Multi-Gigabit GCM-AES Architecture Optimized for FPGAs

Stefan Lemsitzer, Johannes Wolkerstorfer, Norbert Felber, Matthias Braendli

TU Graz, IAIK/ETH Zürich, IIS
stefan@lemsitzer.com

Abstract. This paper presents a design-space exploration of the Galois/Counter Mode (GCM) algorithm with Advanced Encryption Standard (AES) as underlying block cipher for high throughput applications to combine data encryption and message authentication on FPGAs. Four different degrees of parallelism were implemented, namely a 128-, 64-, 32- and 16-bit wide data path calculating an output block in 1, 2, 4 and 8 clock cycles, respectively. Regarding the AES algorithm different *SubBytes()* and round architectures were evaluated against each other. For the multiplier required for GCM, two bit-parallel, a digit-serial and a hybrid architecture were evaluated. The different architectures were designed, implemented and tested on a Xilinx Virtex4-FX100 FPGA. All architectures support key lengths of 128, 192 and 256 bits and are equipped with a ready-to-use interface for real-world applications. A throughput of 15.3 Gb/s was reached. It pointed out that throughput rates for state-of-the-art communication channels can be achieved using reasonable hardware resources. The results comparing slice counts, RAM usage and speed are presented.

Key words: Galois/Counter Mode (GCM), Advanced Encryption Standard (AES), Very Large Scale Integration (VLSI), high throughput, Field Programmable Gate Array (FPGA), digit-serial multiplier, hybrid multiplier, bit-parallel multiplier

1 Introduction

The AES algorithm was published by NIST as FIPS-197 standard in 2001 [1]. For this algorithm, several modes of operation were proposed such as ECB, CBC, CFB, OFB and CTR [2]. These modes provide data encryption but no data authentication. Therefore, NIST proposed among others GCM as a mode of operation supporting authenticated encryption [3]. In this work, the GCM algorithm was implemented to be used in a cryptographic system. For this project the goal was to find an architecture with a reasonable trade off between speed, area utilization and

IO-behavior instead of concentrating on achieving maximum throughput only.

2 Related Work

Regarding AES, several implementations for a broad range of applications are presented in [4–6]. Focusing on GCM, [7] proposes an ASIC implementation using a $0.13\mu m$ CMOS process reaching a throughput of 42.67 Gb/s. Several AES architectures and optimization criteria, as well as two different GCM multiplier architectures were implemented. In [8], a GCM core using a $0.18\mu m$ process was designed reaching a throughput of 34.7 Gb/s. Furthermore, there exist a few implementations from industry [9–11].

3 Algorithm Specification

3.1 Notation

Within this paper, a bit stream $d_0...d_i$ is grouped into bytes $b_0...b_j$, where the MSBit is the bit with the lowest index according to Fig. 1. The operation $||$ refers to the concatenation of two bit streams. The exclusive or operation (XOR) is denoted as $\oplus$. The index $*$ as in C^* denotes a block which might be shorter than 128 bit. $\text{MSB}_i(C)$ refers to the i most significant bits of a block C.

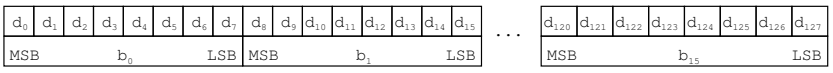

Fig. 1. Bit/Byte representation

3.2 The Galois/Counter Mode (GCM)

GCM is a mode of operation for block ciphers that provides authenticated encryption by using universal hashing over a binary Galois field as can be seen in Fig. 2. As inputs, the algorithm expects a secret key K of appropriate length for the underlying block cipher, a distinct initial vector IV, the plaintext P and some additional authenticated data AAD. As outputs, a ciphertext C and an authentication tag T are generated. P, C and AAD are grouped into 128-bit blocks $P = P_0 \cdots P_n, C =$

