A Hardware Analysis of Twisted Edwards Curves for an Elliptic Curve Cryptosystem

Brian Baldwin[1,2], Richard Moloney[1,3], Andrew Byrne[2], Gary McGuire[1,3], and William P. Marnane[1,2]

[1] Claude Shannon Institute for Discrete Mathematics, Coding and Cryptography
[2] Dept. of Electrical & Electronic Engineering, University College Cork, Cork, Ireland
[3] Dept. of Mathematics, University College Dublin, Dublin, Ireland
{brianb,andrewb,liam}@eleceng.ucc.ie,
{richard.moloney,gary.mcguire}@ucd.ie

Abstract. This paper presents implementation results of a reconfigurable elliptic curve processor defined over prime fields $GF(p)$. We use this processor to compare a new algorithm for point addition and point doubling operations on the twisted Edwards curves, against a current standard algorithm in use, namely the Double-and-Add. Power analysis secure versions of both algorithms are also examined and compared. To the authors' knowledge, this work introduces the first documented FPGA implementation for computations on twisted Edwards curves over fields $GF(p)$.

1 Introduction

In this paper we present implementation results of a reconfigurable elliptic curve processor (ECP) for a generalisation of the Edwards curve [1], the twisted Edwards curve, recently proposed by Bernstein et. al. [2]. We examine both the implementation efficiency and implementation security of twisted Edwards curves and compare them against current standard curves and methods in use today. We examine firstly, in projective coordinates, the explicit formulas for point addition and point doubling of the standard twisted Edwards formulas [1] and compare against the widely used Double-and-Add method [3].We then examine the strongly unified formula, which is resistant to simple power analysis (SPA) [4], a form of side-channel analysis, and compare it to its equivalent, the Double-and-Add-Always method [5].

2 Elliptic Curves

We consider an elliptic curve over the field $GF(p)$ for some prime p. In Jacobian projective coordinates, this curve is given by the equation

$$Y^2 = X^3 + AXZ^4 + BZ^6 \tag{1}$$

[1] Explicit-formulas database. URL: http://hyperelliptic.org/EFD

where the Jacobian projective point $(X_1 : Y_1 : Z_1)$ corresponds to the affine point $(X_1/Z_1^2, Y_1/Z_1^3)$ if $Z_1 \neq 0$, and $\mathcal{O}$ the point at infinity, if $Z_1 = 0$.

The basic operations of ECC are point scalar computations of the form $Q = [k]\,P$. Point scalar multiplication (PM) can be performed using algorithms such as the Double-and-Add method [3]. This method requires $m - 1$ point doublings (PD) and $w - 1$ point additions (PA), where m is the length and w is the Hamming weight of the binary expansion of k.

Each PA and PD is comprised of finite field additions, subtractions, multiplications and inversions. By representing each point on the curve in projective (X, Y, Z) rather than affine (x, y) coordinates, each PA and PD can be performed without the need for inversions, albeit at the cost of extra multiplications. This will improve efficiency since the cost of inversions is significantly more expensive than multiplications [6].

The equations governing PA and PD in projective coordinates using the Double-and-Add method, on a Weierstrass curve, are given in [3]. Each PA requires 16 multiplications and 7 additions/subtractions, with 10 multiplications and 4 additions/subtractions required for a PD.

2.1 Simple Power Analysis Resistance

Simple Power Analysis (SPA), makes use of side-channel analysis to monitor and measure the power emitted from a single execution of a cycle of a crypto processor [7]. Each PA and PD operation produces a different power trace when executed because of the different number of multiplications and additions involved in each, and as the execution of a point addition in the Double-and-Add is directly related to the current bit of the secret key (k_i), it is possible to retrieve the secret key by monitoring the power consumption of a single execution of a scalar multiplication.

The Double-and-Add-Always Algorithm [3], is a simplistic approach to solving the problem of the SPA susceptibility. It performs dummy point addition executions, so that every execution of the key k executes a point double and a point addition regardless of whether $k_i = 0$ or $k_i = 1$, with the key bit deciding where to write the result. This leads to an inefficient design as unnecessary operations are performed, but it does prevent the recognition of individual bits.

