Efficient Parallel-Pipelined GHASH

Ilia Kalistru
ilia.kalistru@infotecs.ru

***Abstract*—Galois/Counter Mode (GCM) is a widely used mode for authenticated encryption. Galois Hash (GHASH) is an important component of it. Two efficient hardware implementations of GHASH are presented in the paper. The first implementation uses several pipeline stages for Galois Field multiplication. The second implementation uses several pipelines similar to the pipeline of the first implementation. Unlike previously known parallel-pipelined implementations of GHASH, the proposed ones have improved data flow. They efficiently use all pipeline stages of the GF multiplier in each clock cycle. Therefore, performance is improved.**

***Index Terms*—GHASH, Message authentication, Pipeline processing, Data flow computing, Parallel architectures, Parallel algorithms.**

I. Introduction

GALOIS-HASH (GHASH) is widely used for message authentication [1] [2] [3]. For example, it is an important part of Galois/Counter Mode (GCM) of operation for symmetric key block ciphers [4]. In order to design a high speed low latency encryption unit which uses GCM for data encryption and authentication, one must find an efficient way to calculate GHASH. For a low latency encryption unit it can be impossible to employ parallel processing of several data frames for speed-up because it requires accumulation of many frames. It is needed n frames to be accumulated for efficient use of n GHASH calculation modules if each module calculates GHASH for its own frame. Such accumulation of many frames can take too much time. This work focuses on a design of a device for fast calculation of GHASH without such accumulation. The device is implemented on a field-programmable gate array (FPGA) integrated circuit.

GHASH has properties that make possible speed improvements. It can be represented as

$$\text{GHASH}(S, H) = X_l,$$

$$X_i = \begin{cases} 0, & \text{if } i = 0 \\ (X_{i-1} \oplus S_i)H, & \text{if } 1 \le i \le l \end{cases},$$

where l is a number of blocks of the frame to be processed, S_i - blocks of the frame, H - key polynomial, k - number of bits in each block, $\oplus$ - addition in GF(2^k), and multiplication is also performed in GF(2^k). By blocks we call the blocks of the additional authenticated data (AAD), the blocks of the encrypted data (ED), and the additional block, containing the length of the AAD field and the length of the ED field.

Parallel-pipelined device was proposed in [5]. It uses several multipliers for GHASH calculation while each multiplier is implemented with several pipeline stages. Pipelined multiplier allows higher clock frequency while parallel pipelines increases number of data blocks processed in each clock cycle. This design has suboptimal flow control because half of clock cycles multiplier does not perform any useful work.

Two devices proposed in this paper have better flow control. Each stage of multiplier pipeline is fully used in each clock cycle if there are enough incoming data blocks. That gives twice as much data blocks per clock cycle.

I. Kalistru is with Infotecs JSC, Moscow, Russia.
This work is a part of a research project of Infotecs JSC.

II. Method of calculating GHASH

X_l can be calculated using L parallel threads of calculation if $l \ge L$. Equation for X_l can be rewritten as

$$X_l = ((\ldots(S_0H \oplus S_1)H \oplus \ldots)H \oplus S_l)H$$

After expansion of this formula we can make L groups of products collecting all products containing S_{iL+j} with the same j in one group, $0 \le j < L$. After applying Horners method to each group

$$\begin{aligned} X_l =& (((\ldots)H^L \oplus S_{l-L})H^L \oplus S_l)H^1 \\ & \oplus (((\ldots)H^L \oplus S_{l-L-1})H^L \oplus S_{l-1})H^2 \\ & \oplus (((\ldots)H^L \oplus S_{l-L-2})H^L \oplus S_{l-2})H^3 \\ & \vdots \\ & \oplus (((\ldots)H^L \oplus S_{l-L-(L-1)})H^L \oplus S_{l-(L-1)})H^L. \end{aligned} \quad (1)$$

Or

$$X_l = \sum_{i=0}^{L-1} A_i \quad (2)$$

where

$$A_i = (((\ldots)H^L \oplus S_{l-L-i})H^L \oplus S_{l-i})H^{i+1}, \quad (3)$$
$$0 \le i < L$$

We call A_i a partial sum.

The equations (2) (3) clearly demonstrate the main idea used to improve speed - calculate L partial sums independently by adding succeeding blocks with step L to the previous result and multiplying it by H^L. The only exceptions are the last multiplications of each partial sum A_i, where the result of previous addition must be multiplied by H^{i+1}, $0 \ge i \ge L-1$.

