DESIGN FEASIBILITY STUDY
FOR A 500 GBITS/S AES CYPHER DECYPHER ENGINE

Abdelhafid Bouhraoua
Computer Engineering Department
King Fahd University of Petroleum and Minerals
P.O.Box 969, Dhahran, 31261
Saudi Arabia
Email: abouh@ccse.kfupm.edu.sa

***Abstract* — A feasibility study for implementing the AES encryption algorithm in hardware achieving 500 Gbits/s is presented. The methodology followed in the process of obtaining the solution allowed us to reach a highly regular solution that is scalable.**

***Index Terms* — AES, High Throughput, ASICs, High Speed Architectures**

DESIGN FEASIBILITY STUDY
FOR A 500 GBITS/S AES CYPHER DECYPHER ENGINE

1. INTRODUCTION

In recent years the internet has become one of the top communication medium used by the general public. More and more services are available through the internet. Managing sensitive information and the need for security has become a major concern for the users as well as the providers. Global security threats, cyber attacks to cripple a network connection or unauthorized intrusions to access restricted information are nowadays network security concerns all over the world. Encryption is a mean by which information can be safely exchanged.

Symmetric encryption is used to exchange high bandwidth sensitive data between users. In October 2000, the American National Institute for Standards and Technology (NIST) announced that the Rijndael algorithm was selected to become the Advanced Encryption Standard (AES) to replace the old Data Encryption Standard (DES).

High speed exchange of secure data is becoming a trend. Large volume data servers and high capacity network routers are starting to implement encryption as part of their portfolio in order to address the need for security. The aim of this work is the demonstration of the feasibility of building a 500 Gbits/s AES encryption decryption engine.

Section 2 briefly describe the AES algorithm and mentions the previous work. The proposed approach, followed in this work is presented in section 3. Section 4 describes the architecture of the building block of the AES implementation proposed in this paper. Section 5 proposes the architecture of the key schedule generation block. Section 6 shows how the 500 Gbits/s is achieved using the building block described in section 4. The interface building and the task scheduling are discussed respectively in sections 7 and 8. This section is followed by conclusions in Section 9.

2. AES ALGORITHM AND PREVIOUS WORK

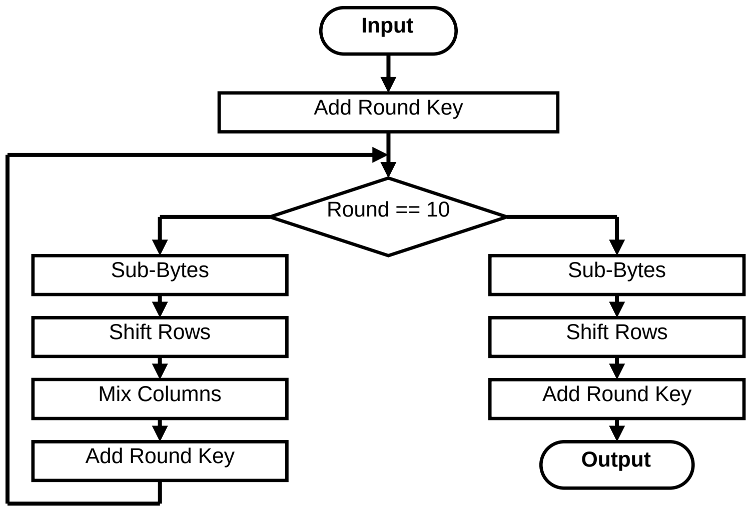

Figure 1 – AES Algorithm

The AES algorithm [1] processes data blocks of 128 bits. It can support a key size of 128, 192 and 256 bits. In our work, only 128 bits keys are considered. Each data block consists of 16 bytes organized as a two dimensional array of 4 x 4 bytes called the *State*. The basic operations of the AES algorithm are performed on the State. After an initial round key bitwise addition (XOR), a round function consisting of four different transformations (sub-bytes, shift-rows, mix-columns, and add-round-key) is applied to the data block in the encryption procedure as shown in Figure 1 below. The round function is iterated 10 times for a key length of 128 bits. The sub-bytes operation is a nonlinear byte substitution that operates independently on each byte of the state using a substitution table (S-Box). The shift-rows and mix-columns operations are circular shifts on the rows and respectively columns of the state with different numbers of bytes (offsets). The mix-columns operation adds to the shifts a multiplication with a fixed polynomial modulo $x^4 + 1$. The add-round-key operation is a XOR that adds a round key to the state. The round keys are a set of 11 32-bits words generated during the key expansion phase. Their generation depends solely on the key and does not depend on the data.

