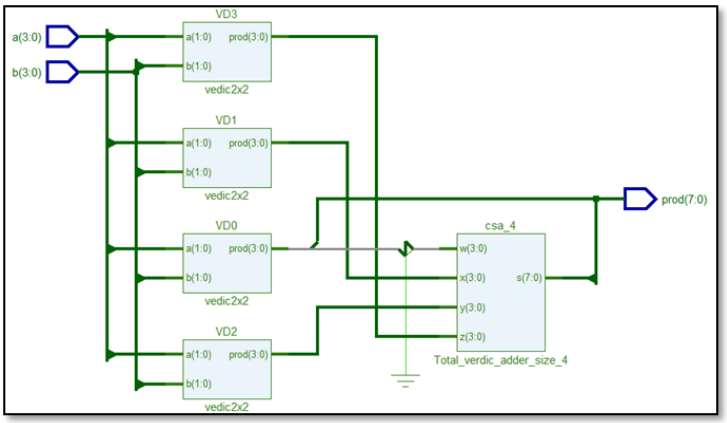

Fig. 6: Vedic Multiplication Steps in Decimal Domain

As shown in Fig. 6, a 4x4-bit Vedic multiplier was implemented using four (4) of the 2x2 multipliers and 2 4-bit Carry-Save Adders (CSA). The CSA architecture in Fig. 5 was parametrized module to aid in various bit length implementation with a single module. The CAS was chosen over other adders for reasons that, its carry is not propagated through the design but stored and added in the final stage of the addition. This reduces the delay or increases the speed of the multiplier architecture. Four (4) of the 4x4 Vedic multiplier and two (2) 8-bit adders are combined to design an 8x8 Vedic adder shown in Fig. 7. In like manner, Fig. 8 is an architecture depicting four (4) of the 8x8 Vedic multipliers and two (2) 16-bit carry-save adders are utilized in designing the 16x16-bit Vedic multiplier. Using a parameterized design approach, the initial base design of 2x2 Vedic multiplier and 4-bit carry-save adders were instantiated appropriately to result in any ***NxN*** Vedic multiplier.

