A Parallel Permutation Multiplier for a PGM Crypto-chip

Tamás Horváth[1], Spyros S. Magliveras[2] and Tran van Trung[1]

[1] Institute for Experimental Mathematics, University of Essen, Ellernstrasse 29, 45326 Essen, Germany
[2] Department of Computer Science and Engineering, University of Nebraska, Lincoln NE, 68588-0115, U.S.A.

Abstract. A symmetric key cryptosystem, called PGM, based on *logarithmic signatures* for finite permutation groups was invented by S. Magliveras in the late 1970's. PGM is intended to be used in cryptosystems with high data rates. This requires exploitation of the potential parallelism in composition of permutations. As a first step towards a full VLSI implementation, a parallel multiplier has been designed and implemented on an FPGA (Field Programmable Gate Array) chip. The chip works as a co-processor in a DSP system. This paper explains the principles of the architecture, reports about implementation details and concludes by giving an estimate of the expected performance in VLSI.

1 Introduction

A symmetric key cryptosystem PGM based on *logarithmic signatures* for finite permutation groups was invented by by S. Magliveras in the late 1970's. The system was described in [1], and its statistical and algebraic properties were studied in [2], [3], [4]. Recent significant results have been obtained on closely related material by S.A. Vanstone and by M. Qu [5]. Here we include only a short description of PGM.

Let G be a finite permutation group of degree n. A *logarithmic signature* for G is an ordered collection $\alpha = \{A_i : i = 1, \ldots, s\}$ of ordered subsets, $A_i = \{a_{i,0}, \ldots, a_{i,r_i-1}\}$ of G, such that each element $g \in G$ can be expressed uniquely as a product of the form

$$g = q_s \cdot q_{s-1} \cdots q_2 \cdot q_1 \qquad q_i \in A_i \tag{1}$$

The A_i are called the *blocks* of α and the vector of block lengths $(r_1, \ldots, r_s)$ is called the *type* of α. A logarithmic signature is called *tame* if the factorization in equation (1) can be achieved in time polynomial in the degree n of G; it is called *supertame* if the factorization can be achieved in time $O(n^2)$. A logarithmic signature is called *wild* if it is not tame. In [4] the authors describe how a logarithmic signature α induces an efficiently computable bijection $\hat{\alpha} : Z_{|G|} \to G$. The inverse of $\hat{\alpha}$ is efficiently computable only if α is tame. *Basic* system PGM is described as follows: For a given pair of tame logarithmic signatures, α, β, the *encryption* transformation $E_{\alpha,\beta}$ is the mapping $E_{\alpha,\beta} = \hat{\alpha}\hat{\beta}^{-1} : Z_{|G|} \to Z_{|G|}$.

Y.G. Desmedt (Ed.): Advances in Cryptology - CRYPTO '94, LNCS 839, pp. 108-113, 1994.

The corresponding *decryption* transformation is obtained by reversing the order of the pair of logarithmic signatures, i.e. $D_{\alpha,\beta} = E^{-1}_{\alpha,\beta} = E_{\beta,\alpha} = \hat{\beta}\hat{\alpha}^{-1}$.

To effect the fastest possible PGM encryption and decryption operations, one must compute efficiently products of permutations as in equation (1). Unlike multiplication of integers, composition of two permutations is inherently parallelizable. Hence, we can achieve fast computation of $\hat{\alpha}$ and its inverse by designing a permutation multiplier which takes advantage of this property of permutation composition. In this paper we describe a design for such a permutation multiplier, as a first step towards a full VLSI implementation of PGM.

2 Principles of multiplication in parallel

For easy understanding, we shall explain the principles by means of a simple example. We consider permutations of degree 4, and represent them in *cartesian* form, $\pi = [\pi(0), \pi(1), \pi(2), \pi(3)]$. This form is particularly convenient for representing permutations in hardware, where a vector register of length n is used to represent a permutation of degree n. For example, $\pi = [3, 2, 1, 0]$ is our notation for the permutation $\pi = (0\ 3)(1\ 2)$ as the product of disjoint cycles. In general, this representation needs $nlog_2 n$ bits to store a permutation of degree n. Throughout the example, we define five input operands to work with, namely: $\iota = [0, 1, 2, 3]$ (the identity permutation), $\alpha = [2, 3, 0, 1]$, $\beta = [1, 3, 2, 0]$, $\gamma = [3, 2, 0, 1]$ and $\delta = [1, 2, 3, 0]$.

