A Flexible Soft IP Core for Standard Implementations of Elliptic Curve Cryptography in Hardware

Bruno F. Ferreira, Ney L. V. Calazans
Faculty of Computer Science - Pontifical Catholic University of Rio Grande do Sul - PUCRS - Porto Alegre, Brazil
bruno.ferreira@acad.pucrs.br, ney.calazans@pucrs.br

Abstract - **Elliptic curve cryptography is a rather new, efficient technology for security. However, its implementation is complex and software versions can be prohibitively slow. The main original contribution of this paper is the proposition of a highly parameterizable soft intellectual property core that implements all the operations needed to perform elliptic curve cryptography in hardware. This core supports several standardized elliptic curves. Here, finite field operations use only affine coordinates, which keeps interoperability with software implementations. Also, core operators such as the finite field multiplier and inversion are highly configurable, to enable fulfilling constraints of area or performance, according to what is relevant to each application. The core can thus serve several different purposes, through the setting of parameters that pick the adequate elliptic curve and others that control the finite field multiplier and inversion operator characteristics. Results obtained by synthesizing the core with different parameters for Xilinx FPGAs and 65nm CMOS ASICs show that the exploitable design space is ample: implementations can take from 17 to 5,800μs to execute a point multiplication, taking from 5.7 to 162.2 thousand LUTs in FPGAs or from 16,6 to 360,7 thousand gates in ASICs.**

Keywords – cryptography; ECC; soft IP; FPGA; ASIC.

I. INTRODUCTION[1]

In the 1980s, Miller [1] and Koblitz [2] were the first to propose the use of elliptic curves for cryptography. Elliptic curve cryptography (ECC) is a public-key scheme just like the widely used Rivest, Shamir, and Adleman (RSA) algorithm. However, ECC has an advantage over RSA, the fact that for a similar security level ECC requires smaller keys than RSA. This means that ECC implementations require less memory and computational power to execute. For example, a key size of 163 and 571 bits in ECC has security equivalent to a 1024 and 15360 bits respectively in RSA implementation. Due this advantage, many standards for ECC were defined in the last decade, such as those proposed by the National Institute of Standards and Technology (NIST) [3], which recommends a set of parameters for several elliptic curves with different sizes, and by the Standard for Efficient Cryptography Group (SECG) [4], an industrial consortium dedicated to achieve interoperability among cryptography equipments. In short, ECC has been demonstrated to be more efficient than RSA. This article proposes a new soft intellectual property (IP) core supporting NIST recommendations and following the SECG standards. Its original contribution is the degree to which this soft core [5] can be parameterized to achieve the requirements of many applications in terms of area, speed and possibly power.

[1] This work was partially supported by the CAPES-PROSUP and FAPERGS (under grant 11/1445-0). Ney Calazans acknowledges the support of CNPq under grant 310864/2011-9.

II. ELLIPTIC CURVE CRYTOGRAPHY CONCEPTS

The main ECC operation is the point multiplication, composed by a point adder and a squarer. The point adder comprises finite field multiplier and inversion operators, which execute the most time-consuming operations in ECC. There are several studies demonstrating how to implement these operations efficiently in hardware or software and because of this, performance measurements are often based on the time taken to compute a single point multiplication in an elliptic curve. This is the basic metric used here for comparing ECC implementations, as Section IV shows.

ECC can be used over different representations and sizes of finite fields. In binary Galois Fields GF(2^m), the polynomial bases and normal bases are the most used, but for ECC in hardware, polynomial bases are the most indicated, due to their efficiency in this kind of implementation. Accordingly this work only considers ECC over binary fields GF(2^m), represented in polynomial bases. The elliptic curves and parameters chosen to be implemented are the Koblitz curves recommended by NIST [3], but the proposed core can also support the choice of other curves and parameters. Points of an elliptic curve can be represented by affine coordinates, as (x, y), or projective coordinates, such as (x, y, z), and this is important because some algorithms for scalar point multiplication, which use projective coordinates, are usually more efficient than those which employ affine coordinates. However, using projective coordinates requires a converter between these two kinds of representations to keep the interoperability among different systems and standards. This usually implies extra area overhead due to the added converter. In this work, we decide to only use affine coordinates, automatically keeping the interoperability with available software implementations.