2.2 Edwards Curves

In [2], Bernstein et al. introduced the twisted Edwards curves

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{2}$$

where $a,\ d \in GF(p)$ are distinct and non-zero. If $a = 1$, the curve may be called an Edwards curve. They further showed that a significant number of elliptic curves over $GF(p)$ (roughly 1/4 of isomorphism classes of elliptic curves) are birationally equivalent to a twisted Edwards curve. Two curves are birationally equivalent if there is an invertible rational mapping between them (such as $(x, y) \mapsto (\frac{y}{x-1}, \frac{x}{y-1})$), which may be undefined at a finite number of points.

The chief advantage of Edwards and twisted Edwards curves over standard curves is that the addition laws defined on them can be made unified, i.e., a single addition formula can be used to add points and double points, with no exception for the identity. We use the projective twisted Edwards curve

$$aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2 \tag{3}$$

so as to avoid inversions. The projective point $(X_1 : Y_1 : Z_1)$ corresponds to the affine point $(X_1/Z_1, Y_1/Z_1)$.

Algorithm 1 and 2 give the PD and PA for the non SPA resistant twisted Edwards algorithms, while Algorithm 3 gives the unified formula.

Algorithm 1: Point Doubling for twisted Edwards

```
input  : $P(X_1, Y_1, Z_1) \in GF(p)$
output : $[2]P(X_3, Y_3, Z_3) \in E(GF(p))$
$B = (X_1 + Y_1)^2, C = X_1^2$
$C = X_1X_2, D = Y_1Y_2, E = aC$
$F = E+D, H = Z_1^2, J = F-2H$
$X_3 = (B - C - D)J,$
$Y_3 = F(E - D), Z_3 = FJ$
```

Algorithm 2: Point Addition for twisted Edwards

```
input  : $P(X_1, Y_1, Z_1);$
         $Q(X_2, Y_2, Z_2) \in GF(p)$
output : $P(X_3, Y_3, Z_3) \in E(GF(p))$
$A = Z_1Z_2, B = A^2;$
$C = X_1X_2, D = Y_1Y_2;$
$E = dCD, F = B - E, G = B + E;$
$X_3 = AF((X_1+Y_1)(X_2+Y_2)-C-D;$
$Y_3 = AG(D - aC), Z_3 = FG$
```

Algorithm 3: Unified twisted Edwards point operation

```
input  : $P(X_1, Y_1, Z_1); Q(X_2, Y_2, Z_2) \in GF(p)$
output : $P + Q(X_3, Y_3, Z_3) \in E(GF(p))$
$A = Z_1Z_2, B = A^2, C_1 = aX_1X_2, C_2 = X_1Y_2;$
$D_1 = Y_1Y_2, D_2 = X_2Y_1, E = dC_2D_2, F = B - E, G = B + E;$
$X_3 = AF(C_2 + D_2), Y_3 = AG(D_1 - C_1), Z_3 = FG$
```

Algorithm 2 requires 12 multiplications and 8 additions, Algorithm 1 requires 8 multiplications and 7 additions and Algorithm 3 processes the same formula for both PA and PD, thereby giving it the same power trace for either operation, at a cost of 14 multiplications and 5 additions per point operation.

3 FPGA Based Elliptic Curve Processor

A reconfigurable architecture for performing elliptic curve cryptography was designed [8] and ported onto an FPGA device. It consists of a controller, containing an instruction set stored in ROM and a finite state machine (FSM), a user definable number of arithmetic logic units (ALU's) for addition, subtraction and multiplication calculations in parallel, and BlockRAM for storage of results, as illustrated in Figure 1.

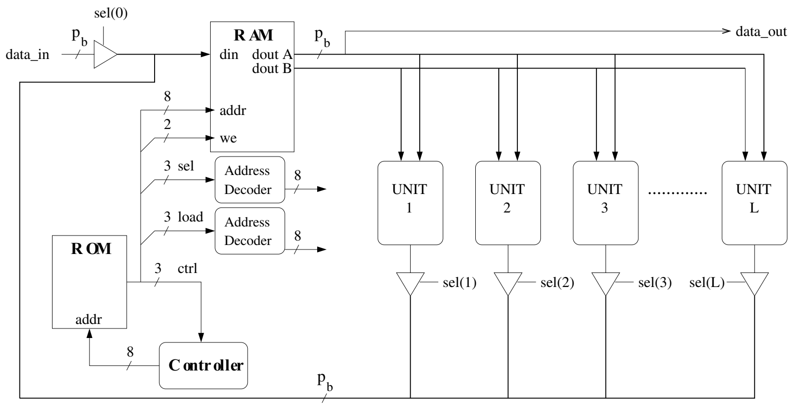

Fig. 1. Reconfigurable Elliptic Curve Processor

The ALUs perform the $GF(p)$ operations described in section 2, namely the modular multiplications, additions and subtractions. Mode bits are used to select between operations. Modular multiplication is achieved using the Montgomery multiplication algorithm [9] and is executed in $p_b + 2$ clock cycles, where p_b is the field size in bits, while modular additions and subtractions each take 2 clock cycles.