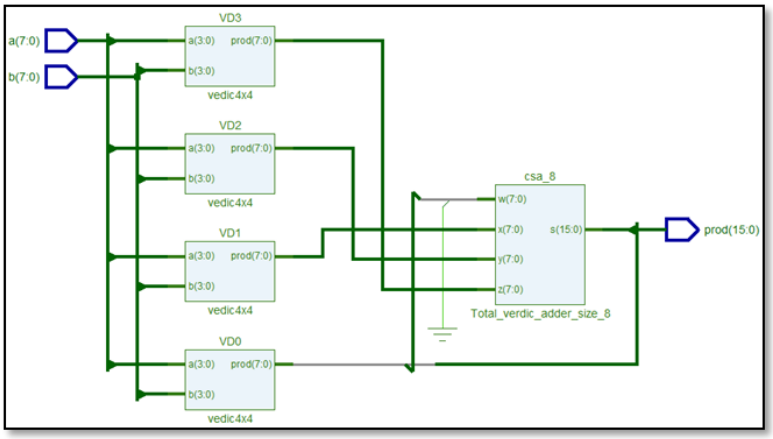

Fig. 7: Block Diagram of Proposed 8x8 Vedic Multiplier

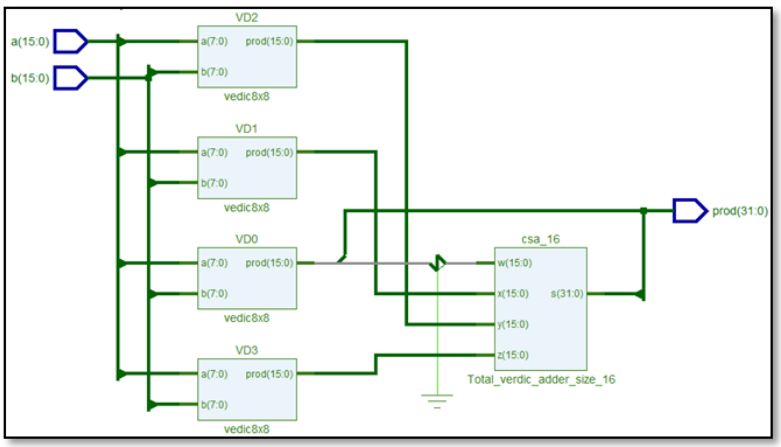

Fig. 8: Block Diagram of Proposed 16x16 Vedic Multiplier

Fig. 9 is a timing simulation of the proposed and implemented 16x16-bit Vedic multiplier. The Simulation shows the 16-bit CSA with four 16-bit inputs ***W***, ***X***, ***Y***, and ***Z*** which are the partial products obtained during the Vedic multiplication. The output of the addition becomes the output of the whole multiplication. Also shown in the simulation are the four 8x8-bit Vedic multipliers utilized in the 16x16-bit Vedic multiplier alongside their partial products which are fed into the CSAs.

Two 16-bit unsigned random numbers were generated in the testbench. These two numbers were passed through the Proposed Vedic multiplier. Also, the generated random numbers are multiplied using the multiplication operator in Verilog. The two results are compared to check equality. Simulating all the possible combinations of multiplicand—a and multiplier—b between the range ***16'h0001*** and ***16'hFFFF*** for both values resulted in a 100% accuracy of the resulting products.

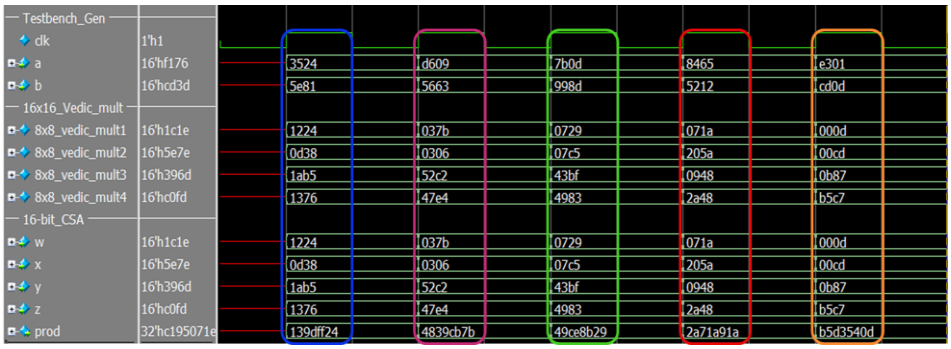

Fig. 9: Timing Simulation of the Proposed 16x16 Vedic Multiplier

4. Experimental Hardware Test Results and Discussion

The proposed hardware-friendly approach of the IDEA algorithm was implemented with the proposed modifications in the previous section. The proposed modified architecture had three main modules namely the ***IDEA_Controller***, the ***Key_Scheduler***, and the ***IDEA_Round_Datapath*** modules. Also, four registers of size 16-bit and labeled ***REG_D1***, ***REG_D2***, ***REG_D3***, and ***REG_D4*** and four 2-to-1 multiplexers were used to obtain the pipeline architecture that allowed the resource sharing architecture to be realized. The ***IDEA_Round_Datapath*** is made up of eight (8) 16-bit wide registers that allowed a single modular multiplication based on the Vedic multiplier, a single modular addition, five eXclusive-OR computations, and four (4) 4-to-1 multiplexers. The ***Key_Scheduler*** generates the keys for each round. It takes as input the original key and the round number. As seen in Fig. 10, six (6) subkeys of size 16-bits each, labeled **K1** to **K6** are generated for each round. The final submodule that was designed was the ***IDEA_Controller***. This module generates the round count, the start, mux select signal to control all the other modules in the proposed crypto core. A single round, therefore, requires a minimum of 5 clock cycles to complete one round—that is four (4) for the round and one for the temporal data turn around. For the eight required rounds, a total of 40 clock cycles is required. Additionally, an extra two (2) clock cycle is required for the final half-round which is for the output transformation, resulting in a total of 42 clock cycles for this architecture. It must be noted that a minimum of about 34 clock cycles can be utilized in the completion of an eight and a half-round cipher core. After the computation, the cipher data is output through the registers labeled **ED1**, **ED2**, **ED3**, and **ED4**. The proposed architecture was implemented using Verilog HDL on Xilinx ISE version 14.7. The design was synthesized using Mentor Graphics' Precision RTL synthesis (PS) tool and simulated with the ModelSim Standard Edition version 10.6d. Fig. 11 shows the simulation of the proposed IDEA core based on the Vedic Multiplier for the modulo multiplication. Table 2 lists sample test vectors for the IDEA core. These vectors include the input data, the encryption key, and the corresponding or resulting cipher data. The values from the