Numerous proposals have addressed the high speed hardware implementation of the AES algorithm. Some of the proposals have focused on an ASIC implementation [2,3,4,8] while others have targeted FPGAs [5,9,11,12,13]. Many techniques have been used to implement the AES algorithm in hardware. The lookup table based approach, the pipelined approach, the loop-unrolling and the sub-pipeline approaches are some of these techniques. Lookup table based designs have been explored in [9]. The idea of the lookup table implementation is to preprocess all the values that can possibly be encountered in a byte transformation and put the values in several lookup tables. The main advantage is to avoid implementing complex operations in hardware. The idea is very attractive for low to medium speed FPGA implementation but is inefficient in the context of ASICs where lookup table implementations are generally slower than regular logic gates. Other FPGA implementations have chosen a memory-less approach where no lookup tables are used [11]. The pipeline approach increases the throughput by processing multiple blocks of data simultaneously. It is realized by inserting registers between blocks of combinational logic representing a single round or a single operation [18]. Several implementations have adopted the pipelined approach [2,11,20]. Sub-pipelining [3,18] is similar to pipelining. The main difference between sub-pipelining and pipelining is the division of a single round or a single operation, which is a single pipeline stage, into several sub-operations or sub-pipeline stages reducing the inter-stage gate delays and increasing the operating frequency of the sub-pipeline. Loop unrolling [5,7] is the opposite of pipelining where several operations and even rounds are sequentially processed using combinational logic within a single clock cycle.

3. PROPOSED APPROACH

The approaches cited above, except for the lookup table based approach, have produced optimized solutions. Often, these solutions are complex solutions in terms of flexibility of implementation, heavily relying on the technology. Most importantly, these solutions are not always scalable. Most of the proposals reach performance levels around few Gbits/s[8] to few tens of Gbits/s[4,11,20]. The lookup table approach relies on the intensive use of memories and therefore cannot be considered in our context. The loop unrolling approach leads to large combinational logic areas inducing a relatively low frequency, contradictory with the objective of achieving high throughput.

In order to achieve our goal of efficiently building a 500 Gbits/s throughput AES engine, we need to fix some design guidelines on the path to the solution:

- Modularity: the solution should be based on a collection of modules that can easily be re-implemented in a wide range of technologies without major changes;
- Scalability: the throughput can be increased without modifying the architecture;
- Reusability: the different modules composing the solution should be reused to confine the design/VLSI implementation complexity within controllable limits.
- Local Clocking Strategy: In order to reduce or eliminate the negative effect of synchronization of a global clock, a Globally Asynchronous Locally Synchronous (GALS) design style should be adopted.

After enumerating the design guidelines, let the objectives be stated and prioritized. The objectives are in order of importance:

- Achievable throughput of 500 Gbits/s
- Relatively acceptable area and power consumption
- Low Latency

Most of the approaches cited in the literature survey focus on single objectives which are either speed and/or area optimization. Trade-off design style encompasses speed, area and power consumption without neglecting the scalability, the reuse and the clock locality. An example of a work that followed such approach can be found in [14].

The main idea is to produce a highly regular implementation that will allow us to meet all the objectives stated above.

4. THE ITERATIVE AES PROCESSOR

Pipelined, sub-pipelined design and loop unrolling have aimed at increasing the throughput or decreasing the latency. In the context of high speed processing for networking architectures, the latency is not a critical factor especially when it amounts around the order of 100 ns. It will certainly take more time to retrieve the routing information of a packet or to just transfer to/from the storage areas.

Based on the lower importance of the latency, we propose to implement the AES algorithm into an iterative processor. The processor can perform the four operations of the AES algorithm. One operation at a time is performed. This implies that the three hardware blocks implementing the other operations will be idle when one of the four operations is being performed. Therefore, in order to keep the hardware busy, the processor should be able to process four different data blocks simultaneously enforcing the reuse guideline. Each data block is called a *flow*. Four registers for storing the four different intermediate state values of the four different flows are then required. Moreover, because of the byte granularity of the AES algorithm, the size of data in the iterative processor is reduced to a single byte.