4 Performance Results of ECC Algorithms

The ECP can be programmed to run any number of ALUs in parallel to process an elliptic curve formula. This design is limited only by the size of the FPGA. For this paper, the architecture was evaluated on a Spartan3E XC3S500E, and used one to four ALUs operating in parallel. Table 1 shows the measured results for the FPGA.

Table 1 firstly details the post place and route (PPR) clock frequency (F_{max}). There is very little variation in clock frequency between the different algorithms. The clock frequency in fact depends on the number and type of ALUs used. The minimum PPR frequency reported for all the configurations and combinations was recorded when using four multipliers.

The circuit design also remains the same for each of the four formulae, differing only in the size of the instruction set in ROM. The area, therefore, approximately remains the same for each of the four different formulae, and increases equivalently with more ALUs.

The average power (Pwr) dissipation of the processor was measured at a frequency of 10 MHz. The current being drawn by the FPGA on its $VCCINT$ and $VCCAUX$ lines was measured. The voltage supplied to each line by the board's voltage regulator was also measured. These voltage and current measurements were then used to calculate the total average power consumed on both lines. The energy per average multiplication is also given. The energy is calculated using the

Table 1. Spartan3E XC3S500E-4fg320 FPGA Results

Double-and-Add						
ALU	F_{max} (Mhz)	*Area* (slices)	*Time* (ms)	*Pwr* (mW)	*Energy* (mJ)	*ATProd* (Clk/Slic)
1	27.921	1703	30.704	88.04	2.703	60.831
2	28.333	2896	16.582	96.53	1.601	56.692
3	27.84	3988	12.917	109.58	1.415	59.756
4	26.438	4654	12.905	107.52	1.386	66.161
Double-and-Add-Always						
ALU	F_{max} (Mhz)	*Area* (slices)	*Time* (ms)	*Pwr* (mW)	*Energy* (mJ)	*ATProd* (Clk/Slic)
1	28.539	1702	48.863	87.84	4.292	98.893
2	28.183	2897	25.935	100.56	2.608	88.231
3	27.808	3989	18.331	106.99	1.961	84.726
4	26.68	4654	17.012	105.73	1.798	92.205
twisted Edwards						
ALU	F_{max} (Mhz)	*Area* (slices)	*Time* (ms)	*Pwr* (mW)	*Energy* (mJ)	*ATProd* (Clk/Slic)
1	28.245	1703	18.428	88.08	1.623	36.935
2	28.746	2898	10.565	98.67	1.042	36.144
3	28.01	4269	7.448	115.82	0.863	36.884
4	25.097	4654	7.836	105.93	0.83	40.173
twisted Edwards Strongly Unified						
ALU	F_{max} (Mhz)	*Area* (slices)	*Time* (ms)	*Pwr* (mW)	*Energy* (mJ)	*ATProd* (Clk/Slic)
1	27.976	1700	27.247	88.09	2.4	55.08
2	27.852	3171	19.743	98.67	1.931	66.058
3	27.816	4553	18.041	115.82	1.926	73.868
4	25.052	4654	12.543	105.93	1.317	64.784

Table 2. Point Double and Point Addition Timing

Double-and-Add						
	Point Double			Point Addition		
ALU	*Mul*	*Add*	*Clks*	*Mul*	*Add*	*Clks*
1	10	13	1948	26	20	5035
2	6	13	1178	13	20	2538
3	5	13	962	9	20	1770
4	5	13	962	8	20	1578
Double-and-Add-Always						
	Point Double			Point Addition		
ALU	*Mul*	*Add*	*Clks*	*Mul*	*Add*	*Clks*
1	25	20	4842	25	20	4842
2	13	20	2538	13	20	2538
3	9	20	1770	9	20	1770
4	8	20	1578	8	20	1578
twisted Edwards						
	Point Double			Point Addition		
ALU	*Mul*	*Add*	*Clks*	*Mul*	*Add*	*Clks*
1	8	7	1536	12	8	2322
2	4	7	770	8	6	1550
3	3	7	578	5	7	976
4	3	7	578	5	6	974
twisted Edwards Strongly Unified						
	Point Double			Point Addition		
ALU	*Mul*	*Add*	*Clks*	*Mul*	*Add*	*Clks*
1	14	5	2700	14	5	2700
2	9	4	1736	9	4	1736
3	7	4	1352	7	4	1352
4	6	4	1160	6	4	1160

average power value and the average time per point multiplication, as shown in Tables 1, based on the number of clock cycles and the 10 MHz clock frequency.

