A version of this paper appears in ARC 2009, the 5th International Workshop on Applied Reconfigurable Computing

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

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

[1] Claude Shannon Institute for Discrete Mathematics, Coding and Cryptography. Dept. of Electrical & Electronic Engineering, University College Cork, Cork, Ireland
[2] Claude Shannon Institute for Discrete Mathematics, Coding and Cryptography. 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. Secure power analysis versions of both algorithms are also examined and compared. The algorithms are implemented on an FPGA, and the speed, area and power performance of each are then evaluated for various modes of circuit operation using parallel processing. To the authors' knowledge, this work introduces the first documented FPGA implementation for computations on twisted Edwards curves over fields $GF(p)$.

1 Introduction

Elliptic Curve Cryptography (ECC) was established as a form of public key cryptosystem in 1985 by Miller [1] and Koblitz [2]. Its advantage over other public key cryptosystems, such as RSA [3], is that it provides an equivalent level of security using shorter cryptographic key sizes. Increasing the key size increases the overall security of the cryptosystem [4]. In practice however, hardware resources such as computer processing power and system memory are limiting factors which can reduce the speed and increase the area for such an increase. Therefore, a balance needs to be set between the complexity of the mathematical computation, and the resources available to implement the computations.

Another element to take into account when designing a public key cryptosystem is the physical security of the system. Implementations can leak sensitive information during the execution of a computation [5], which may lead to a release of secret information, no matter how mathematically secure the system may be. This method of side-channel analysis consists of monitoring some side-channel information, such as the power consumption [6], and using the data emitted to deduce, or partially deduce, the secret key [7].

In this paper we present implementation results of a reconfigurable elliptic curve processor for a generalisation of the Edwards curve [8], the twisted Edwards curve, recently proposed by Bernstein et al [9]. 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 widely used Double-and-Add method [10] and compare against the standard twisted Edwards formulas [1].We then examine the strongly unified formula, which is resistant to simple power analysis (SPA) [7], a form of side-channel analysis, and compare it to its equivalent, the Double-and-Add-Always method [11].

2 Elliptic Curves

We consider an elliptic curve over the field $GF(p)$ for some prime p, given by the affine Weierstrass equation

$$y^2 = x^3 + Ax + B. \tag{1}$$

In Jacobian projective coordinates, this curve is given by the equation

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

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 $Z_1 = 0$. We consider Jacobian projective coordinates as they currently provide the fastest implementation of ECC for hardware.

The basic operations of ECC are point scalar computations of the form:

$$Q = [k]\,P = \underbrace{P + P + \cdots + P}_{k\ times} \tag{3}$$

The Elliptic Curve Discrete Logarithm Problem (ECDLP) is the problem of retrieving k given P and Q, where P is a point on the curve and k is an integer [12]. The assumed difficulty of this problem is the basis of security for elliptic curve public key schemes. Point scalar multiplication (PM) can be performed using algorithms such as the Double-and-Add method, as shown in Algorithm 1. 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

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

will improve efficiency since the cost of inversions is significantly more expensive than multiplications [13].

The equations governing PA and PD in projective coordinates using the Double-and-Add method, Algorithm 1, on a Weierstrass curve, are given in Algorithms 3 and 4 respectively. Each PA requires 16 multiplications and 7 additions/subtractions, with 10 multiplications and 4 additions/subtractions required for a PD.

Algorithm 1: Double-and-Add

```
input : $p \in E(GF(p))$;
        $k = \sum_{i=0}^{n_k-1} k_i 2^i$
output: $Q = [k]p \in E(GF(p))$
Initialise: $Q = P$;
for $i \leftarrow n_k - 2$ to 0 do
    $Q = 2Q$;
    if $k_i = 1$ then
        $Q = Q + P$
    end
end
```

Algorithm 2: Double-and-Add-Always

```
input : $p \in E(GF(p))$;
        $k = \sum_{i=0}^{n_k-1} k_i 2^i$
output: $Q = [k]p \in E(GF(p))$
Initialise: $Q = P$;
for $i \leftarrow n_k - 2$ to 0 do
    $Q[0] = 2Q[0]$;
    $Q[1] = Q[0] + P$;
    $Q[0] = Q[k_i]$;
end
```