data and key columns of the first row in Table 2 are set as input to the simulation shown in Fig. 11.

The convenience of calculations, the anonymous nature of the participants, and the absence of a state monopoly on the issue made the cryptocurrency a rather expensive asset in the property of the owner. Unlike a generally accepted e-wallet, which requires an actual transfer of money to a certain currency (for example, through a bank), cryptocurrency is focused on the vast Internet network and is in no way connected to one of the existing conventional currencies.

Table 2: Sample Test Vectors from IDEA Core

Input Data	Key	Cipher
64'h0000000000000000	*128'h1F2F3F0F3000195702552020000A1988*	*64'h230CF227D574B5D2*
64'h0000000000000000	*128'h1F2F3F0F3000195702552020000A1984*	*64'hC7E98EC16C93581*
64'h0123456789ABCDEF	*128'h0F2F3F0F3000195702552020000A1988*	*64'hB30512ACDD5EBAC7*
64'hDEADBEEFFEEDDEED	*128'h00000000000000000000000000000001*	*64'h576E7FC507C9250C*

$$\textit{Throughput} = [\textit{(Frequency x Number of bits)}] / [\textit{(Number of Clock Cycles)}] \quad (1)$$

$$\textit{Efficiency} = [\textit{Throughput (Mbps)}] / [\text{Area } \textit{(Slices)}] \quad (2)$$

The design was tested on the COMBO-II DLD board from Hanback Electronics which is fitted with the XC6SLX45 Spartan-6 FPGA chip. The board was operated at a clock frequency of 50MHz. The performance of the proposed design was measured and compared to that of some existing designs. Equations (1) and (2) were used in computing and determining the throughput and efficiency of the proposed design. Processing 64-bits of data during a clock cycle of 42 will yield a throughput of 385.5 megabits per second. Although the throughput is reduced compared to the architectures in (Ledda et al., 2019) and (Penumetcha et al., 2015), the overall efficiency is better compared to the design in (Ledda et al., 2019) because the area occupied by the proposed design is smaller and is associated with the Vedic modular multiplier designed. This makes the proposed design about 800% more efficient than (Ledda et al., 2019) and over 180% more efficient than the efficient design of (Penumetcha et al., 2015). Additionally, the 16x16-bit Vedic multiplier occupied only 28 Slices out of the 212 for the entire encryption core. Table 3 summarizes the hardware results and performance in comparison with other existing designs.

5. Conclusion

This paper proposes a modification of the IDEA algorithm for its usage in hardware. The proposed modification results in a reduction in the area required to implement the IDEA algorithm.