To get the GHASH value, all partial sums must be added after they are calculated.

$$\text{GHASH} = \sum_{i=0}^{L-1} A_i$$

The method of calculating GHASH value through calculation of partial sums can be used to increase number of

calculation modules or to increase maximum clock frequency of the device using pipelining. In the next two sections of the paper we discuss a device which takes advantage of increased clock frequency and a modified device which additionally uses several pipelines to calculate GHASH.

III. Single-pipeline implementation

Only addition in GF(2^k) and multiplication in GF(2^k) are needed for GHASH calculation. It is reasonable to optimize multiplication in GF(2^k) because addition in GF(2^k) corresponds to simple bitwise XOR operation. The multiplication module can be implemented with several pipeline stages. It reduces the critical path of the circuit, and thus clocking frequency of the circuit can be increased. This technique allows frequency increase up to C times with critical path split into C parts. The multiplication module with C pipeline stages takes C clock cycles to calculate each product, but as it is possible to load a new pair of values to the pipeline in each clock cycle, and to have C products calculated simultaneously on different pipeline stages, there is no loss of performance on per cycle basis. This fact together with clocking frequency increased by a factor of C gives overall increase of performance by a factor of C.

The multiplication module with C pipeline stages can be used to calculate $L \geq C$ partial sums of GHASH. The new block of the frame is added to an output of the multiplication module and fed back to the first input of the multiplication module with an appropriate power of H sent to the second input of the multiplication module. The feedback circuit and the adder can take additional B cycles, so the full feedback loop has $L = C + B$ cycles. With L stages of calculation in complete feedback loop it is possible to calculate L partial sums simultaneously.

The device for GHASH calculation consists of pre-processing module and processing module.

Pre-processing module is used to associate a number to each block of a data frame. These numbers of the blocks are used in processing module to choose precalculated H^i, $1 \leq i \leq L$ and to distinguish blocks of partial sums from intermediate values after multiplication module. We call these numbers labels. Details on how to distinguish the partial sums from the intermediate results are given later in the description of accumulation module.

The labels are assigned to the blocks from the end of each frame starting from 0 up to L. The last block of each frame is labeled by 0, the block before it is labeled by 1 and so on until label L. All other blocks are labeled L. To perform this function pre-processing module contains a FIFO and writes each incoming block of data to the FIFO. Pre-processing module reads a block from the FIFO if there is L blocks of data in the FIFO, or if there is the last block of at least one frame in the FIFO. Each time pre-processing module reads a block of data from the FIFO it associates a label to it according to the following rule: it associates label L to the block being read if there is no last blocks of frames in the FIFO, and it associates $n + 1$ to the block being read if there is the last block of at least one frame in the FIFO. Here n is a number of blocks contained in the FIFO between block being read and the last block of the frame. It is possible to have more than one frame in the FIFO if there are frames shorter than L blocks. The last block of the frame which is closer to the output of the FIFO is used to calculate n in this case.

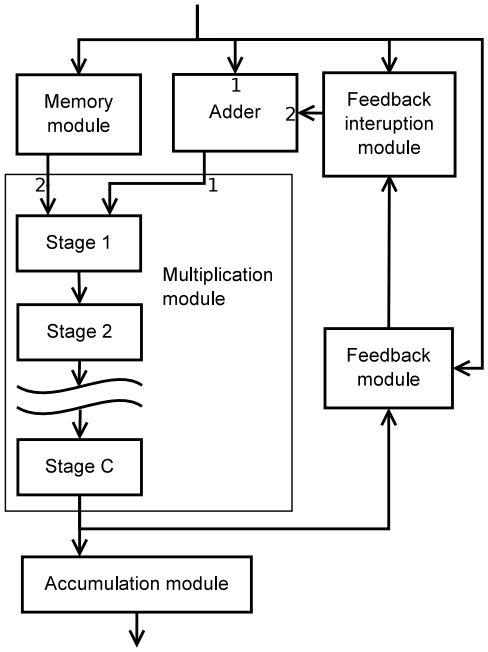

Fig. 1. Structure of processing module.

The pre-processing module numerates blocks of frames without collecting whole frames. It collects no more than L blocks of each frame and it has latency no more than L clock cycles.

Numerated blocks of data are passed to processing module.