Algorithm 3: Point Addition in Jacobian Coordinates

```
input : $P(X_1, Y_1, Z_1)$;
        $Q(X_2, Y_2, Z_2) \in GF(P)$
output: $P + Q(X_3, Y_3, Z_3) \in E(GF(p))$
$A = X_1 Z_2^2, B = X_2 Z_1^2$,
$C = Y_1 Z_2^2, D = Y_2 Z_1^3$,
$E = B - A, F = D - C$,
$X_3 = -E^3 - 2AE^2 + F^2$,
$Y_3 = -CE^3 + F(AE^2 - X_3)$,
$Z_3 = Z_1 Z_2 E$
```

Algorithm 4: Point Doubling in Jacobian Coordinates

```
input : $P(X_1, Y_1, Z_1) \in GF(P)$
output: $[2]P(X_3, Y_3, Z_3) \in E(GF(p))$
$A = 4X_1 Y_1^2$,
$B = 3X_1^2 + a_4 Z_1^4$
$X_3 = -2A + B^2$,
$Y_3 = -8Y_1^4 + B(A - X_3)$,
$Z_3 = 2Y_1 Z_1 E$
```

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. 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 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 first successful power analysis attack against an FPGA was done by Örs et al. [14] in which they attacked an elliptic curve processor and retrived the secret key.

The Double-and-Add-Always, Algorithm 2, 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 $k_i = 1$. 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 [9], Bernstein et al. introduced the twisted Edwards curves

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{4}$$

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

$$aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2 \tag{5}$$

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

2.2.1 Addition law on twisted Edwards curve

Let $(X_3 : Y_3 : Z_3)$ be the sum of the two points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ on the projective twisted Edwards curve, i.e.,

$$(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3).$$

Then

$$\begin{aligned} X_3 &= Z_1Z_2(X_1Y_2 + X_2Y_1)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2) \\ Y_3 &= Z_1Z_2(Y_1Y_2 - aX_1X_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2) \\ Z_3 &= (Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2). \end{aligned} \tag{6}$$

We note that the projective twisted Edwards curve has two singular points, $(1 : 0 : 0)$ and $(0 : 1 : 0)$, and the addition law is not defined at these points. An elliptic curve cryptosystem based on an implementation of Edwards or twisted Edwards curve should not allow either of these points as inputs.

Algorithm 5 and 6 give the PA and PD for the non SPA resistant twisted Edwards algorithms, while Algorithm 7 gives the unified formula.

Algorithm 5: 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 6: 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 7: Unified twisted Edwards point operation

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

For the separate PA and PD formulae, each PA requires 12 multiplications and 8 additions, while the PD requires 8 multiplications and 7 additions. The unified single point operation, 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 [15] 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. Software was developed in C++ to generate the instruction set and associated VHDL code for the reconfigurable processor. The elliptic curve processor (ECP) properties can be configured by the user for any characteristic p, and extension field m, as well as the respective memory sizes. In this respect it can be modified to perform a number of different algorithms in

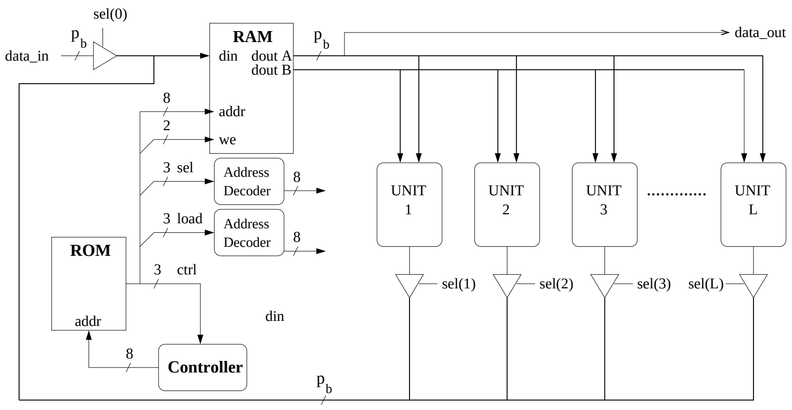

Fig. 1. Reconfigurable Elliptic Curve Processor

$GF(p)$, including the various forms of the Double-and-Add algorithms and the twisted Edwards algorithms.