The multiplication unit is in essence a crossbar switching network. A 4x4 switching matrix is depicted in Figure 1. The matrix has three input ports, labeled A, B and C respectively, and one output port named Q. Ports B and C are connected to the vertical lines in the matrix, whereas A and Q to the horizontal lines. At the cross-points of vertical and horizontal lines reside the *switching cells*, each consisting of a *transfer gate* and *cell-logic*. The cell-logic controls the transfer gate. If the gate is open, which is denoted by a dot ($\bullet$) in the Figures, it allows the signal to propagate from the vertical line onto the corresponding horizontal line. A closed gate does not influence the signal on the horizontal line. Multiplication takes place in two phases:

(a) In the first, so-called *setup phase* the contents of A and B appear on the horizontal and vertical lines respectively. At each cross-point, the corresponding cell-logic unit compares the horizontal input to the vertical input and saves the result of the comparison: 1 in case of a match, and 0 otherwise. (See Figure 1(a))

(b) In the second, so-called *pass-through phase*, the transfer gates at cells where a match was found, open. The C operand is placed onto the vertical lines and is transferred via the open gates onto port Q as the result. (See Figure 1(b))

It is now relatively easy to see that the result Q can be expressed in terms of the other operands as $Q = (A \circ B^{-1}) \circ C$, where $\circ$ denotes *composition* of permutations. We verify the result Q when $A = \alpha = [2, 3, 0, 1], B = \beta = [1, 3, 2, 0]$ and $C = \gamma = [3, 2, 0, 1]$ as in Figure 1: $([2, 3, 0, 1] \circ [1, 3, 2, 0]^{-1}) \circ [3, 2, 0, 1] = [2, 3, 0, 1] \circ [3, 0, 2, 1] \circ [3, 2, 0, 1] = [0, 2, 1, 3]$.

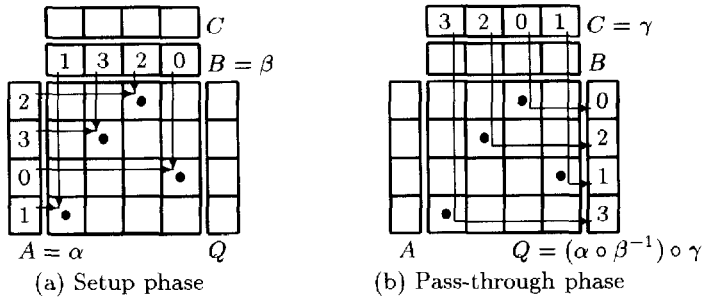

Fig. 1. Principle of multiplication in the switching matrix.

We remark here that the partial product $\pi = \alpha \circ \beta^{-1}$ is implicitly stored in the state of the transfer gates, and can be retrieved by passing $C = \iota$ through the matrix. Furthermore, it is possible to compute several products with the same first operand π, without setting up the matrix again. This kind of operation we call *continuous mode*. By dedicating separate lines to A, B, C and Q respectively, it becomes possible to overlap in time the pass-through phase of a multiplication and the setup phase of the next one. This two-stage *pipelining* is shown in Figure 2. The state of the gates is always changed at the end of the phases, thus pass-through operations can take place using the previous setup.

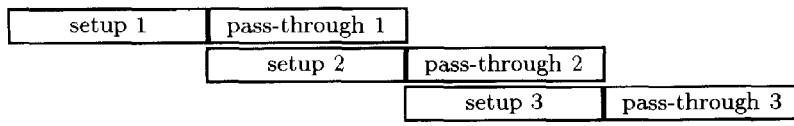

Fig. 2. Pipelining setup and pass-through stages.

A particular case of the pipelined operation is used in the implementation. The contents of port Q are fed back as input to port A, within the same phase. By letting $B = \iota$ constantly, the matrix evaluates products of the form $\pi_1 \circ \pi_2 \ldots \circ \pi_n$ in exactly n cycles, i.e. without losing cycles, by merely loading back partial products as input operands. This mode of operation is called *feedback mode*. The computation of $\alpha \circ \gamma \circ \delta = [2,3,0,1] \circ [3,2,0,1] \circ [1,2,3,0] = [1,2,0,3]$ can be followed in Figure 3.