Processing module is shown on Fig. 1. It contains memory module, adder, multiplication module, feedback module, feedback interruption module and accumulation module.

Memory module contains precalculated powers H^i, $1 \leq i \leq L$ of a key polynomial H. Memory module contains H^i at the address location $i - 1$ for $1 \leq i \leq L$. It also contains H^L at the address location L of memory module. The label of each incoming block of data to processing module is treated as an address for memory module and corresponding H^i is extracted from memory module. Extracted H^i is sent to the second input of multiplication module.

A block of data coming to processing module is sent to the first input of adder. The second input of adder receives the result of the previous iteration of Horner scheme from the output of feedback interruption module in the same clock cycle. Addition is performed in GF(2^k). The resulting block is labeled with the label of the block from the first input of adder and is sent to the first input of multiplication module.

Latency of memory module is matched with latency of adder in order to have the result from adder and corresponding H^i from memory module at inputs of multiplication module in the same clock cycle.

Multiplication module has several pipeline stages and performs multiplication in GF(2^k) (with reduction). It can be

designed in different ways. For example, it can be designed using multi-step Karatsuba-Ofman algorithm [6] with pipeline registers after each step of the algorithm.

Multiplication module produces a block of data from two blocks taken from its two inputs and labels the resulting block of data with the label of the block taken from the first input of the module. The resulting block is sent to feedback module and accumulation module.

Feedback module is needed if stream of data blocks coming to the device is not continuous. For example, as for each frame of data there is an additional block of data containing lengths of AAD and ED fields to be processed. To keep up with incoming data, a device performing GHASH computation needs to operate on a frequency which is high enough to process this additional block of data for each frame. The worst case scenario is a stream of frames of minimal possible lengths. If the device's operating frequency is high enough to be able to process short data frames and the device does not accumulate the whole frame before its processing, then for longer frames it's possible to have clock cycles when there is still no next block of the frame to process. Data blocks from multiplication module need to be stored for later use when there is a clock cycle with no next data block coming from pre-processing module.

Feedback module contains a FIFO register and each incoming block is written to the FIFO. A data block is read from the FIFO to the input of feedback interruption module if there is incoming block of data to processing module. The block of data from feedback module must come to adder in the same cycle as data block which caused the block from feedback module to be read from the FIFO of feedback module. So, additional registers must be added before first input of adder if there is additional latency in feedback interruption module.

Feedback interruption module counts blocks of data incoming to processing module and outputs a block of data containing 0 in each bit of the block for the first L blocks of each incoming frame. For other blocks incoming to processing module feedback interruption module copies blocks of data from its input. Feedback interruption module is needed to prevent irrelevant data blocks from adding to the first L blocks of each frame.

With L pipeline stages in the full feedback loop (adder - multiplication module - feedback module - feedback interruption module - adder) and with L threads of calculation distributed among different pipeline stages of the loop a result of previous iteration of Horner scheme comes to the second input of adder just in time to be added to the next block of data of the same partial sum, coming to the first input of adder from pre-calculation module.

Accumulation module is used to add L partial sums to get the final result. The module has a register for one block of data, which is used to store intermediate results. As it can be seen from the description of adder and multiplication module, each data block coming to accumulation module has a label associated with it. This label is equal to a label of the last data block used to calculate the block coming to accumulation module. If this label is less than L then the block is the value of a partial sum. For each incoming block of data with an associated label less than L, accumulation module adds content of the incoming block to the register. After addition of a block with associated label 0 (the last partial sum) the module sends content of the register to the output and resets the register to 0. Output of accumulation block is a GHASH value.

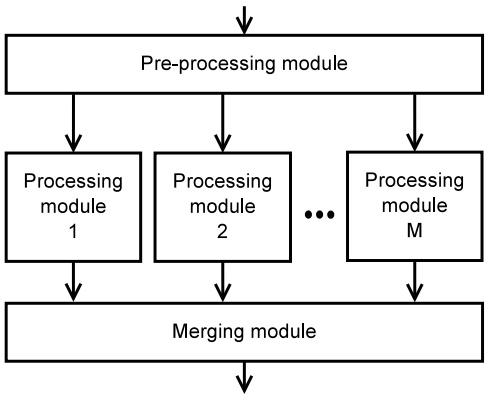

Fig. 2. Multi-pipeline implementation.

