IMPLEMENTATION OF MONTGOMERY MULTIPLICATION IN A COARSE-GRAINEDED RECONFIGURABLE ARCHITECTURE

Arnaldo Azevedo, Rodrigo Soares, Ivan Saraiva Silva
Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
[arnaldo, rodsoares]@lcc.ufrn.br, ivan@dimap.ufrn.br

ABSTRACT

The Montgomery Multiplication is largely used in cryptosistens, like RSA and ECC, which needs efficient implementations. This paper presents an implementation of the Montgomery Multiplication in a coarse-grained reconfigurable architecture, the X4CP32.

1. INTRODUCTION

Public-key cryptosystems, like RSA and Elliptic Curves Cryptography (ECC), make massive use of modular arithmetic, mainly multiplication and exponentiation. The Montgomery Modular Multiplication [1], is an efficient and flexible algorithm, that can be used to support fast cryptosystem implementations.

The Montgomery algorithm computes: A x B x r^{-1} mod M. The constraints of this algorithm are: A and B ought be smaller than M and r ought be relatively prime to M.

2. ALGORITHM USED

The Montgomery algorithm works for any base, so it is possible to take advantage of the 32 bits word of the X4CP32 architecture to have a better implementation. Equations 1 presents the word representation of 'A', 'B' and 'M'.

$$A = \sum_{i=0}^{k-1} a_i \times \beta^i;\ B = \sum_{i=0}^{k-1} b_i \times \beta^i;\ M = \sum_{i=0}^{k-1} m_i \times \beta^i$$

Equations 1 – A, B and M representation

Where: k = number of words, t = size in bits of the word and $\beta = 2^t$.

The word-level Montgomery Modular Multiplication algorithm is presented in Equations 2**:**

```
1   Int Montgomery(A,B,M, (-m0)^-1){
2           int R = 0;
3           for(int I; I <= k; I++){
4              tmp = (r0 + aI * b0) * (-m0)^-1 mod β;
5              R = R + aI * B + tmp * M;
6              R /= β; }
7   return R; }
```

Equations 2 – Algorithm

3. TARGET ARCHITECTURE

The X4CP32 [2, 3] is a coarse-grained reconfigurable architecture that consists of two hierachic levels of abstraction: RPU and Cell. Each one has its own mechanisms of progamability and configurability.

The Reconfigurable and Programming Unit (RPU) is the main entity in the X4CP32 architecture. The RPU is responsible for seeking instructions in the main memory and executing them. The Execution Mode defines the behavior of the RPU. There are two Execution Modes: the Programming Execution Mode and the Reconfigurable Execution Mode.

In Programming Execution Mode the RPU acts as a parallel processor. The top left Cell assumes the Processor Operation Mode. The other Cells assume the Dynamic ALU Operation Mode, to execute the instructions sent from the top left Cell. In the Reconfigurable Execution Mode the RPU configures each Cell inputs, operations, outputs and routings, this way building a systolic data path, just like the usual reconfigurable architectures.

4. IMPLEMENTATION

For exploiting the processor's word length, 't' is set to 16, since the product of two 16 bits number is a 32 bits number. 'A', 'B' and 'M' are represented as a 64 positions array each.

To compute the multiplication of a 1024 bits number (B and M) by others numbers (a_i and tmp), with 16 bits length, it is necessary to multiply each array (which represents the 1024 bits numbers) position by the 16 bits multiplicand, preserving the 16 lower bits and sum the 16 higher others (carry in) to the next position.

Those calculi don't overflow the word capacity. The maximum product of two 16 bits numbers is 0xFFFE0001, with the maximum carry of 0xFFFE and a maximum result of 0xFFFEFFFF.

When the procesing of a_i*B and tmp*M is finished, it's still necessary to perform the sum of three 1024 bits numbers (line 7 in Equations 2). The division of R by β (line 8) is done by shifting the array one position down, when saving it to the memory.