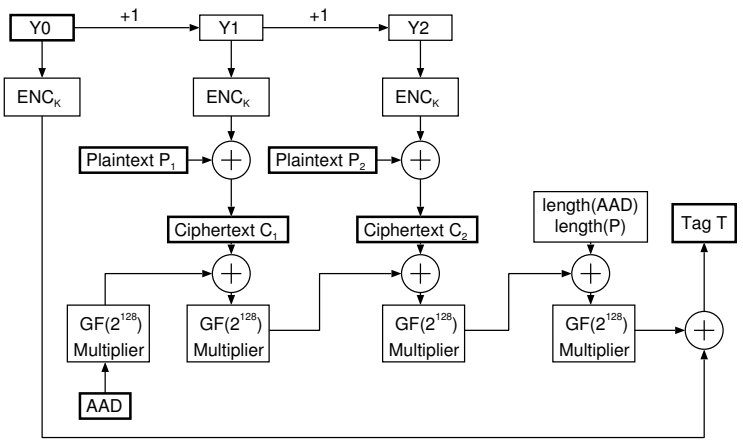

Fig. 2. Authenticated encryption overview (simplified)

$C_0 \cdots C_n$, $AAD = AAD_0 \cdots AAD_m$. The constant H is obtained by encrypting a zero block under the given secret key. Note, that for GCM only the encryption algorithm (ENC) of the underlying block cipher is needed. The authenticated encryption operation is defined the following way:

$$\begin{aligned} H &= \mathrm{ENC}(K, 0^{128}) \\ Y_0 &= \begin{cases} IV||0^{31}1 & \text{if } \mathrm{len}(IV) = 96 \\ \mathrm{GHASH}(H, \{\}, IV) & \text{otherwise} \end{cases} \\ Y_i &= \mathrm{incr}(Y_{i-1}) \qquad \text{for } i = 1, \ldots n \\ C_i &= P_i \oplus (\mathrm{ENC}(K, Y_i) \qquad \text{for } i = 1, \ldots, n-1 \\ C_n^* &= P_n^* \oplus \ \mathrm{MSB}_u(\mathrm{ENC}(K, Y_n)) \\ T &= \ \mathrm{MSB}_t(\mathrm{GHASH}(H, AAD, C) \oplus \mathrm{ENC}(K, Y_0)) \end{aligned} \tag{1}$$

The *GHASH* function, which is mainly responsible for tag creation, is shown as follows, the indices u, v are denoting the missing bits to a full block of 128 bits:

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus AAD_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (AAD_m^* || 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_m^* || 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\mathrm{len}(AAD) || \mathrm{len}(C))) \cdot H & \text{for } i = m+n+1 \end{cases} \tag{2}$$

For decryption, the ciphertext C and the plaintext P change their position in (1) except for tag creation.

4 GCM-AES Implementation

The presented hardware implementation of GCM-AES computes the encryption and the GCM authentication in parallel. The main parts of the combined encryption and authentication core are shown in Fig. 3. The

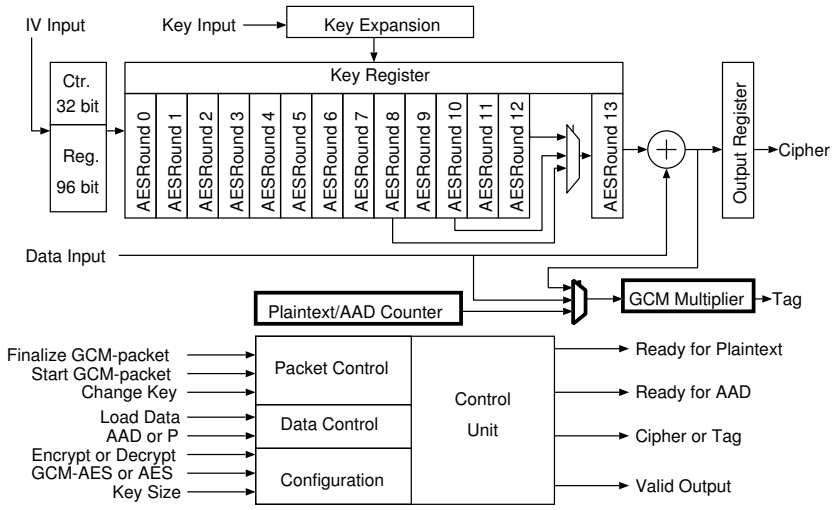

Fig. 3. Data path of the whole GCM Core

following restrictions have been specified: Input data is 128 bit aligned which is required for the cryptographic system. The sizes of additional authenticated data (AAD) and plaintext (P) are counted internally and do not need to be applied externally. Two status signals indicate whether loading of AAD or P is allowed. The length of the IV is restricted to 96 bits which is recommended for high throughput implementations [3]. Key changes and GCM-packet starts are possible in the same clock cycle. At key change, the configuration (Key size, Encryption/Decryption, AES/GCM-AES) of the core is configured as well.