The device presented here does not require that number of blocks in frames is a multiple of L. Number of blocks also does not have to be bigger than L. There is also no need to wait until the pipeline of the device is fully emptied before loading data blocks of the next data frame. The first block of the next data frame can be loaded to the device in the clock cycle after the cycle with the last block of the previous frame. As a result, multiplier's pipeline is fully loaded all the time if there are enough incoming data blocks.

IV. Multi-pipeline implementation

Decomposition of a module on several stages of computation gives good results only up to some number of pipeline stages. After that number of stages, factors other than simple amount of combinatorial logic between two flip-flops become significant and prevent further increase of maximum clocking frequency. Techniques other than spreading workload between different pipeline stages of the same multiplication block are needed to overcome this limitation. In this section of the article a device that uses several pipelines to calculate one GHASH value is discussed. Each of the pipelines of the device uses multiple pipeline stages to calculate different partial sums as it has been discussed earlier.

With M parallel pipelines, each having L pipeline stages in the feedback loop, calculation process of GHASH function is split into ML calculations of ML partial sums.

Multi-pipeline implementation is shown on Fig. 2. It consists of pre-processing module, M processing modules and merging module.

Pre-processing module splits the incoming stream of data blocks between M lanes numbered 0 to $M-1$ in a following manner. If one block of data of the frame is sent to the lane n then the next block is sent to the lane $n+1$ and so on in a round-robin principle until the end of the frame. Each clock cycle M blocks of data are distributed among M lanes. Blocks of the next frame cannot be distributed among the lanes in the

same cycle with blocks of the previous frame. If the last block of the frame happens to be in the lane $s < M - 1$ then lanes from $s + 1$ to $M - 1$ are filled with blocks which are marked as invalid.

Pre-processing module is also associate a label to each block of the data frame similar to pre-processing module of the single-pipeline implementation, discussed in the previous section of the paper. For this purpose it contains M FIFO modules, and data blocks of each lane are stored in corresponding FIFOs. Pre-processing module monitors number of data blocks stored in the FIFOs and also monitors if there are last blocks of frames in FIFOs. Pre-processing module simultaneously reads a block of data from each FIFO if there is at least one last block of some frame in any of the FIFOs. In this case it assigns to each of M blocks being read the label q

$$q = \begin{cases} x & \text{if } x < ML \\ ML & \text{if } x \geq ML \end{cases},$$

where x is equal to a number of the block counting from the last block of the frame and starting with 0. Pre-processing module also simultaneously reads a block of data from each FIFO if there is $L + 1$ blocks in each FIFO even if there is no last block in the FIFOs. Label ML is associated with each block of data being read from the FIFOs in this case, because absence of any last block in the FIFOs guarantees that there are more than ML blocks to the end of the frame from any block being read.

The labels will be used in processing module for selection of correct powers of H for multiplication and to distinguish blocks of partial sums from intermediate values after multiplication module. Labeled blocks from the FIFOs are sent to M outputs of the pre-processing module.

Blocks of data from pre-processing module are sent to corresponding M independent processing modules. Processing modules have some differences from processing modules discussed in the previous section of the paper.

The first difference is that memory modules now contain precalculated powers H^i, $1 \leq i \leq LM$ at address $i - 1$. They also contain H^{LM} at address LM.

The second difference from the processing module described in the previous section of the paper is that accumulation module adds content of the incoming block to the register for each incoming block of data with associated label less than LM. Reception by accumulation module of a block with a label less than M means that there will not be more blocks of the same frame coming to this accumulation module. For example, if $M = 3$ and accumulation module of processing module 1 receives a block with label 2, that means that the next block with label 1 is processed by processing module 2 and the last block of a frame with label 0 will be processed by processing module 0. It can be seen that the block with label 2 is the last block of the frame processed by processing module 1. This distribution of blocks is shown on TABLE I.

After addition of a block with a label less than M the module sends content of the register to the output and resets the register to 0. The resulting value is a sum of L partial sums. This value is sent to merging module.

TABLE I
EXAMPLE OF DISTRIBUTION OF BLOCKS AMONG PROCESSING MODULES

Module name	cycle 1	cycle 2	cycle 3
Processing module 0	label=3	label=3	label=0
Processing module 1	label=3	label=2	invalid
Processing module 2	label=3	label=1	invalid

TABLE II
PROCESSING MODULE RESOURCE CONSUMPTION

Module name	×	LUT types			Total LUT	Reg.
		LUT	SRL	MLUT		
Mult. module	1	3028	4	0	3032	10324
Memory module	2	0	0	160	160	128
Others	–	397	0	160	557	465
Total	–	3425	4	480	3909	11045

Merging module receives blocks from all M processing modules and adds them together to get the final result of the computation process - the GHASH value.