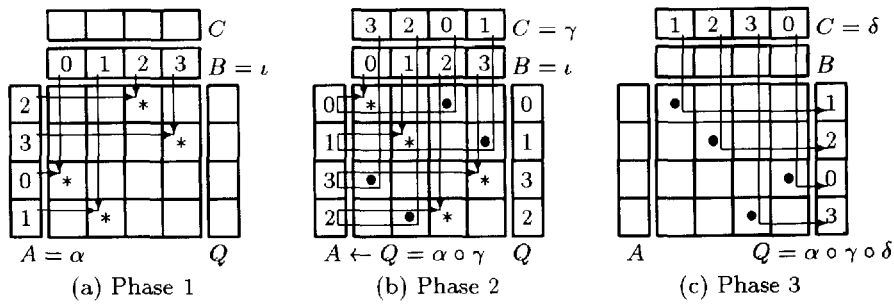

Fig. 3. Feeding back Q to A to compute $\alpha \circ \gamma \circ \delta$

3 Implementation details

As a first step towards a VLSI implementation of PGM, a hybrid hardware-software prototype has been developed based on a Texas Instruments 320C30 DSP processor. Multiplication of permutations is effected in the permutation co-processor chip, which is connected to the DSP system via a 16 bit peripheral bus, called DSPLINK. The DSP accesses the co-processor through I/O instructions. The co-processor is an XC3190 FPGA (*Field Programmable Gate Array*), a product of the Xilinx Corporation. The FPGA is a perfect prototyping tool, in view of the flexibility it affords for design changes. However, the achievable complexity is rather low, only a few thousand gate equivalents. This constraint limits the degree n of permutations that are processed on the chip to $n = 16$.

In order to be able to carry out one setup or pass-through operation in each cycle, the operands have to be led through the crossbar network in parallel, i.e. needing $log_2 n$ lines per operand. For practical applications n should be at least 32, requiring thus at least $5.2^5 = 160$ lines. Although a fully parallel implementation may still be feasible on a VLSI chip, we follow a different approach. The vectors of first, second, etc. bits of the n elements in the permutation are sent through the crossbar serially, in $log_2 n$ cycles. This principle reduces dramatically the total number of lines needed, the complexity of the cells, and hence the overall chip area. Due to shorter lines, propagation delays shorten considerably, too. We estimate the performance of a serialized multiplier to be about 50% that of a fully parallel one. This seems to be a good trade-off between price and speed.

Let us now take a closer look at the FPGA multiplier. The circuitry belonging to one cell is depicted in Figure 4. As a convention, the vector of least significant bits (LSBs) is processed first, followed by the other bit layers in order of significance. The cells function as follows:

- Essentially, the XOR gate compares the corresponding bits of the operands A and B, received from the neighboring horizontal and, respective vertical lines. The result of a bit comparison is AND-ed with the accumulated result of previous comparisons, and is reflected by the state of ACCUFF. The output of the AND function becomes the new accumulated result, and is written into ACCUFF at the end of the cycle due to a low-high transition on the global ACCU clock net.
- During the first cycle of the setup phase a global signal, called INIT, is activated. This makes the cells ignore the accumulated result, and simply enter the output of the XOR gate into ACCUFF.
- At the end of the last cycle a transition occurs in FIRE, the second global clock net, which causes the final result to be entered into FIREFF. The output of this flip-flop controls GATE, a tri-state buffer, thus a new setup also comes into effect at this moment.

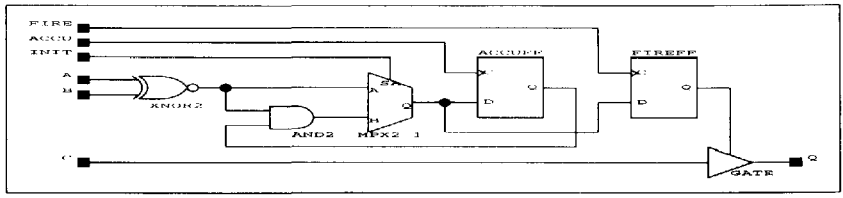

Fig. 4. The logic scheme of a cell.