3.1 Arithmetic Units

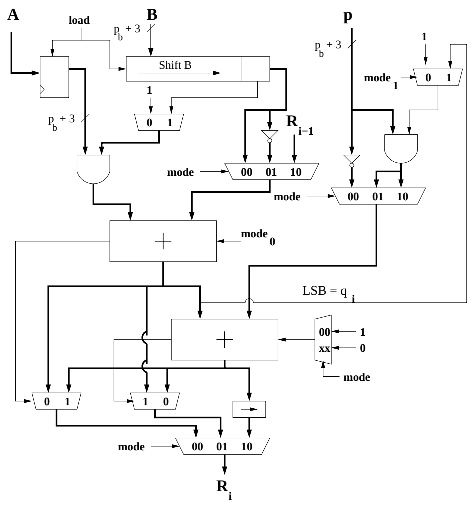

Fig. 2. Field Operation ALU

The ALUs shown in Figure 2 perform the $GF(p)$ operations described in section 2, namely the modular multiplications, additions and subtractions. Mode bits are used to select between operations.

For modular addition, the modular addition operation adds A and B in the first adder and subtracts the modulus p from the sum. To subtract the modulus from the intermediate result, the modulus is bitwise inverted and added to $(A + B)$ with the carry-in set to 1, thus performing a two's complement subtraction. The carry-out of the second adder controls which intermediate result holds the correct result. If $(A + B)$ is in the correct range, the result of the first adder is the correct result, otherwise, the second adder holds the correct result.

For modular subtraction, B is bitwise inverted and added to A with the carry-in set to 1. If the carry-out of this adder is low, the modulus must be added to give an output in the correct range.

Modular multiplication is more complex, but by using the Montgomery multiplication algorithm [16], we can compute the binary number while avoiding the need to perform a division by the modulus. Due to the large number of multiplications required for calculation by the elliptic curve processor, it is more cost effective to initially convert all values to the Montgomery domain, and then to convert them back afterward. The Montgomery modular product is defined as:

$$R = Mont(A, B, p) = AB2^{-p_b+2}(mod\ p) \tag{7}$$

where p_b is the field size in bits and $Mont$ is a montgomery multiplication. The result of a Montgomery multiplication is therefore out by a factor of 2^{-p_b+2}. To correct this reduction, the output must be Montgomery multiplied by $2^{2p_b+2}(mod\, p)$, and a value is converted back by Montgomery multiplying it by 1. For modular

Algorithm 8: Montgomery Multiplication

```
input  : $A = \sum_{i=0}^{p_b} a_i 2^i; B = \sum_{i=0}^{p_b} b_i 2^i; M = \sum_{i=0}^{p_b} p_i 2^i$
output: $R = AB2^{-p_b+2}(mod\ p)$
Initialise: $R \leftarrow 0; b_{p_b} + 1 \leftarrow 0$;
for $i \leftarrow 0$ to $p_b + 1$ do
    $q_i = R_{i-1} + b_i A(mod\ p)$;
    $R_i = (R_{i-1} + Q_i M + b_i A)/2$;
end
```

multiplication, following the process described in Algorithm 8, the inputs to the first adder are $b_i A$ and the previous result R_{i-1}. $q_i p$ is added to the sum of the first adder if the LSB of the sum (q_i) is equal to 1. A shift register is then used to check each bit of B for $b_i A$ and the final result is right shift divided by 2.

Modular multiplication 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.

5.2 GFP Controller

The ROM is generated with the use of microcode stored in Xilinx BlockROM. This is implemented to reduce the development time of the processor and to increase the flexibility of the design. A major advantage of this is that the instruction set can be updated to perform any number of operations without a need to recompile the entire processor. The instruction set to control the algorithm is stored in ROM and the instructions are processed consecutively.