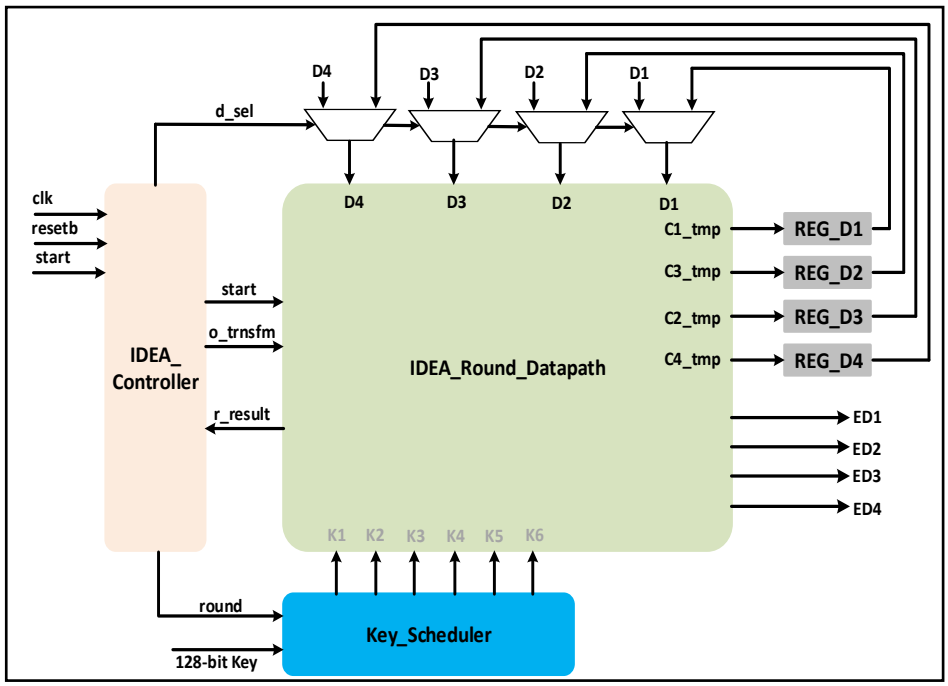

Fig. 10: Block Diagram of the Proposed Resource-Sharing IDEA Architecture

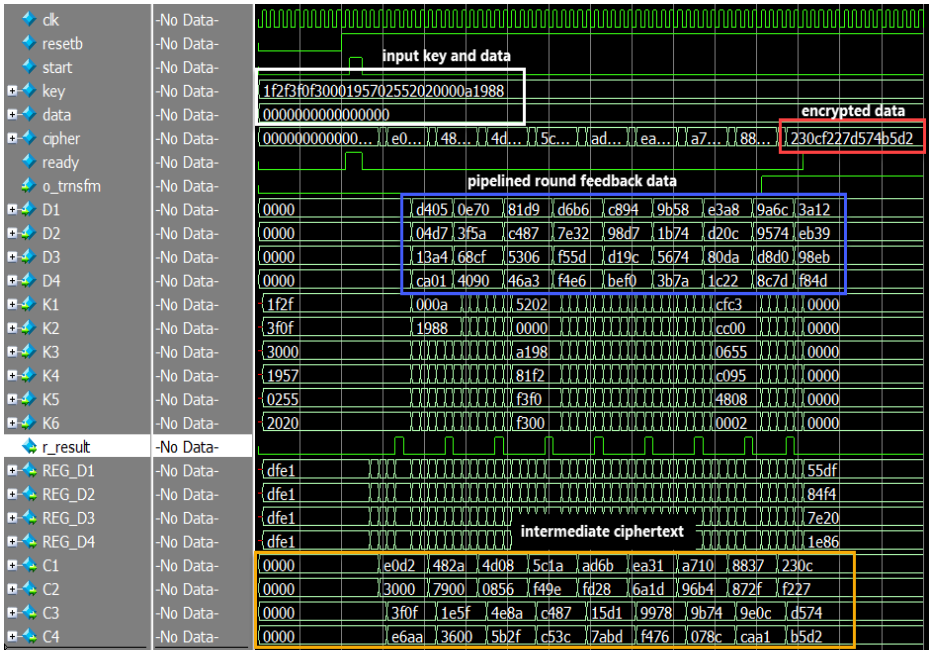

Fig. 11: Timing Simulation of the Proposed Efficient IDEA Algorithm

This is attained by carefully sectioning the algorithm into four main cycles

where a portion of the algorithm is executed in each cycle and the temporal results are held in registers. This approach allowed the use of only 8 registers which accounted for the reduced hardware area.

