Implementation of Elliptic Curve Cryptosystems over GF(2^n) in Optimal Normal Basis on a Reconfigurable Computer

Sashisu Bajracharya[1], Chang Shu[1], Kris Gaj[1], Tarek El-Ghazawi[2]

[1] ECE Department, George Mason University,
4400 University Drive, Fairfax, VA 22030, U.S.A.
{sbajrach, cshu, kgaj}@gmu.edu
[2] ECE Department, The George Washington University,
801 22nd Street NW, Washington, D.C., U.S.A.
tarek@gwu.edu

Abstract. Reconfigurable Computers are general-purpose high-end computers based on a hybrid architecture and close system-level integration of traditional microprocessors and Field Programmable Gate Arrays (FPGAs). In this paper, we present an application of reconfigurable computers to developing a low-latency implementation of Elliptic Curve Cryptosystems, an emerging class of public key cryptosystems used in secure Internet protocols, such as IPSec. An issue of partitioning the description between C and VHDL, and the associated trade-offs are studied in detail. End-to-end speed-ups in the range of 895 to 1300 compared to the pure microprocessor execution time are demonstrated.

1 Introduction

Reconfigurable Computers are high-end computers based on the close system-level integration of traditional microprocessors and Field Programmable Gate Arrays (FPGAs). Cryptography, and in particular public key cryptography, is particularly well suited for implementation on reconfigurable computers because of the need for computationally intensive arithmetic operations with unconventionally long operands sizes.

As a platform for our experiments we have chosen one of the first general-purpose, stand-alone reconfigurable computers available on the market, the SRC-6E [1], shown in Fig. 1. The microprocessor subsystem of SRC-6E is based on commodity PC boards. The reconfigurable subsystem, referred to as MAP, is based on three Xilinx Virtex II FPGAs, XC2V6000.

As shown in Fig. 2, each function executed on the SRC-6E reconfigurable computer can be implemented using three different approaches: 1) as a High Level Language (HLL) function running on a traditional microprocessor, 2) as an HLL function running

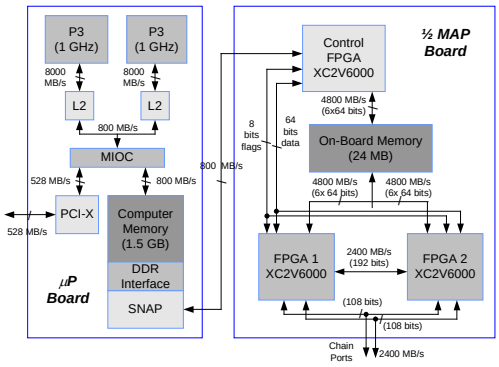

Fig. 1. Hardware architecture of the SRC-6E

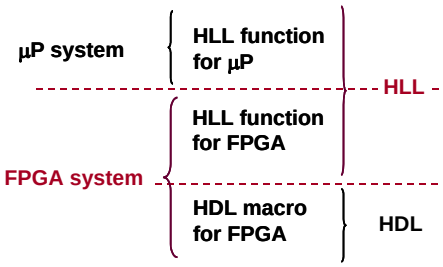

Fig. 2. Three ways of implementing a function on a reconfigurable computer

on an FPGA, and 3) as a Hardware Description Language (HDL) macro running on an FPGA.

As a result, any program developed for execution on the SRC-6E needs to be partitioned taking into account two independent boundaries, the first, between the execution on a microprocessor vs. execution on an FPGA system; and the second between the program entry in HLL vs. program entry in HDL.

2 Elliptic Curve Cryptosystems

Elliptic Curve Cryptosystems (ECCs) are a family of public key cryptosystems. The primary application of ECCs is secure key agreement and digital signature generation and verification [2]. In both of these applications the primary optimization criterion, from the implementation point of view, is the minimum latency for a single set of data (rather then the data throughput for a large set of data). The primary operation of ECCs is elliptic curve scalar multiplication (kP). In our implementation of scalar multiplication we adopted the optimized algorithm for computing scalar multiplication by Lopez and Dahab [3]. Our implementation supports elliptic curve operations over GF(2^n) with optimal basis representation for n=233, which is one of the sizes recommended by NIST [2].