4.1 Hardware Platform

A PCIe XpressFX [12] evaluation board equipped with a Xilinx Virtex4-FX100 FPGA was used. There are 42176 slices and 376 block RAMs of

18Kb available on this device. The block RAMs in Xilinx Virtex4 FPGAs are clocked elements and behave like registers. In the following sections this feature is used extensively.

4.2 AES Round Architectures

An architecture which uses 14 pipelined AES rounds as shown in Fig. 3 was designed. The rounds operate concurrently without using inner pipelining to speed up the circuit at cost of latency as done in [4]. This was not necessary because the GCM multiplication mainly defined the critical path. Due to this architecture decision, all round keys need to be made available within each cycle and therefore have to be stored within a large register bank. In order to calculate one data block consuming the same amount of cycles as the GCM multiplier, four different implementations using different data path widths have been designed and evaluated.

128-bit data path. This architecture requires 16 *SubBytes()* and four *MixColumns()* blocks as shown in Fig. 4. When block RAMs are used for the *SubBytes()* operation, the 128-bit pipeline register at the output can be saved. A complete AES block is calculated within each cycle.

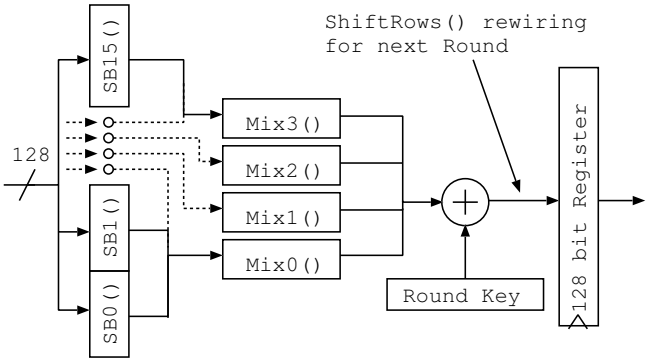

Fig. 4. AES round architecture using a 128-bit data path

64-bit data path. This architecture shown in Fig. 5 uses 8 *SubBytes()* blocks, two *MixColumns()* blocks and two independent 32-bit registers to store intermediate values. One of them is reloaded in each cycle and stores b_0, b_3, b_4 and b_5, and the other one is reloaded in each other cycle storing b_1, b_2, b_6 and b_7. Using such a wiring no multiplexer is needed in

front of the two register banks. The *ShiftRows()* operation can easily be performed by the output multiplexer.

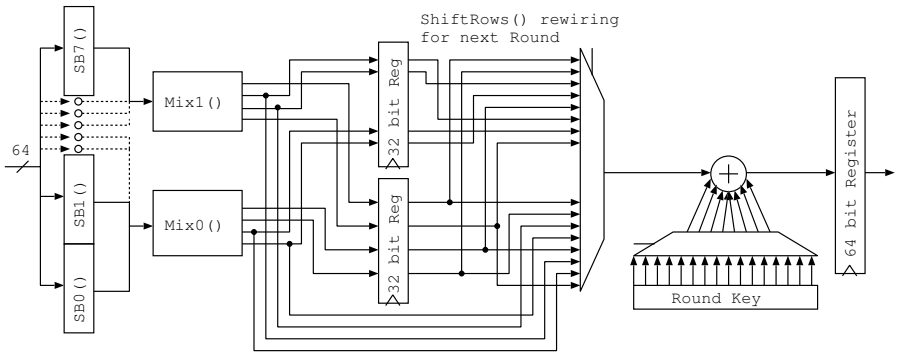

Fig. 5. AES round architecture using a 64-bit data path

32-bit data path. Figure 6 shows an architecture which uses four *SubBytes()* and one *MixColumns()* block, followed by six shift registers which delay the incoming bytes according to their usage within *ShiftRows()*. Note that there is no multiplexer needed to load the shift registers.