The gray boxes indicate that the RPU is in Programming Execution Mode. White boxes are RPU's in Reconfigurable Execution Mode. Each circle in the

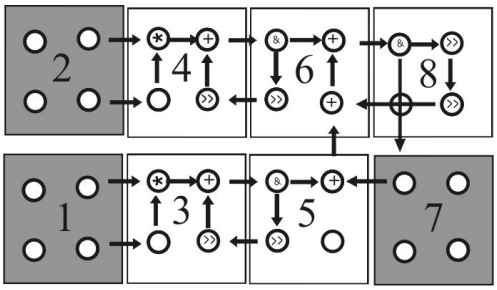

Figure 1

RPU indicates a Cell, and the symbol within the circle is the operation executed by the Cells. When operations are performed in RPU's in Programming Execution Mode the symbol within the circles represent the main (but not the only) operation of that Cell. When no operation is indicated in any cell of the RPU, it performs I/O operation only.

The RPU '1' calculates line 6 of Equations 2 first, in a common processor fashion. The systolic way would consume many Cells and the performance doesn't make for it. This computation consumes 51 clock cycles.

RPU '1' just sends a_i and B[x] as inputs of RPU's '3'. The same goes to RPU '2', which sends *tmp* and M[x] to RPU '4'. The output RPU '7', only sends a position of R to be added and receives a new R position to be stored in.

Reconfigurable Execution Mode RPU's (3 to 6 and 8) implement the carrying operation. These RPU's sum the result sent by a Programming Execution Mode RPU with the shifted right high part of the previous result. To the result of this sum, an *and* operation with 0000FFFF is applied to save the 16 lower bits. To the same sum result a 16 positions left shift (8 in each shift Cell) is applied to implement the 'carry in' to be added to the next result sent by the Programming Execution Mode RPU.

5. RESULTS AND COMPARISON WITH OTHERS IMPLEMENTATIONS

An ASIC systolic design, synthesized in FPGA, presented in [4] was chosen for comparison. It presents a bit-wise version of the algorithm. Another chosen architecture is an ASIC presented in [5]. A general-purpose processor (GPP) C implementation was developed using the same word level algorithm and ran in a Duron 950Mhz with 88MB RAM (measured with the Unix's *time* tool). Table I presents the results.

Table I – Comparison

Architecture	Clock (Mhz)	Time to compute A x B mod M (ms)
GPP	950	0.4
FPGA [4]	7.4	0.4
ASIC [5]	41	0.04
X4CP32	65.9	0.7

The results shown in Table I gives the idea of the flexibility/performance tradeoff. Both ASIC and FPGA have better performance, as it was expected, because of heavy flexibility limitation. Their implementation extracts the best possible results for one application only, by exploiting its specific properties. The ASIC implementation results are 20 times better than X4CP32's ones, however, it is important to notice that they are two different technologies with two different approaches (ASIC/Silicon and General-Purpose/FPGA). The FPGA results are closer to X4CP32's ones. That's due the similar technologies used. The difference in clocks frequencies is a matter of project focus and cannot taken into account, because once again the comparison is between two different approaches (ASIC and General-Purpose). The comparison with the GPP is the most appropriated to analyze X4CP32 performance, because they are both general-purpose processors. They have similar results despite the huge clock frequency gap. The silicon implementation of the GPP gives it a good advantage over the FPGA X4CP32 prototype. This time the comparison is between two similar approaches implementations but two different technologies (Silicon and FPGA).

The efficiency of the X4CP32 in this application can increase by replicating the presented implementations to take advantage of the parallelism of the processor.

6. CONCLUSIONS

This paper presented an implementation of the Montgomery Multiplication and compared it with other implementations. The results show that general-purpose reconfigurable architectures can be very competitive in this area of applications.

7. REFERENCES

[1] Montgomery, Peter L. "Modular Multiplication Without Trial Division", Mathematics of Computation, Vol 44. pp 519-521. 1995.

[2] R. Soares, A. Pereira, I. Saraiva, "X4CP32: a Programmable Multi-level Reconfigurable Microprocessor", *Proceedings of Students Forum on Microeletronics 02*, SBC, Porto Alegre, pp. 69-73, 2002.

[3] A. Pereira, R. Soares, I. Saraiva, "Implementação da DCT 2D em arquiteturas reconfiguráveis utilizando a X4CP32", *Proceedings of Iberchip 02*, Havana, Cuba, 2002.

[4] N. Nedjah, L. M. Mourelle. "Two Hardware Implementations for Montgomery Modular Multiplication: Sequential versus Parallel". Proceedings of 15th Symposium on integrated circuits and system design SBCCI, Brasil. 2002.

[5] G. Gaubatz, "Versatile Montgomery Multiplier Architectures", Master' s Thesis, Worcester Polytechnic Institute, May 2002.