The device presented here does not require that number of blocks in frames is a multiple of M. There is also no need to wait until the pipeline of the device is fully emptied before loading data blocks of the next data frame. The first block of the next data frame can be loaded to the device in the next clock cycle after the last block of the previous frame. The device is designed to have fully loaded multiplier except for the border of two frames when there can be a clock cycle when not every pipeline is able to receive a new data block.

The device described here has been implemented on Virtex UltraScale+ FPGA with $C = 3$ stages of multiplication pipeline, $L = 6$, and only $M = 2$ parallel pipelines, achieving maximum speed 102.4 Gbit/s (400 MHz with block size 128 bit). The device contains two memory modules in each processing module to be able to load new H^i while processing data with old ones. TABLE II shows resource consumption by processing module with 2 memory modules for on-the-fly key replacement. TABLE III shows resource consumption by the device.

TABLE III
RESOURCE CONSUMPTION

Module name	×	LUT types			Total LUT	Reg.
		LUT	SRL	MLUT		
Pre-proc. mod.	1	85	740	0	825	467
Proc. module	2	3425	4	480	3909	11045
Merging module	1	41	12	0	53	471
Total	–	6976	760	960	8696	23028

V. CONCLUSION

Accumulation of multiple frames of data can significantly increase latency of a device because of time needed to accumulate several frames. The proposed devices allow fast low latency GHASH calculation without such accumulation. The devices efficiently utilize multiplication module's pipeline by using all the stages of the pipeline in each clock cycle, except for the boundaries of frames in case of multi-pipeline implementation.

The first proposed device allows efficient use of multiplication module with several pipeline stages. The device has improved performance because pipelined multiplication module can work at a higher frequency. Another important feature of the device is that the first block of the next frame can be sent to the device in the next clock cycle after the last block of the previous frame. This prevents performance losses in case of short data frames. The device also can be used when there can be clock cycles without incoming data blocks, which can be useful if the source of data frames can produce such idle clock cycles and a target latency of the device does not allow full frame accumulation before the GHASH calculation module. The device is designed to have multiplier's pipeline fully loaded all the time if there are enough incoming data blocks. Therefore, performance is improved.

If further increase in the number of pipeline stages gives little improvement in speed, the second device can be used to achieve higher speeds. It uses several modified processing modules of the first device to form parallel pipelines which are used to calculate GHASH values of frames. The second device with M processing modules has up to M times higher performance than the first device and don't require accumulation of several frames of data.

REFERENCES

[1] IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Security - Amendment 1: Galois Counter Mode-Advanced Encryption Standard-256 (GCM-AES-256) Cipher Suite.

[2] RFC 4106, IETF Request for Comments, The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP), J. Viega, D. McGrew, June 2005.

[3] RFC 5288, IETF Request for Comments, "AES Galois Counter Mode (GCM) Cipher Suites for TLS" J. Salowey, A. Choudhury, and D. McGrew, August 2008.

[4] "NIST SP 800-3 8D", Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, Nov 2007.

[5] Abdellatif, Karim M. & Chotin-Avot, Roselyne & Mehrez, Habib. (2012). Efficient Parallel-Pipelined GHASH for Message Authentication. 2012 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2012. 10.1109/ReConFig.2012.6416742.

[6] A. Karatsuba and Yu. Ofman (1962). "Multiplication of Many-Digital Numbers by Automatic Computers". Proceedings of the USSR Academy of Sciences. 145: 293-294. Translation in the academic journal Physics - Doklady, 7 (1963), pp. 595-596.

Ilia Kalistru was born in Michurinsk, Tambov Oblast, Russia, in 1985. He received the B.S. and M.S. degrees in radiophysics from Voronezh State University, Voronezh, Russia, in 2006 and 2008, respectively.

He is an FPGA design engineer since 2009. Since May 2014, he has been an FPGA design engineer at Infotecs JSC. His current research interests include high speed low latency data processing, high speed interfaces, practical aspects of digital electronic circuit design and design of cryptographic hardware.

Mr. Kalistru is a counselor of Russian Academy of Natural History and a member of International Association of Engineers (IAENG)