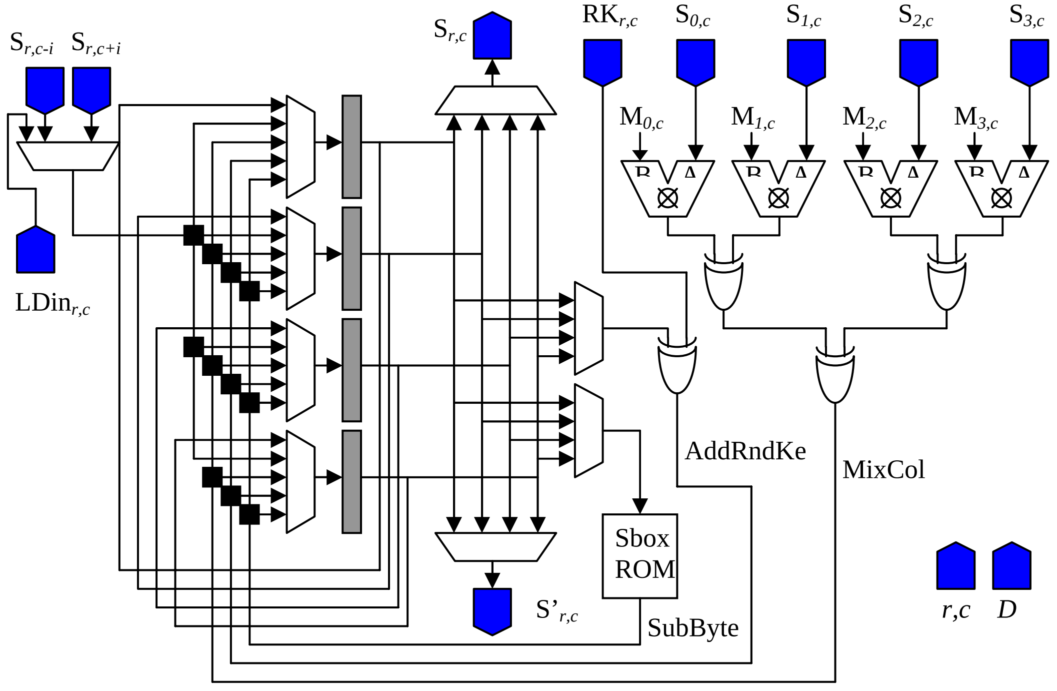

Figure 2 – Byte Datapath Internal Structure

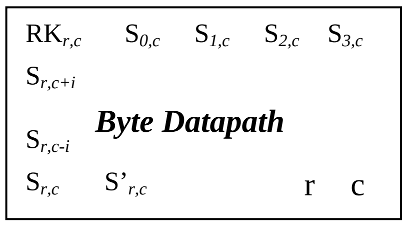

Figure 3 – Byte Datapath Black Box Representation

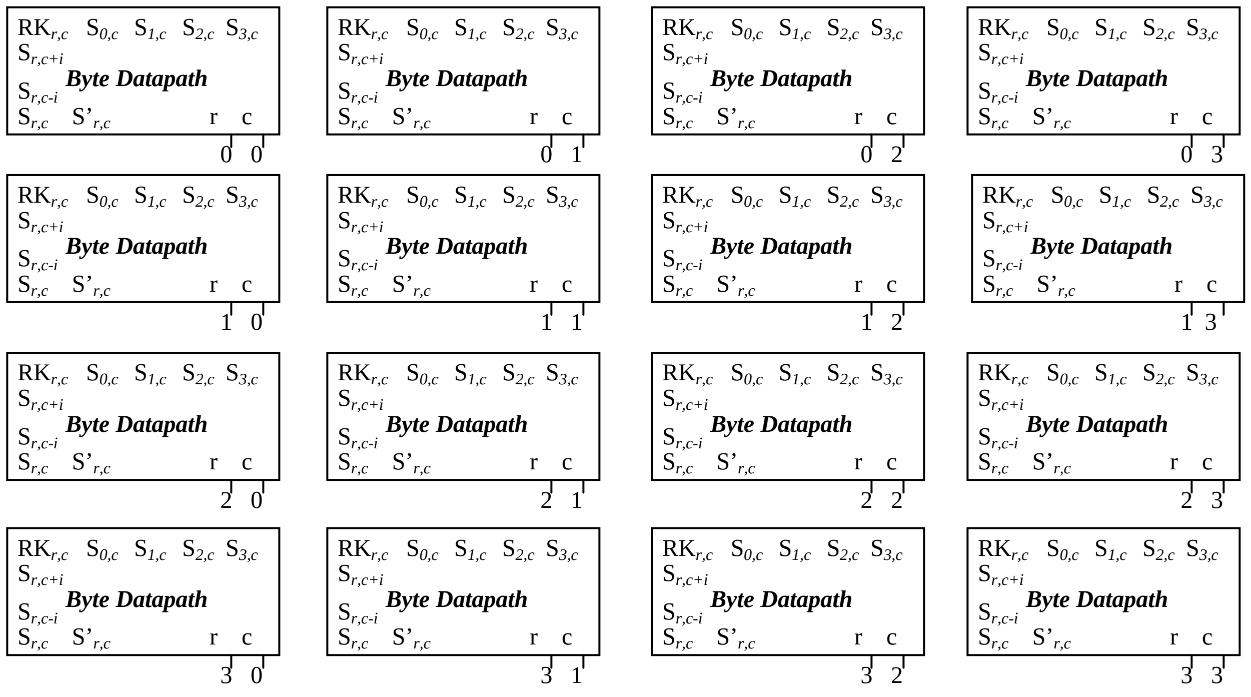

Figure 4 – AES Encryption Engine

Figures 2 and Figure 3 show the internal diagram as well as the black box representation of the byte datapath. The different connections to the state registers are realized through 4-to-1 multiplexers to select where to store the result of each one of the four operations. Each operation input circuitry is also connected to 4-to-1 multiplexers to select which flow will be processed during the current clock cycle.

The different input/output connectors are mainly used to propagate the different intermediate values needed by the shift rows or the mix-columns operations. The byte datapath is further organized in a *4 x 4* array to implement the state as described in the AES algorithm [1]. Figure 4 shows the block diagram of the AES encryption engine using a 4 x 4 array of byte datapath blocks.