At a high level of abstraction, ECC executes an algorithm composed by the elements and steps of Figure 1. The computation uses a pre-defined elliptic curve, such as one of Koblitz curves, to calculate the listed steps. All these steps are performed over GFs. Elements in uppercase are points of the pre-defined curve and the other elements are integers. According to the literature, the main representations used for ECC are GFs over prime fields, GF(p^q) with p being a

prime, and over the special case where p=2, called binary Galois Fields, GF(2^m). The arithmetic operations are different in GF from the traditional arithmetic with integers or real numbers. Integer additions or subtractions in GFs are a same operation, which correspond to just an exclusive-or between binary representations of two integers. Point addition and point multiplication are also modular operations. Implementing these is one of the difficulties in ECC, as numbers are at least 160-bit wide.

- **Q** is a point of the curve, which is the public key;
- **d** is a random number, which is the private key;
- **P** is the point generator of the curve;
- **M** is the message to encrypt;
- **k** is a random number, generated for each message;
- **C1** and **C2** are points of the curve, which represent the encrypted message;

1. Key Generation: **Q = d * P**
2. Encryption: **C1 = k * P ; C2 = M + k * Q**
3. Decryption: **M = C2 – d * C1**

Figure 1 - Simplified view of the ECC cryptography algorithm steps.

III. ARCHITECTURE AND DESIGN OF ECC CORE

The main objective of the soft IP core design proposed in this work is to be very flexible, by supporting many different parameter settings and curve choices. This provides a hardware description that can be used in several applications with widely different constraints of area, timing and power. Thus, each operation was developed as a module (a finite field squarer, an inversion/divider, a multiplier, a point adder and a point multiplier).

Figure 2 shows an example of module composition. All modules are configurable to support different sizes, according to the elliptic curve in use. Depending on the size 'm' in bits, the best architecture can be chosen for some modules. For example, the finite field inversion operator offers two different architectures each based on an algorithm that gives better values for a different metric.

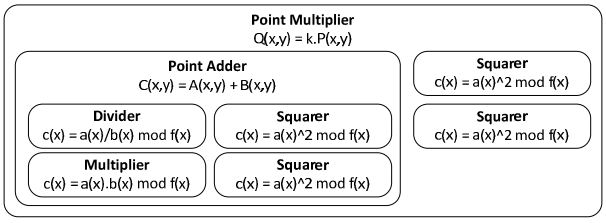

Figure 2 - Composition of modules to create a point multiplier.

Finite field and elliptic curves operators rely on the algorithms proposed by Deschamps et al. [6], which discuss each operation, provide alternative implementations and assess relative efficiencies. Using the most efficient algorithms from that reference, we designed a generic description able to be used with any of the 5 Koblitz curves with sizes 163, 233, 283, 409 and 571 bits. Among all operations, the squarer is the simplest, because to square a number of 'm' bits in polynomial bases consists in inserting '0s' between all the other bits and then reduce the '2*m' bits using an algorithm with the irreducible polynomial of the elliptic curve in use. Squaring is thus a fully combinational block in hardware, efficiently computable in one clock cycle.

For the finite field multiplier we chose to use the interleaved multiplication algorithm, because it is a shift-and-add method with an interleaved reduction step, which can be implemented to use smaller area and yet achieve the same efficiency of other algorithms. This algorithm can be improved to perform a multiplication in fewer clock cycles, if more area is used, as detailed in [7]. This is one of the improvements of our design compared to the literature: we developed a configurable multiplier. When choosing the elliptic curve, it is also possible to choose a relative performance for the multiplier. Its hardware can be set to compute from one bit per clock cycle to up to 'n' bits per clock cycle, where n is the result of rounding half the key size. Considering a binary field GF(2^m), the multiplier can perform one operation in 'm' clock cycles down to '2' clock cycles. Let us take as an example the elliptic curve K-163 that uses GF(2^{163}). If the multiplier is configured to calculate one bit per clock cycle, then one finite field multiplication takes 163 clock cycles. However, if it is configured to calculate 82 bits per clock cycle, then one finite field multiplication will take only 2 clock cycles. Meanwhile, area overhead is not increased 41 times, as results in Section IV demonstrates. Thus, designers can select adequate timing-area trade-offs.

The finite field inversion or divider operator also requires many clock cycles to execute. We designed two alternative architectures to it. One implements the binary algorithm detailed in [6], that performs a division in '2*m' clock cycles, which is good when area constraints are strict. Itoh and Tsujii [8] proposed a method to calculate the inverse of a number in normal bases that can be performed with (m-1) squaring operations and (m-2) multiplications. This would take much more time than the binary algorithm, due to the required multiplications, but some papers such as Guajardo et al. [9] and Bednara et al. [10] present the Itoh-Tsujii method adapted for standard bases, or polynomial bases. Adapting these methods, it is possible to calculate the inverse of a number in much fewer clock cycles, based on a theorem that the inverse of a number A in a finite field is $A^{-1} = A^{2^{m-1}-1}$. Decomposing the exponent, it is possible to compute the inverse using only 'm-1' squaring operations and 9 to 12 multiplications when using the Koblitz curves K-163 to K-571, respectively. Thus, if a finite field multiplication can be performed efficiently and considering that a squaring operation is performed in one clock cycle, then it is possible to computer the inverse in fewer clock cycles than the binary algorithm, which takes '2*m' clock cycles. For example, using again K-163 and a finite field multiplier configured to multiply in 2 clock cycles, to compute the inverse would take 162 squaring operations and 9 multiplications, which amounts to '162*1+9*2' clock cycles or 180 clock cycles, much less than the 326 clock cycles required by the binary algorithm.