The data-path of the permutation chip, reduced to 4 bit vector length, can be seen in Figure 5. Ports A and Q of the multiplier array are unified as port AQ on the left edge. Similarly, ports B and C are fused to form port BC on the top edge. The external pins of AQ and BC are also connected together on the embedding card to form one data bus for connecting to DSPLINK. The identity operand I is *hard-wired* on the chip in units ICODE_n All signals controlling the ports and cells, are generated in unit CTRLOGIC.

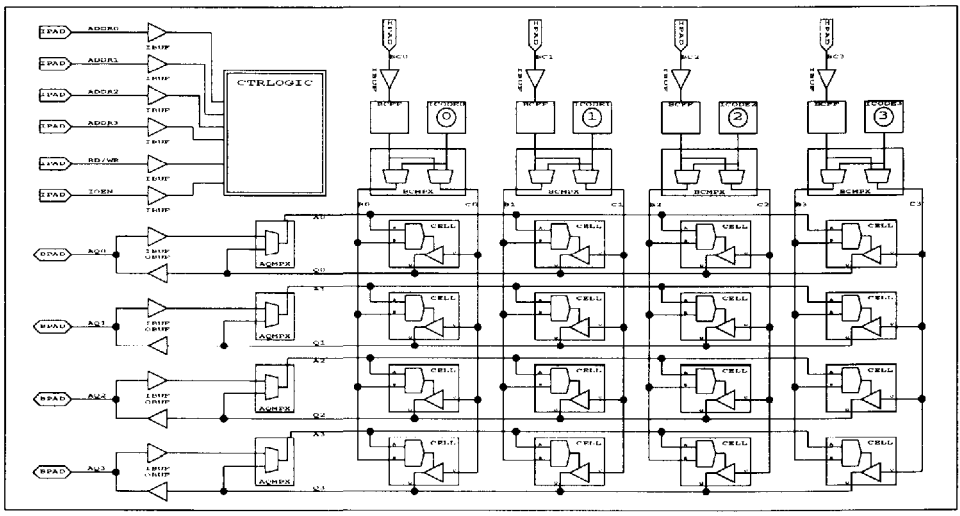

Fig. 5. The data path of the chip, reduced to vector length 4.

The 4 bit address bus of DSPLINK presents the instruction code to the chip, thus instruction and data are transferred at the same time. An instruction set of 6 elements has been defined to control the assembly. Because of space limitations we can not go into their semantics here.

4 Conclusions

A permutation multiplier chip has been developed, verified by simulation, attached to a DSP system and successfully tested by means of a simple DSP program. The processing speed is satisfactory, 100ns for one cycle.

In our implementation the degree of processed permutations was set to $n = 16$.

This is of course too small for practical applications. Nevertheless, we consider this prototyping work an important step towards a full VLSI implementation of PGM. Our multiplier architecture can be easily extended to larger n, and be quickly transferred to larger scale technology.

For future work we plan to complete the DSP implementation so as to gain more insight into the actual processing and storage requirements of the PGM algorithm. Afterwards we intend to augment the permutation matrix with other hardware units to embrace the entire algorithm with fast, special-purpose hardware.

References

1. S. S. Magliveras, A cryptosystem from logarithmic signatures of finite groups, In *Proceedings of the 29'th Midwest Symposium on Circuits and Systems*, Elsevier Publishing Company (1986), pp 972–975.
2. S. S. Magliveras and N. D. Memon, The Linear Complexity Profile of Cryptosystem PGM, *Congressus Numerantium*, Utilitas Mathematica, **72** (1989), pp 51–60.
3. S. S. Magliveras, N. D. Memon and K.C. Tam, Complexity tests for cryptosystem PGM, *Congressus Numerantium*, Utilitas Mathematica, **79** (1990), pp 61–68.
4. S. S. Magliveras and N. D. Memon, Algebraic Properties of Cryptosystem PGM, in *Journal of Cryptology*, **5** (1992), pp 167 –183.
5. M. Qu and S. A. Vanstone, Factorizations of elementary Abelian p-groups and their cryptographic significance, to appear in J. of Cryptology.