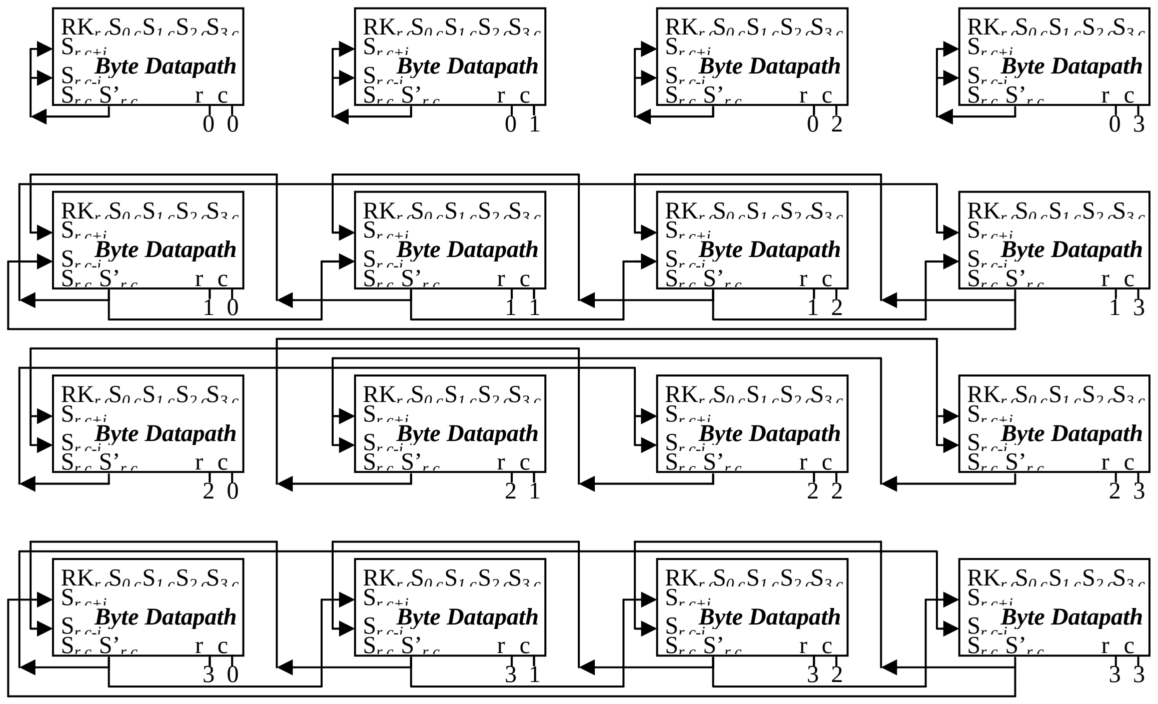

Figure 5 – Shift Rows Operation Connections

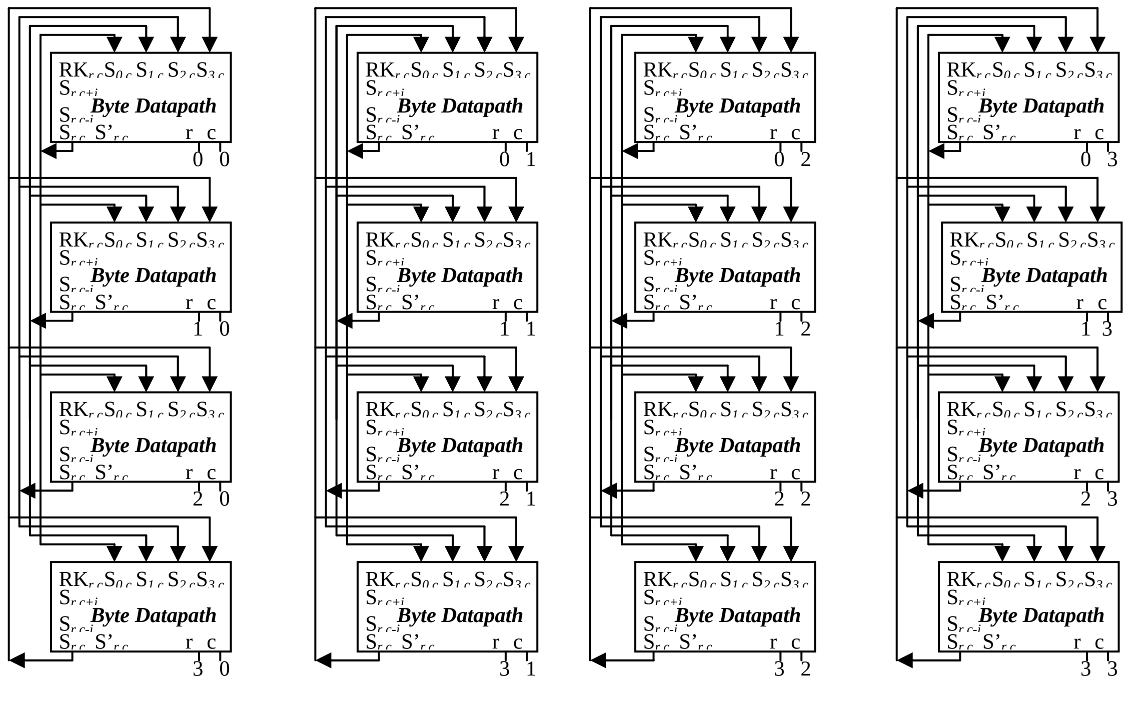

Figure 6 – Mix-Columns Operation Connections

This highly regular structure is built using a single byte datapath block instantiated regularly. Each block is customized by adequately connecting its inputs and outputs. Hence, the *(r,c)* pair that identifies the position of each block in the array is set accordingly as shown in Figure 4. The shift- rows operation as well as the mix-columns operation both require the transfer of values from a state byte to another. Figure 5 and Figure 6 show the connections of the encryption engine for implementing the shift-rows and mix-columns operations respectively.

The datapath has been designed to both implement the cipher and decipher operations of the AES algorithm. The connections of Figure 5 reflect the support of both cipher and decipher shifts for the shift-rows operation.

5. Key Expansion Unit

The key schedule generation block is used to generate the key schedule for each key. Every iteration, a different set of generated key schedule values are used within the AddRoundKey operation. The key schedule generation block has been designed so that it produces the key schedule within a minimum number of clock cycles. It is capable of producing 4 round keys of 32-bits each every clock cycle. The produced round keys are stored in the key schedule memory that is part of the key schedule generation block. A dual port memory has been selected for implementing the key schedule memory. This allows simultaneous access from the key schedule generation and the iterative processor. Because the AES algorithm key schedule generation amounts a total of 10, 12 or 14 128-bits round keys, the mapping of the key schedule in the key schedule memory starts on a 16-word boundary. This makes the decoding, retrieval and allocation mechanisms trivial.