Of course, finite field multiplier configurations affect the finite field inversion module, because the latter uses the architecture that is more efficient, depending on the multiplier configuration. If the number of cycles to perform 'm-1' squaring operations and 9 to 12 multiplications is bigger

than '2*m' cycles, inversion uses the first architecture (the binary algorithm), otherwise it uses the second architecture (the adapted Itoh-Tsujii algorithm).

After designing the three basic modules (squarer, multiplier, inversion), we adapted the point adder and point multiplication modules detailed in [6] to perform operations over any of the Koblitz curves. Next, we developed a basic architecture to implement ECC in hardware, including key generation, encryption and decryption, as Figure 3 shows. All these operations can be performed simultaneously. Key generation comprises a point multiplier. The Encrypter contains two point multipliers and one point adder. The Decrypter has one point multiplier and one point adder.

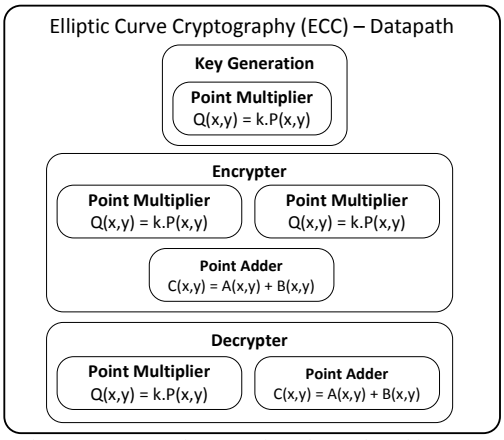

Figure 3 - Proposed ECC soft IP datapath architecture.

IV. Experiments and Results

We conducted post-synthesis timing simulations of the point multiplier (Figure 2) in Modelsim, varying several parameters and measuring the average number of clock cycles taken to compute a single point multiplication for the 5 Koblitz curves recommended by NIST. We also synthesized all configurations to obtain area and operating frequency data. Synthesis results are based on the Xilinx ISE 14.1 XST tool, targeting a Virtex-7 XC7V2000T FPGA with speed grade -2. We also synthesized all configurations for the STMicroelectronics 65nm CMOS technology using Cadence Encounter and the foundry standard cell library in its Low Power, High Threshold version ($LPHV_t$).

Table 1 shows the average timing to compute one single point multiplication (k*P) for simulations in Modelsim with different parameters. The first column specifies the key size (M) in bits, according to the employed Koblitz curve. The second column defines the multiplier kind, based on how many bits (W) are computed at each clock cycle. The third column is M/W, the time expected to compute a multiplication in clock cycles. The fourth column shows the average time obtained for each architecture. As expected, the bigger the multiplier, the less clock cycles are necessary to perform each multiplication and inversion. Consequently, point multiplication is also performed in less clock cycles. Table 1 also shows the FPGA synthesis results, for all multiplier architectural choices, and the synthesis results for the same architectures in the mentioned ASIC technology. The fifth and ninth columns show the maximum predicted operating frequency and the sixth and tenth the associated clock period, in nanoseconds. Considering the measured number of cycles in the fourth column, the eighth and fourteenth columns show the average time in nanoseconds for executing one point multiplication. The seventh column shows the amount of used LUTs, from 5.768 to 162.251 LUTs, which correspond to less than 1% and to up to 13% respectively of target FPGA utilization. The eleventh column contains the needed number of gates for the point multiplier. Finally, the twelfth and thirteenth columns show the leakage and dynamic power estimated, while Figure 4 shows the estimated total power for the ASIC implementations. Table 1 demonstrates that it is possible to configure parameters according to application constraints of area, power and performance. Again, it is possible to achieve higher performance using the same design and just setting parameters according to the desired security level, while considering constraints of area and performance.

Table 1 - Modelsim simulation results; Synthesis results for ISE 14.1 XST; Synthesis results of Cadence Encounter for 65nm CMOS.