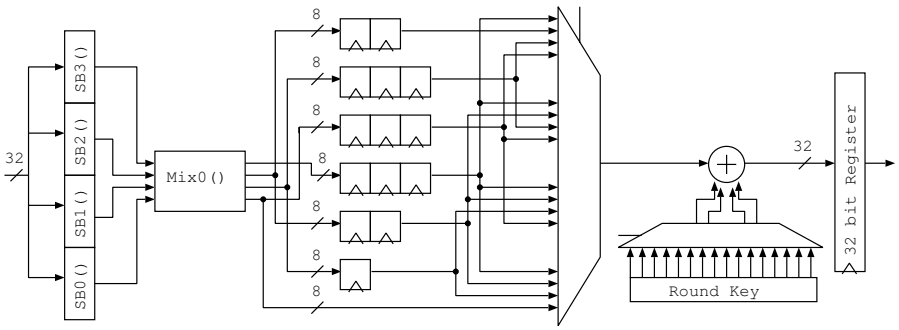

Fig. 6. AES round architecture using a 32-bit data path

16 bit data path. This architecture is similar to the previous one but only two *SubBytes()* blocks are required. Because *MixColumns()* processes 32-bits of input data, there is an additional 32-bit input register in front of *MixColumns()* to store two outputs of the *SubBytes()* blocks.

4.3 *SubBytes()* Implementation

For AES, the main time and area critical part is the *SubBytes()* operation. It performs a byte substitution according to a substitution table called S-box. This table is created by taking the multiplicative inverse of each byte in the finite field GF(2^8), followed by an affine transformation over GF(2) [1]. Three different architectures for this operation were evaluated.

As a first approach, the substitution table can be precomputed and defined as a constant array which then is optimized by the synthesizer. In [5] was shown that the truth table of the S-Box is an almost perfect random number table which makes it very difficult to optimize its structure when using standard optimization algorithms.

In [13], Rijmen showed a method to calculate the substitution values online using operations in the composite field GF($(2^4)^2$). The detailed architecture is described in [14]. Using that method, the area utilization could be decreased by 32% compared to lookup tables.

For implementations on a Xilinx Virtex 4 FPGA the SBoxes can be stored in the dual-port block RAMs which are configured as ROMs. The dual-port capability halves the hardware requirements. If block RAMs are not a scarce resource for the given target application, this implementation results in a very fast circuit using no other FPGA resources. In addition, the output registers of the AES rounds (Fig. 4, 5, 6) can be removed due to the synchronous behavior of the block RAMs.

4.4 GCM Multiplier

GCM requires a multiplier for the finite field GF(2^{128}) which needs to be carefully designed to reach reasonable area utilization and performance. The multiplier uses the irreducible polynomial $p(x) = x^{128}+x^7+x^2+x+1$ to compute $C = AB \bmod p(x)$. In [15], several implementation options for such a multiplier are proposed, including bit-parallel, digit-serial and hybrid multipliers.

Bit-parallel implementations. An initial architecture was realized by designing a block which multiplies a 128 bit input by x in $GF(2^{128})$. This partial product generator was then instantiated 128 times and optimized using the synthesizer.

In a second architecture, the method proposed in [16] was implemented. The matrices and vectors defined in [16] needed to be precomputed and exported as VHDL files using MATLAB. The resulting circuit contains bitwise multiplication (AND Gates) and addition (XOR Gates).

Digit-serial Implementation. A digit-serial implementation was designed similar to the architecture proposed in [17] which can be seen in Fig. 7. This multiplier calculates a multiplication in the field $GF(2^k)$ in $n = k/m$ cycles. A multiplier in $GF(2^k)$ multiplies m-bits of the input A with the input B. The multiplication by x^m can be calculated by aligning intermediate results accordingly.