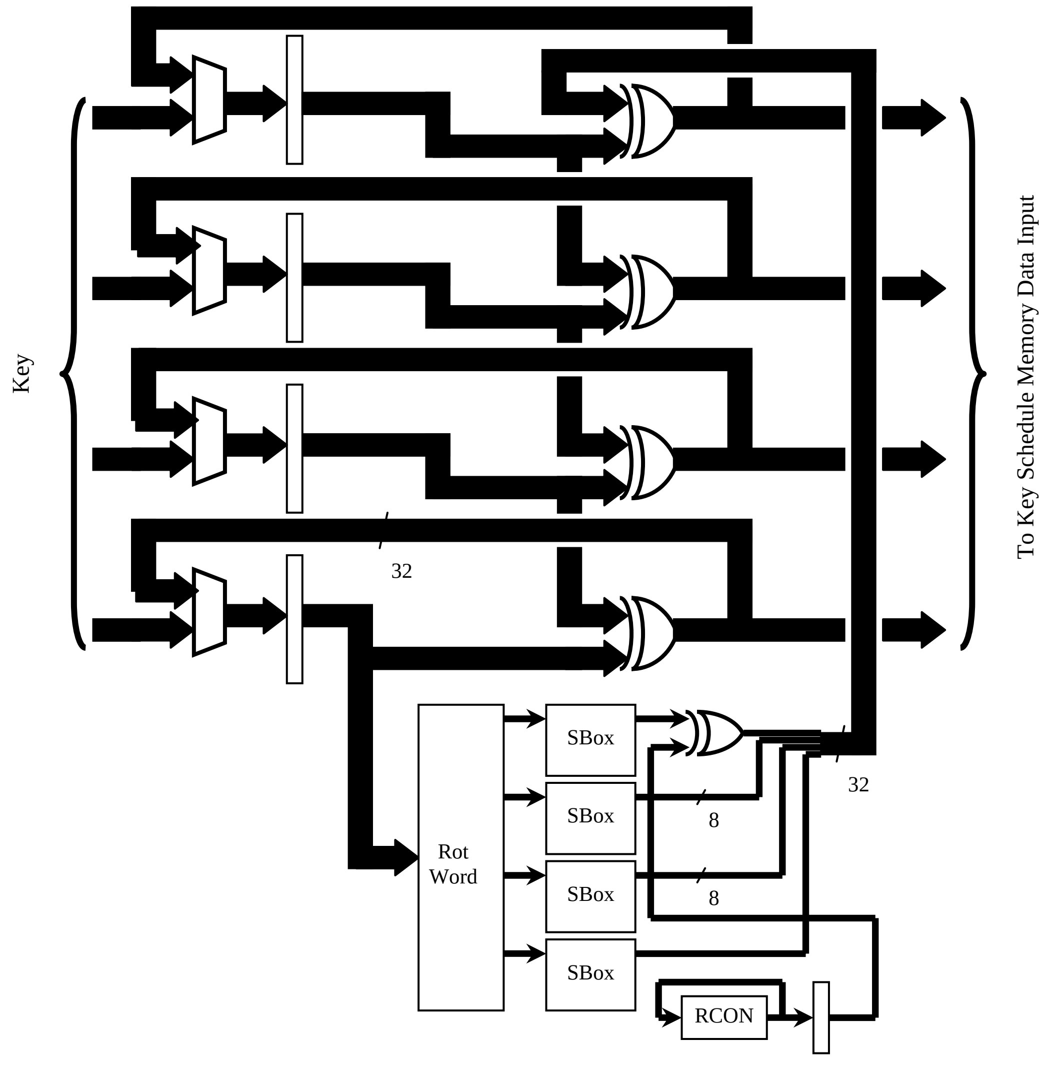

Figure 7 – Key Schedule Generation Block

Figure 7 shows the key schedule operating on four round keys of 32-bits each in every iteration. The use of four instances of Sbox ROMs facilitates the implementation.

6. 500 GBITS/S AES IMPLEMENTATION

The iterative processor above can achieve a throughput of 4 data blocks every *4 x (10 + 2) = 48* clock cycles which means one flow every *12* clock cycles. Instantiating *12* iterative processor blocks operating on *12 x 4* different data blocks will achieve a throughput of one data block every clock cycle. Following the same principle, an array of *N x 12* iterative processors will yield a throughput of *N* data blocks per clock cycle. Figure 8 shows an *N* rows *x 12* columns array of iterative processors.

The iterative processor above can achieve a throughput of 4 data blocks every *4 x (10 + 2) = 48* clock cycles which means one flow every *12* clock cycles. Instantiating *12* iterative processor blocks operating on *12 x 4* different data blocks will achieve a throughput of one data block every clock cycle. Following the same principle, an array of *N x 12* iterative processors will yield a throughput of *N* data blocks per clock cycle. Figure 7 shows an *N* rows *x 12* columns array of iterative processors.