Parameters		Average Timing (clock cycles)		FPGA - XC7V2000T-2FLG1925				CMOS - 65 nm					
M (bits)	W (bits)	Multiplications	Point Multiplications (k*P)	Max. Clock Freq. (MHz)	Clock Period (ns)	Slice LUTs	Point Multiplication kP (ns)	Max. Clock Freq. (MHz)	Clock Period (ns)	Gates	Leakage Power (mW)	Dynamic Power (mW)	Point Multiplication – k*P (ns)
163	1	163	48,863	223.326	4.478	5,768	218,797	1,338	0.747	16,652	1,241	29,979	36,520
	21	8	29,463	223.328	4.478	9,976	131,928	1,329	0.752	34,418	2,239	58,896	22,170
	82	2	23,061	222.027	4.504	18,852	103,866	1,342	0.745	72,895	4,174	90,605	17,185
233	1	233	103,449	225.405	4.436	6,737	458,948	1,355	0.738	23,744	1,741	42,086	76,347
	30	8	55,889	225.405	4.436	14,721	247,95	1,342	0.745	58,333	3,642	91,942	41,647
	117	2	45,449	225.405	4.436	32,271	201,633	1,351	0.740	136,846	7,985	140,210	33,642
283	1	283	164,823	192.945	5.183	8,245	854,249	1,283	0.779	29,458	2,215	51,795	128,467
	36	8	85,367	184.733	5.413	19,866	462,111	1,283	0.779	77,858	4,784	116,741	66,538
	142	2	70,469	184.114	5.431	45,510	382,747	1,297	0.771	195,382	11,188	190,865	54,333
409	1	409	314,085	147.358	6.786	11,229	2,131,442	1,126	0.888	36,272	2,567	62,047	278,939
	52	8	142,045	147.358	6.786	33,863	963,945	1,058	0.945	127,355	6,641	152,607	134,259
	205	2	123,829	147.358	6.786	85,447	840,328	1,003	0.997	360,755	18,096	281,859	123,459
571	1	571	654,543	111.506	8.968	16,337	5,870,025	889	1.125	48,691	3,187	71,508	736,269
	72	8	267,205	111.506	8.968	58,164	2,396,329	896	1.116	221,393	11,116	230,604	298,220
	286	2	251,287	111.506	8.968	162,251	2,253,574	844	1.185	673,526	29,623	414,768	297,734

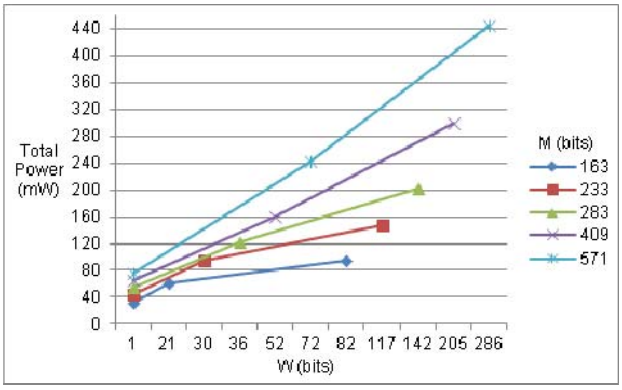

Figure 4 – Total power dissipation comparison for ECC ASIC implementations.

Our ECC soft IP core demonstrates to be very flexible, due to the possibility of being configured with different sets of elliptic curves and parameters for the basic operations of finite fields.

Next, Table 2 compares results obtained here with some related works. We considered only the higher performances in each related work. Table 2 specifies the coordinate representation, the target platform and the maximum frequency, to allow a fair comparison among all works.

TABLE 2 - COMPARISONS OF PERFORMANCE WITH RELATED WORKS. NOTE THAT LUTS FOR VIRTEX-7 HAVE 6 INPUTS, WHILE FOR VIRTEX-4 AND EP2S LUTS HAVE 4 INPUTS.

Work	Coordinate System	Koblitz Curves GF(2^m)	Platform	Max. Freq. (MHz)	Area (LUTs or Gates)	Q=k*P (ms)
Li et al. [10]	Projective Coordinates	K-283	Virtex-4 XC4VFX140-11	171	51,094	0.304
Dias et al. [11]	Affine Coordinates	K-163	2 Altera FPGAs (EP2S180F1020C4 and EP2S180F1020C3)	250	216,288	0.100
Loi and Ko [12]	Projective Coordinates	K-163	Virtex-4 XC4VFX12	155	3,815	0.273
		K-233				0.604
		K-283				0.735
		K-409				1.926
		K-571				4.335
This Work	Affine Coordinates	K-163	Virtex-7 XC7V2000T-2	222	18,852	0.103
			65nm CMOS	1,342	72,895	0.017
		K-233	Virtex-7 XC7V2000T-2	225	32,271	0.201
			65nm CMOS	1,351	136,846	0.033
		K-283	Virtex-7 XC7V2000T-2	184	45,510	0.382
			65nm CMOS	1,297	195,382	0.054
		K-409	Virtex-7 XC7V2000T-2	147	85,447	0.840
			65nm CMOS	1,003	360,755	0.123
		K-571	Virtex-7 XC7V2000T-2	111	162,251	2.253
			65nm CMOS	844	673,526	0.297