The design also operates at 253.3 MHz clock frequency and had an efficiency or performance of 1.81 making the proposed design 800% more efficient compared to (Ledda et al., 2019) and 180% more efficient than the pipeline implementation of (Penumetcha et al., 2015). Although the IDEA algorithm is not widely used as other algorithms such as AES, it is still part of the Pretty Good Privacy email encrypting protocol (Open PGP, 2016). Having gains and improvement in efficiency, the proposed architecture enables an efficient multiphase cryptosystem design. Not all, the proposed architecture also serves as an excellent crypto algorithm for teaching and learning. In the next stage of this research, the decryption core will be implemented to create a complete and unified IDEA crypto core. This is necessary because the decryption process requires multiplicative inverse operations which are computationally intensive to implement on hardware.

References

Barbarosoglu, G. & Pinhas, D. (1995). Capital rationing in the public sector using the analytic hierarchy process. *The Engineering Economist*, 40, 315-341.

Lai, X. Massay, J. (1999). A proposal for a new block encryption standard. *Advances in Cryptology, Proceedings of Eurocrypt*, 389-404.

Lai, X. Massay, J. Murphy, S. (1991). Markov ciphers and differential cryptanalysis. *In Advances in Cryptology, Proceedings of Eurocrypt*, 17-38.

DES Challenge III broken in record 22 hours (1999). WEB*: http://cs-exhibitions.uni-klu.ac.at/index.php?id=263.*

NESSIE Submissions (2004). Encryption-the IDEA Algorithm. WEB*: https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions.html*

Biham, E., Dunkelman, O., Keller, N., Shamir, A. (2015). New Attacks on IDEA with at Least 6 Rounds. *Journal of Cryptology*, 28(2), 209-239.

Khovratovich, D., Leurent, G., Rechberger, C. (2012). Narrow-Bicliques: Cryptanalysis of Full IDEA. Advances in Cryptology – *EUROCRYPT 2012, Lecture Notes in Computer Science*, 7237, 392-410.

Kant, A., Sharma, S. (2015). Application of Vedic multiplier designs – A review. *4th International Conference on Reliability, Infocom Technologies and Optimizations (ICRITO)*, 1-6.

Sona, M.K., Somasundram, V. (2020). Vedic Multiplier Implementation in VLSI. *Materials Today: Proceedings*, 24(4), 2219-2230

Abdullah, D., Rahim, R., Andysah, P.U.S., Ananda, FU., Zahratul, F., Malahayati, M., Harun, H. (2018) Super-Encryption Cryptography with IDEA and WAKE Algorithm. *Journal of Physics*, 1019, 208-211.

Hassan, M.S., Hamza, G.G. (2021). Real-time FPGA implementation of concatenated AES and IDEA cryptography system. *Indonesian Journal of Electrical Engineering and Computer Science,* 22(1), 71-82

Leong, M.P., Cheung, O.Y.H., Tsoi, K.H., Leong, P.H.W. (2000). A bit-serial implementation of the international data encryption algorithm IDEA, *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, 122-131.

Modugu, R., Park, N., Choi, M. (2009). A Fast Low-Power Modulo 2n+1 Multiplier Design. *IEEE International Instrumentation and Measurement Technology Conference*, 951-956.

Modugu, R., Kim, Y., Choi, M. (2010). Design and performance measurement of efficient IDEA (International Data Encryption Algorithm) crypto-hardware using novel modular arithmetic components. *2010 IEEE Instrumentation & Measurement Technology Conference Proceedings*, 1222-1227.

Ledda, M.K.C., Gerardo, B.D., Hernandez, A. A. (2019). Enhancing IDEA Algorithm using Circular Shift and Middle Square Method. *17th International Conference on ICT and Knowledge Engineering (ICT&KE)*, 1-6.

Penumetcha, D.V., Jiafeng, X., Saiyu, R. (2015). FPGA design space exploration of IDEA cryptography IP core. *IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1-4.

Wilmshurst, T. (2010). CHAPTER 11 - Data acquisition and manipulation. *Designing Embedded Systems with PIC Microcontrollers (Second Edition)*, *Newness*, 339-369.

Open Pretty Good Privacy. (2016). WEB*: https://www.openpgp.org/about/history/*