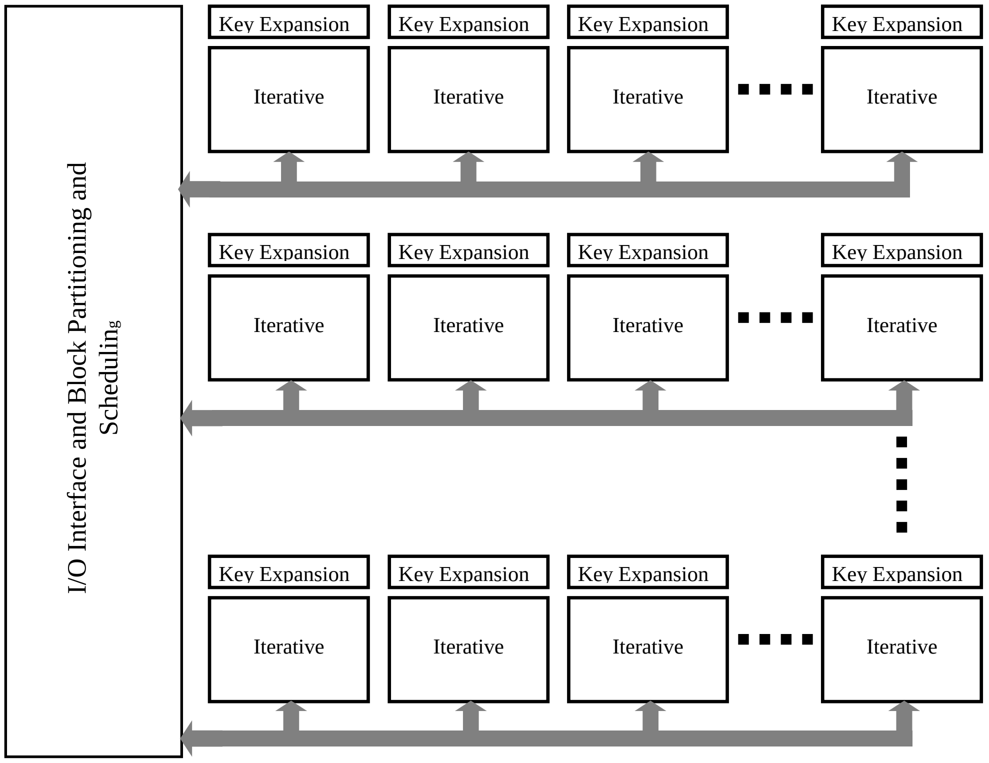

Figure 8 – Scalable AES Implementation

Table 1 determines, for different values of the clock frequency, the corresponding values of *N* in order to achieve a 500 Gbits/s throughput. Numbers are rounded to the upper integer value to keep *N* integer.

The maximum number of logic levels within a single iterative processor is less than 10 gates. It implies that very high frequencies can be achieved.

TABLE 1:VALUES OF *N*

Clock Frequency (MHz)	***N***	**# of Blocks**
100	40	480
200	20	240
300	14	168
400	10	120
500	8	96
600	7	84
800	5	60

The clock frequency values retained can easily be implemented in several technologies. A preliminary gate estimation gives a total of 25000 gates per iterative processor which produces a maximum of 12 million gates for the 100 MHz clock frequency and a minimum of 1,5 millions for the 800 MHz. These levels are acceptable within the ASIC implementation range.

7. I/O Interface

Several choices are available to implement the I/O interface. All of the solutions are based on the use of low voltage differential serial links. The following possibilities are:

- 200 differential serial links running at 2.5 Gbits/s will require a total data pins of 1000 pins.
- 12 different SPI-5 interfaces from the OIF consortium [21]. Each SPI-5 interface uses 38 pins in both directions requiring a total of 456 pins
- 160 differential serial links running at 3.125 Gbits/s which requires a total of 640 pins
- More choice will be at hand with the availability of faster low power, low voltage, differential serial links.

8. Task Distribution and Scheduling

The flow of data in and out of the iterative processors is inherently parallel and needs a structure that allows the seamless distribution of the load while keeping the throughput at its maximum values. In order to avoid congestion, a distributed scheduling model should be followed otherwise congestion will introduce delays that will lower the throughput.

The distributed model suggested here is based on the use of an internal network on a chip that connects the iterative processors, the key generation blocks and the scheduling elements.

The scheduling elements are two kinds. The first layer that is responsible for receiving the requests from the different interface ports and mapping them onto the iterative processors. As the interface is multiple ports by construction, each scheduling element, in the first layer, should be responsible for receiving and processing the requests received from one port. The second layer is responsible for scheduling the received requests for a single (or a group) of iterative processors. This second layer will be able to accept or reject new jobs sent by elements in the first layer implementing a de-facto load-balancing. The generation of periodic status messages broadcast to the first layer will certainly reduce jitter and delay in processing requests.

9. Conclusion

In this paper a novel approach for implementing the AES encryption algorithm has been presented. This approach showed how to achieve a very high throughput of 500 Gbits/s while keeping the complexity of the design relatively low by using a regular design. Trading off power consumption with area is possible by selecting the appropriate pair of clock frequency/Number of rows opening the door for implementations targeting a wide range of technologies. Simulations and tentative implementations will be carried out in the future to determine the actual performance numbers in terms of speed and area. The interface feasibility has been discussed and proved feasible in nowadays technologies. For the task distribution and scheduling, the overall approach, based on the use of a network-on-chip has been presented.