V. CONCLUSIONS

According to the achieved results Table 2 demonstrates, our soft IP core for ECC can achieve higher performance, even when compared to other works which are not so flexible and use projective coordinates. This occurs mostly because the FPGA and ASIC technologies employed here are clearly more advanced (at least two generations ahead for FPGAs), but the improved architecture design are also expected to contribute to achieve higher performance. Finally, Table 3 shows some synthesis results for the complete ECC architecture depicted in Figure 3 (including the datapath and associated controller), which is a complete core for ECC in hardware. This can be compared with the partial performance and size results.

TABLE 3 - SYNTHESIS RESULTS OF ECC SOFT IP CORE.

Parameters (bits)		FPGA XC7V2000T-2		CMOS 65 nm	
M	W	Clock Frequency	LUTs	Clock Frequency	Gates
163	1	168 MHz	27,297	909 MHz	77,243
	82	158 MHz	103,468	899 MHz	402,587
233	1	199 MHz	39,559	909 MHz	106,141
	117	199 MHz	192,637	906 MHz	756,057
283	1	171 MHz	48,307	858 MHz	137,820
	142	156 MHz	268,095	895 MHz	1,093,746
409	1	147 MHz	64,393	893 MHz	192,421
	205	147 MHz	506,902	869 MHz	2,147,644
571	1	106 MHz	92,459	855 MHz	280,220
	286	95 MHz	964,039	883 MHz	4,062,631

REFERENCES

[1] Miller, V. "*Use of Elliptic Curve in Cryptography*". Advances in Cryptology, LNCS, 218, 1986, pp. 417-426.

[2] Koblitz, N. "*Elliptic Curve Cryptosystems*". Mathematics of Computation, 48(177), Jan 1987, pp. 203-209.

[3] National Institute for Standards and Technology, "*Recommended Elliptic Curves for Federal Governement Use*", Jul 1999.

[4] Brown, D. "*Standard for Efficient Cryptography 1*". Certicom Research, May 2009.

[5] Gupta, R.; Zorian, Y. "*Introducing Core-Based System Design*". IEEE Design & Test of Computers, 14(4), Oct-Dec 1997, pp.15-25.

[6] Deschamps, J.; Imanã, J.; Sutter, G. "*Hardware Implementation of Finite-Field Arithmetic*". McGraw-Hill, 2009.

[7] Guajardo, J.; Güneysu, T.; Kumar, S.; Paar, C.; Pelzl, J. "*Efficient Hardware Implementation of Finite Field with Applications to Cryptography*". Acta Applic. Mathematica, 93, Sep 2006, pp. 75-118.

[8] Itoh, T.; Tsujii, S. "*A Fast Algrithm for Computing Multiplicative Inverse in GF(2^m) Using Normal Bases*". Information and Computation, 78, 1988, pp. 171-177.

[9] Guajardo, J.; Paar, C. "*Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes*". Designs, Codes and Cryptography, 25, Feb 2002, pp. 207-216.

[10] Bednara, M.; Daldrup, M.; Gathen, J.; Shokrollahi, J.; Teich, J. "*Reconfigurable Implementation of Elliptic Curve Crypto Algorithms*". In: IPDPS'02, 2002.

[11] Li, H.; Huang, J.; Sweany, P.; Huang, D. "FPGA Implementations of Elliptic Curve Cryptography and Tate Paring Over Binary Field". Journal of Systems and Architecture, 54, Dec 2008, pp. 1077-1088.

[12] Dias, M.; Gouveia, M.; Oliveira, J.; Muñoz, I.. "*Bit-Parallel Coprocessor for Standard ECC-GF(2^m) on FPGA*". International Journal of Applied Mathematics, 26(2), 2013, pp. 241-262.

[13] Loi, K. C. C.; Ko, S. B. "*High Performance Scalable Elliptic Curve Cryptosystem Processor for Koblitz Curves*". Microprocessors and Microsystems, 37, Jun 2013, pp. 394-406.