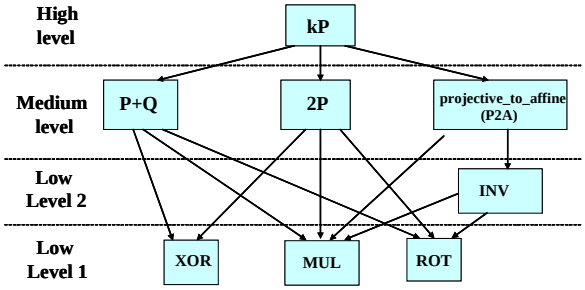

Fig. 3. Hierarchy of the ECC operations

3 Investigated partitioning schemes

A hierarchy of operations involved in an elliptic curve scalar multiplication for the case of an elliptic curve over GF(2^n) is given in Fig. 3. Four levels of operations are involved in this hierarchy: scalar multiplication (kP) at the high level (H), point addition (P+Q), point doubling (2P), and projective-to-affine conversion (P2A) at the medium level (M), inversion (INV) at the low level 2 (L2), and the GF(2^n) multiplication (MUL), squaring (rotation) (ROT), and addition (XOR) at the lowest level (L1). Functions belonging to each of these four hierarchy levels (high, medium, low 2 and low 1) can be implemented using three different implementation approaches shown in Fig. 1. In this paper, each of these approaches is characterized by a three-letter codename, such as 0HM. The meaning of these codenames is explained in Fig. 4.

We used as our reference case the complete ECC implementation running in the microprocessor and based on [4]. All other implementations ran entirely on the FPGA and were partitioned between C code that was automatically translated to VHDL and hand-coded VHDL.

4 Results

The results of the timing measurements for all investigated partitioning schemes are summarized in Table 1. The FPGA Computation Time, T_{FPGA}, includes only the time spent performing computations using User FPGAs. The End-to-End time, T_{E2E}, includes the FPGA Computation time and all overheads associated with the data and control transfers between the microprocessor board and the FPGA board.

The Total Overhead, T_{OVH}, is the difference between the End-to-End time and the FPGA Computation Time. Two specific components of the Total Overhead listed in Table 1 are DMA Data In Time, $T_{DMA\text{-}IN}$, and DMA Data Out Time, $T_{DMA\text{-}OUT}$. They represent, respectively, the time spent to transfer inputs from the Microprocessor Memory to the On-board Memory, and the time spent to transfer outputs from the On-Board Memory to the Microprocessor Memory.