Acknowledgment

Facilities support by King Fahd University of Petroleum and Minerals is highly appreciated by the author.

References

[1] *Advanced Encryption Standard (AES)*, Nov. 26, 2001.

[2] Kotturi, D.; Seong-Moo Yoo; Blizzard, J.; "AES Crypto Chip Utilizing High-Speed Parallel Pipelined Architecture", IEEE International Symposium on Circuits and Systems, ISCAS2005. 23-26 May 2005 Page(s):4653 - 4656 Vol. 5

[3] Xinmiao Zhang; and K.K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm", IEEE Transactions on VLSI Systems, Volume 12, Issue 9, Sept. 2004 Page(s):957 – 967

[4] S. Morioka, S. and A. Satoh,; "A 10 Gbps Fu11-AES Crypto Design with a Twisted-BDD S-Box Architecture"IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002, 16-18 Sept. 2002 Page(s):98 – 103

[5] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalist. presented at *Proc. 3rd AES Conf. (AES3)*. Available:http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html

[6] V. Fischer and M. Drutarovsky, “Two methods of Rijndael Implementation in reconfigurable hardware,” in *Proc. CHES 2001*, Paris, France, May 2001, pp. 77–92.

[7] K. Gaj and P. Chodowiec. Comparison of the hardware performance of the AES candidates using reconfigurable hardware. presented at *Proc. 3rd AES Conf. (AES3)*.

[8] H. Kuo and I. Verbauwhede, “Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm,” in *Proc. CHES 2001*, Paris, France, May 2001, pp. 51–64.

[9] M. McLoone and J. V. McCanny, “Rijndael FPGA implementation utilizing look-up tables,” in *IEEE Workshop on Signal Processing Systems*, Sept. 2001, pp. 349–360.

[10] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, “Efficient implementation of Rijndael encryption with composite field arithmetic,” in *Proc. CHES 2001*, Paris, France, May 2001, pp. 171–184.

[11] K. U. Jarvinen, M. T. Tommiska, and J. O. Skytta, “A fully pipelined memoryless 17.8 Gbps AES-128 encryptor,” in *Proc. Int. Symp. Field-Programmable Gate Arrays (FPGA 2003)*, Monterey, CA, Feb. 2003, pp. 207–215.

[12] G. P. Saggese, A. Mazzeo, N. Mazocca, and A. G. M. Strollo, “An FPGA based performance analysis of the unrolling, tiling and pipelining of the AES algorithm,” in *Proc. FPL 2003*, Portugal, Sept. 2003.

[13] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat, “Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements & design tradeoffs,” in *Proc. CHES 2003*, Cologne, Germany, Sept. 2003.

[14] S. Mangard, M. Aigner, M. and S. Dominikus, “A highly regular and scalable AES hardware architecture”, IEEE Transactions on Computers, Volume 52, Issue 4, April 2003 Page(s):483 - 491

[15] C. Paar, “Efficient VLSI architecture for bit-parallel computations in Galois field,” Ph.D. dissertation, Institute for Experimental Mathematics, University of Essen, Essen, Germany, 1994.

[16] M. H. Jing, Y. H. Chen, Y. T. Chang, and C. H. Hsu, “The design of a fast inverse module in AES,” in *Proc. Int. Conf. Info-Tech and Info-Net*, vol. 3, Beijing, China, Nov. 2001, pp. 298–303.

[17] C. C. Lu and S. Y. Tseng, “Integrated design of AES (advanced encryption standard) encrypter and decrypter,” in *Proc. IEEE Int Conf. Application Specific Systems, Architectures Processors*, 2002, pp. 277–285.

[18] X. Zhang and K. K. Parhi, “Implementation approaches for the advanced encryption standard algorithm,” *IEEE Circuits Syst. Mag.*, vol. 2, no. 4, pp. 24–46, 2002.

[19] X. Zhang, Parhi, K.K., “An efficient 21.56 Gbps AES implementation on FPGA”, Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004. Volume 1, 7-10 Nov. 2004 Page(s):465 - 470 Vol.1

[20] A. Hodjat, I. Verbauwhede, “Minimum area cost for a 30 to 70 Gbits/s AES processor”, Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2004, 19-20 Feb. 2004 Page(s):83 – 88

[21] The Optical Internetworking Forum at www.oiforum.org