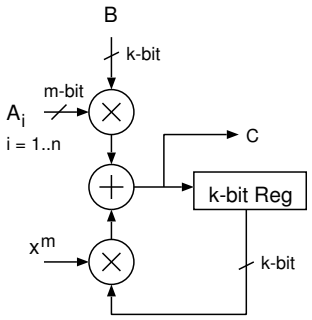

Fig. 7. Digit-serial multiplier

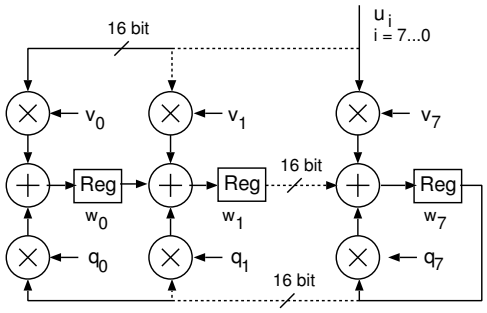

Fig. 8. Hybrid multiplier

Hybrid Multiplier. A Galois field GF(2^k) can be represented by a composite field GF($(2^n)^m$) where $k = mn$. Arithmetic calculations can then be performed using bit-parallel architectures in the subfield GF(2^n) and bit-serial structures in the extension field GF($(2^n)^m$). Additionally, the inputs and outputs need to be mapped between the two field representations. For computations in the composite field, an irreducible polynomial $R(x) = x^n + r_{n-1}x^{n-1} + ... + r_0$ with coefficients $r_i \in$ GF(2) constructing the subfield GF(2^n), and an irreducible polynomial $Q(x) = x^m + q_{m-1}x^{m-1} + ... + q_0$ with coefficients $q_i \in$ GF(2^n) constructing the extension field GF($(2^n)^m$), need to be found. In [18], a method to derive the mentioned polynomials and two conversion functions *map* and map^{-1} between the composite field and the field GF(2^k) was proposed. Using this method, a hybrid multiplier was created using eight cycles to create one output block as shown in Fig. 8. Therefore, the composite field GF($(2^{16})^8$) was constructed. The structure of the multiplier was proposed in [19]. The input words $u_i = \{u_0, u_1, ..., u_7\} = \text{map}(A)$ starting with the most significant word are multiplied with the input words $v_i = \{v_0, v_1, ..., v_7\} = \text{map}(B)$ by a 16-bit multiplier. This multiplication in the subfield GF(2^{16}) uses the irreducible polynomial $R(x) = x^{16} + x^{15} + x^{13} + x^{12} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^4 + 1$. The output

of the register w_7 is multiplied with the coefficients $q_i \in \mathrm{GF}(2^{16})$ and fed back according to Fig. 8. After eight cycles, the output can be read out of the registers w_i. In GCM, one input of the multiplier is always the value H as described in (2), which changes only if the secret key changes. Therefore, within this architecture only one *map* circuit is needed.

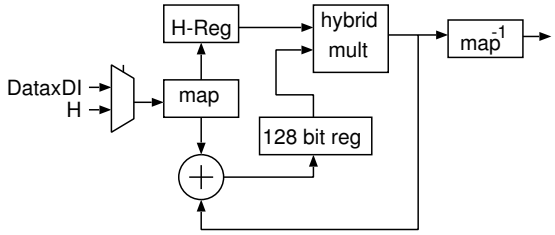

Fig. 9. Architecture of the GCM Hybrid multiplier

Latency. Depending on the key size and the operation mode, a different number of cycles has to be waited after key change before the first AAD/P can be loaded provided that the start GCM-packet signal was applied simultaneously to the key change. Tab. 1 gives a summary of the number of wait cycles. This delay is not a real drawback because the key change occurs infrequently compared to the GCM-packet change. Pipelining of the key change would result in a complex control unit due to the reload of the core configuration at the key change. If operated correctly, at GCM-packet changes, no plaintext can be loaded for $3 + \lceil \mathrm{length}(AAD)/128 \rceil$ cycles in GCM-AES mode. Assuming a GCM-packet contains 128 data blocks and one AAD block, this results in an overhead of 3 %. In AES-only mode, the plaintext can be loaded within each cycle.

	128 bit keys	192 bit keys	256 bit keys
AES only mode	12	14	16
GCM-AES mode	14	16	18

Table 1. Latency for key change

5 Results

The combined GCM-AES hardware was modeled in VHDL. The HDL modules were synthesized using Synplify Pro and then placed and routed

using Xilinx ISE. For each architecture the result with the best throughput/slice ratio after P&R is listed in the tables. A golden model test and a long time test have been performed on the PCIe XpressFX evaluation board using on-board memory as stimuli and response storage. The results for the AES core without authentication are listed separately.