4.1 Computation Time

The scheduling of the point operations was examined, again with a variable number of ALUs, to review the timing of each algorithm. Using a key size of 192 and performing all multiplications and additions/subtractions for the Point Addition and Point Doubling Algorithms for the Double-and-Add, and the twisted Edwards Algorithms, 1, 2 and 3, defined in Section 2, the timing results in Table 2 were obtained. As can be seen from the table, the number of multiplication stages (Mul) required to process an algorithm decreases with an increase in parallelisation. We do not modify the number of addition stages *Add* as an addition is completed in only two clock cycles. Next we tested each of the formulae with a 192-bit value for the key (k) and a Hamming weight of $\frac{k}{2}$ to measure an iteration of the algorithms. Table 1 shows the timing results. The table shows that the standard twisted Edwards algorithm performs on average 60% faster than its equivalent Double-and-Add algorithm for all counts of ALUs.

The table also shows that the SPA resistant unified twisted Edwards performs comparably to the non SPA resistant Double-and-Add method and performs faster for one or four ALUs. However, neither the standard twisted Edwards nor Double-and-Add achieve any great increase in timing when increasing from

3 ALUs to 4 ALUs, due to the algorithms limitations of parallelism. From the table we can see that again the standard twisted Edwards gives the best value across the range of ALUs, with 3 ALUs giving the best performance. For the Unified twisted Edwards and both the Double-and-Add formulae 4 ALUs gives the fastest timing.

4.2 Area Time Product

The area-time (AT) product was calculated to get a representation of any speed increase against the increase in size, as shown in Table 1. This gives a more accurate representation of the cost that each increase in ALU has in relation to the overall system. The minimum AT value, i.e. the most efficient combination in an area time sense, is again the standard twisted Edwards, giving the best value across the range of ALUs, with 2 ALUs giving the best performance. For the Unified twisted Edwards, a single ALU gives the best performance, while the Double-and-Add formulae give best AT at 2 and 3 ALUs respectively.

5 Conclusions

In this paper, we presented implementation results of an ECP with a reconfigurable architecture and used it to compare the standard and strongly unified formulae that define the twisted Edwards curve, against the Double-and-Add and Double-and-Add-Always formulae. We showed that the twisted Edwards performs on average 60% faster and uses less area than the Double-and-Add, and that the performance of the SPA resistant strongly unified version of the twisted Edwards, far exceeded its Double-and-Add-Always equivalent. We also showed that by using one or four ALUs operating in parallel, the strongly unified twisted Edwards execution time exceeds the Double-and-Add for an equivalent number of ALUs. Future work could involve an examination of the cost of converting a Double-and-Add to a strongly unified twisted Edwards curve to gain SPA resistance at comparable speeds.

Acknowledgement

Supported by the Science Foundation Ireland under Grant No. 06/MI/006.

References

1. Edwards, H.M.: A normal form for elliptic curves. Bulletin of the American Mathematical Society 44(3), 393–422 (2007)
2. Bernstein, D., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008)

3. Knuth, D.E.: The Art of Computer Programming, 3rd edn. Seminumerical Algorithms of Addison-Wesley series in computer science and information processing, vol. 2. Addison-Wiley, Chichester (2001)
4. Brier, E., Joye, M.: Weierstraß Elliptic Curves and Side-Channel Attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
5. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
6. Orlando, G., Paar, C.: A scalable GF(p) elliptic curve processor architecture for programmable hardware. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 356–371. Springer, Heidelberg (2001)
7. Örs, S.B., Oswald, E., Preneel, B.: Power-analysis attacks on an FPGA – first experimental results. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 35–50. Springer, Heidelberg (2003)
8. Byrne, A., Popovici, E., Marnane, W.P.: Versatile processor for GF(p) arithmetic for use in cryptographic applications. In: Computers & Digital Techniques, IET, vol. 2, pp. 253–264 (July 2008)
9. Montgomery, P.: Modular multiplication without trial division. In: Mathematics of Computation, vol. 44, pp. 519–521 (1985)