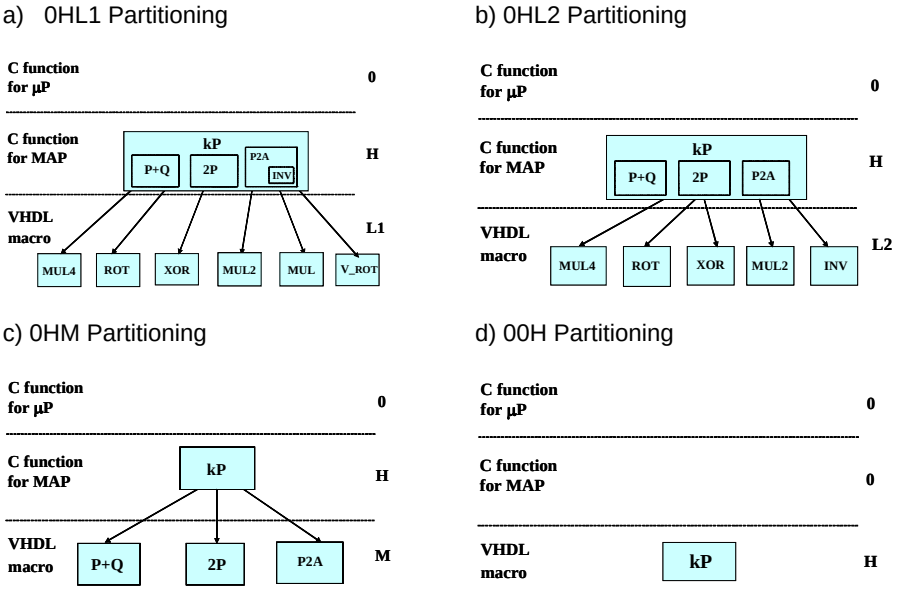

Fig. 4. Four alternative program partitioning schemes

The scheme that requires the smallest amount of hardware expertise and effort, 0HL1, is 893 times faster than software and less than 50% slower than pure VHDL macro. Implementing inversion in VHDL, in the 0HL2 scheme, does not give any significant gain in performance and only small reduction in the resource usage.

The 0HM scheme is more difficult to implement than 0HL1 and 0HL2 schemes, because of the additional operations that need to be expressed in VHDL Nevertheless, using this scheme gives substantial advantages in terms of both performance (about 45% improvement) and resource usage (e.g., reduction in the number of CLB slices by 24% compared to the 0HL1 scheme). The most difficult to implement, the 00H scheme (the entire kP operation described in VHDL) appears to have the same speed as 0HM, but it provides an additional substantial reduction in terms of the amount of required FPGA resources.

Table 1. Results of the timing measurements for several investigated partitioning schemes. Notation: SP_{SW} – speed-up vs. software, SL_{VHDL} – slow-down vs. VHDL macro.

	T_{E2E} (µs)	$T_{DMA\text{-}IN}$ (µs)	T_{FPGA} (µs)	$T_{DMA\text{-}OUT}$ (µs)	T_{OVH} (µs)	SP_{SW}	SL_{VHDL}
Soft-ware	772,519	N/A	N/A	N/A	N/A	**1**	**1,305**
0HL1	866	37	472	14	394	**893**	**1.46**
0HL2	863	37	469	14	394	**895**	**1.45**
0HM	592	37	201	12	391	**1305**	**1.00**
00H	592	39	201	17	391	**1305**	**1.00**

Table 2. Resource utilization for several investigated partitioning schemes

	% of CLB slices	CLB slices vs. 00H	% of LUTs	LUTs vs. 00H	% of FFs	FFs vs. 00H
0HL1	99	**1.68**	57	**1.30**	68	**2.61**
0HL2	92	**1.56**	52	**1.18**	62	**2.38**
0HM	75	**1.27**	48	**1.09**	39	**1.50**
00H	59	**1.00**	44	**1.00**	26	**1.00**

The current version of the MAP compiler (SRC-6E Carte 1.4.1) optimizes performance over resource utilization. As it matures the compiler should be expected to balance high performance, ease of coding, and resource utilization to yield a truly optimized logic.

5 Conclusions

While earlier publications (e.g., [5]) regarding implementations of cryptography on reconfigurable computers have already proven the capability of accomplishing a 1000x speed-up compared to the microprocessor implementations in terms of the data throughput, this is a first publication that shows a comparable speed-up for data latency.

This speed-up is even more remarkable taking into account that the selected operation has only limited amount of intrinsic parallelism, and cannot be easily sped up by multiple instantiations of the same computational unit. In spite of these constraints, a speed-up in the range of 895-1300 has been demonstrated compared to the public domain microprocessor implementation using four different algorithm partitioning approaches.

References

1. SRC Inc. Web Page, http://www.srccomp.com/
2. FIPS 186-2, Digital Signature Standard (DSS), pp. 34-39, 2000, Jan. 27, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf
3. López, J., and Dahab, R.: Fast Multiplication on Elliptic Curves over GF(2^m) without precomputation. CHES'99, LNCS 1717, (1999)
4. Rosing, M., Implementing Elliptic Curve Cryptography, Manning, 1999
5. Fidanci O. D., Poznanovic D., Gaj K., El-Ghazawi K., and Alexandridis N., "Performance and Overhead in a Hybrid Reconfigurable Computer," Reconfigurable Architecture Workshop, RAW 2003, Nice, France, Apr. 2003