5.1 AES Core

The AES core implements the Counter mode. Key sizes of 128, 192 and 256 bits are supported. The architecture using block RAMs is very suitable and competitive for throughputs between 5 and 15 Gb/s at moderate clock frequencies as can be seen in Tab. 2 and Tab. 3. Nevertheless, the costs for block RAMs must not be neglected. For all presented architectures, the low latency enables fast key and counter changes. The slice count for the AES rounds using block RAMs is almost independent of the width of the data path.

	16-bit data path			32 bit data path			64-bit data path			128-bit data path		
	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
Slices	4.0k	5.6k	5.8k	3.9k	7.3k	7.5k	3.8k	8.9k	11.1k	3.8k	13.7k	18.4
Clk [MHz]	170	130	160	140	110	130	140	90	140	140	100	140
Block RAMs	15	0	0	30	0	0	58	0	0	114	0	0
Gb/sec	2.7	2.1	2.56	4.5	3.5	4.2	9	5.8	9	17.9	12.8	17.9
kb/slice	676	370	436	1152	479	556	2326	651	804	4719	933.4	975.7

Table 2. Comparison of different architectures of AES Counter mode. (a): RAM, (b): composite field, (c): LUT

	Device	Slices	# RAMs	Gb/s	kb/slice	Key Size	Latency
This work	Virtex4-FX100	3.8k	114	17.9	4719	128/192/256	12/14/16
Hodjat [4]	XC2VP20-7	5.17k	84	21.54	4161	128	31
Standaert [20]	Virtex 3200 E	2.78k	100	11.8	4229	128	21

Table 3. Comparison of three AES implementations using block RAMs

5.2 GCM-AES Core

For the 128-bit data path implementation the multiplier using the shifted polynomial basis led to a faster but larger design compared to the syn-

thesizer optimized architecture. The hybrid multiplier pointed out to be larger and slower on a FPGA implementation than the digit-serial version. This was mainly caused by the generated polynomial $R(x)$ used within the subfield multiplier which has eleven non-zero coefficients. (Note that the pentanomial used within the GCM algorithm has only five non-zero coefficients). Tab. 4 lists the results for the 128-bit, Tab. 5 for the 64-bit, Tab. 6 for the 32-bit and Tab. 7 for the 16-bit data path. Note that single architectures could be further optimized by improving the attributes and constraints set for synthesis and P&R individually.

	Frequency	Slices	# block RAMs	Gb/s	kb/slice	Limit
Shifted polynomial basis, RAM	110 MHz	13.2k	114	14.1	1062	GCM
Shifted polynomial basis, comp.	90 MHz	23.2k	0	11.5	496	AES
Shifted polynomial basis, LUT	120 MHz	27.8k	0	15.3	552	GCM
Synthesizer optimized, RAM	100 MHz	12k	114	12.8	1065	GCM
Synthesizer optimized, comp.	90 MHz	21.6k	0	11.5	534	AES
Synthesizer optimized, LUT	90 MHz	26.4k	0	11.5	435.9	GCM

Table 4. GCM-AES FPGA core calculating one cipher block within each cycle - 128-bit data path.

	Frequency	Slices	# block RAMs	Gb/s	kb/slice	Limit
Digit serial, RAM	130 MHz	7.7k	58	8.32	1076	GCM
Digit serial, comp.	90 MHz	12.9k	0	5.8	446	AES
Digit serial, LUT	120 MHz	15k	0	7.7	513	GCM

Table 5. GCM-AES FPGA core calculating a cipher block every other cycle - 64-bit data path.

	Frequency	Slices	# block RAMs	Gb/s	kb/slice	Limit
Digit serial, RAM	140 MHz	6.0k	30	4.48	738	GCM
Digit serial, comp.	120 MHz	9.4k	0	3.8	407	AES
Digit serial, LUT	130 MHz	9.8k	0	4.16	422	GCM

Table 6. GCM-AES FPGA core calculating a cipher block every four cycles - 32-bit data path.

	Frequency	Slices	# block RAMs	Gb/s	kb/slice	Limit
Digit serial, RAM	150 MHz	5.5k	15	2.4	436	GCM
Digit serial, comp.	130 MHz	6.8k	0	2.0	305	AES
Digit serial, LUT	150 MHz	7.4k	0	2.4	322	GCM
Hybrid, RAM	110 MHz	7.3k	15	1.8	240	GCM
Hybrid, comp.	110 MHz	8.4k	0	1.8	211	GCM
Hybrid, LUT	120 MHz	9.3k	0	2.0	212	GCM

Table 7. GCM-AES FPGA core calculating a cipher block every eight cycles - 16-bit data path.

6 Conclusion

Different high-throughput architectures of the GCM-AES algorithm have been compared for their use in FPGAs. Two bit-parallel, a digit-serial and a hybrid multiplier were designed to perform multiplications in the field $GF(2^{128})$. Three different *SubBytes()* implementations were evaluated. All cores are equipped with a ready-to-use interface for real world applications and support 128-, 192- and 256-bit keys. The FPGA designs were tested on a Xilinx Virtex4-FX100.

An efficient GCM-AES core reached a throughput of 14.1 Gb/s using 13.2 k-slices (31 % of Xilinx Virtex4-FX100) and 114 block RAMs (30 % of Xilinx Virtex4-FX100) running at 110 MHz. For fast FPGA implementations, the GCM multiplier was the limiting factor. The hybrid multiplier using the irreducible polynomial $R(x)$ created within this work pointed out to perform worse than its digit-serial competitor.

References

1. NIST: Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197 (2001) `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
2. Dworkin, M.: Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A (2001)
3. NIST: CSRC Proposed Modes of Operation (March 2007) `http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/index.html`.
4. Hodjat, A., Verbauwhede, I.: A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on (2004)
5. Moroioka, S., Satoh, A.: A 10 Gbps Full-AES Crypto Design with a Twisted-BDD S-Box architecture. Proceedings of the 2002 IEEE International Conference on Computer Design: ICCD 02 (2002)
6. Hamalainen, P., Alho, T., Hannikainen, M., Hamalainen, T.D.: Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In: DSD

'06: Proceedings of the 9th EUROMICRO Conference on Digital System Design, Washington, DC, USA, IEEE Computer Society (2006) 577–583
7. Satoh, A.: High-speed Hardware Architectures for Authenticated Encryption Mode GCM. Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium ISCAS 2006 (2006)
8. Yang, B., Mishra, S., Karri, R.: A High Speed Architecture for Galois/Counter Mode of Operation (GCM). Cryptology ePrint Archive, Report 2005/146 (2005) `http://eprint.iacr.org/`.
9. Helion Technology: AES-GCM Cores (March 2007) `http://www.heliontech.com/aes_gcm.htm`.
10. Algotronics Ltd.: GCM Extension for AES G3 Core (March 2007) `http://www.algotronix.com/engineering/aes_gcm.html`.
11. Elliptic Semiconductor Inc.: High Throughput AES-GCM Core - 5 Gbps (March 2007) `http://www.ellipticsemi.com/pdf/CLP-24_60102.pdf`.
12. PLD Applications Inc.: PCIe Xilinx-based Prototyping Boards (March 2007) `http://www.plda.com/products/boards_xilinx.php`.
13. Rijmen, V.: Efficient Implementation of the Rijndael S-box `http://www.iaik.tugraz.at/research/krypto/AES/old/~rijmen/rijndael/sbox%.pdf`.
14. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC Implementation of the AES SBoxes. B. Preneel (Ed.): CT-RSA 2002, LNCS 2271 (2002)
15. Paar, C.: Implementation Options for Finite Field Arithmetic for Elliptic Curve Cryptosystems. The 3rd workshop on Elliptic Curve Cryptography, October 1999. (1999)
16. Reyhani-Masoleh, A., Hasan, M.A.: Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over GF(2^m). IEEE Transactions on Computers **53**(8) (2004) 945–959
17. Wenkai Tang, H.W., Ahmadi, M.: VLSI implementation of bit-parallel word-serial multiplier in $GF(2^{233})$. IEEE-NEWCAS Conference, 2005. The 3rd International (2005)
18. Sunar, B., Savas, E., Koc, C.K.: Constructing Composite Field Representations for Efficient Conversion. IEEE Trans. Comput. **52**(11) (2003) 1391–1398
19. Paar, C., Fleischmann, P., Soria-Rodriguez, P.: Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents. IEEE Trans. Comput. **48**(10) (1999) 1025–1034
20. Standaert, F.X., Rouvroy, G., Quisquater, J.J., Legat, J.D.: Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Tradeoff. Lecture Notes in Computer Science **2779/2003** (2003) 334–350