For the architecture to accommodate this, mode bits are used to set the operation of the ALUs. After initially loading the elliptic curve parameters and Montgomery constants into RAM, the controller performs operations for the selected cryptographic algorithm. The initial operands and the results from the arithmetic units are stored in a RAM block. This RAM can be configured for single port, or dual port operation, thereby increasing the speed of the ECP, through the use of parallelisation.

4 Performance Results of ECC Algorithms

Table 1. Spartan3E XC3S500E-4fg320 FPGA Results

Double-and-Add				
ALU	F_{max}(Mhz)	$Area$(slices)	Pwr(mW)	$Energy$(mJ)
1	27.921	1703	88.04	2.703
2	28.333	2896	96.53	1.601
3	27.84	3988	109.58	1.415
4	26.438	4654	107.52	1.386
Double-and-Add-Always				
ALU	F_{max}(Mhz)	$Area$(slices)	Pwr(mW)	$Energy$(mJ)
1	28.539	1702	87.84	4.292
2	28.183	2897	100.56	2.608
3	27.808	3989	106.99	1.961
4	26.68	4654	105.73	1.798
twisted Edwards				
ALU	F_{max}(Mhz)	$Area$(slices)	Pwr(mW)	$Energy$(mJ)
1	28.245	1703	88.08	1.623
2	28.746	2898	98.67	1.042
3	28.01	4269	115.82	0.863
4	25.097	4654	105.93	0.83
twisted Edwards Strongly Unified				
ALU	F_{max}(Mhz)	$Area$(slices)	Pwr(mW)	$Energy$(mJ)
1	27.976	1700	88.09	2.4
2	27.852	3171	98.67	1.931
3	27.816	4553	115.82	1.926
4	25.052	4654	105.93	1.317

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 $VCCAUX$ line 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 average power value and the average time per point multiplication, as shown in Tables 2 and 3, based on the number of clock cycles and the 10 MHz clock frequency.

4.1 Computation Time

The scheduling was examined, again with a variable number of ALUs, to examine the timing of each algorithm. As described in Algorithm 1, a multiplication is executed in $p_b + 2$ clock cycles, while additions and subtractions each take 2 clock cycles. Using a key size of 192 and performing all multiplications and additions/subtractions for the Algorithms 3, 4, 5, 6 and 7, defined in Sections 2 and 4, the timing results in Tables 2 and 3 were obtained. As can be seen from the table, the number of multiplication stages required to process an algorithm decreases with an increase in parallelisation.

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. The graph in Figure 3 shows the timing results. The graph 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 graph 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 graph 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.

Table 2. Double-and-Add 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

Table 3. twisted Edwards Point Double and Point Addition Timing

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

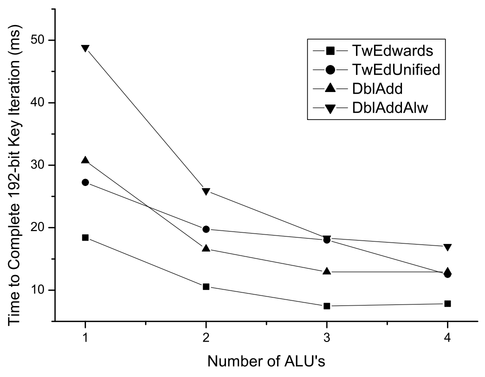

Fig. 3. Time Required to Complete an 192-bit Key Iteration

4.2 Efficiency

Although the ECP can be configured to run any number of ALUs in parallel, some operations in a particular formula are dependent on the results of other operations, which creates a limit to the amount of parallelism that can be exploited [17]. This leads to redundancy in the design, as there comes a point where the addition of extra ALUs leads to a decrease in the efficiency, and results in a small increase in speed at the cost of large increase of area. We define the efficiency as the number of multiplication operations, and therefore the number of ALUs, that can be run in parallel at each particular time stage, in relation to the overall number of ALUs available for parallel processing for a time stage. Figure 4 shows the schedule for a point operation for a four ALU strongly unified twisted Edwards, where the first eight memory addresses containing the points P, Q, a and d. As can be seen from the schedule, of the six multiplication stages

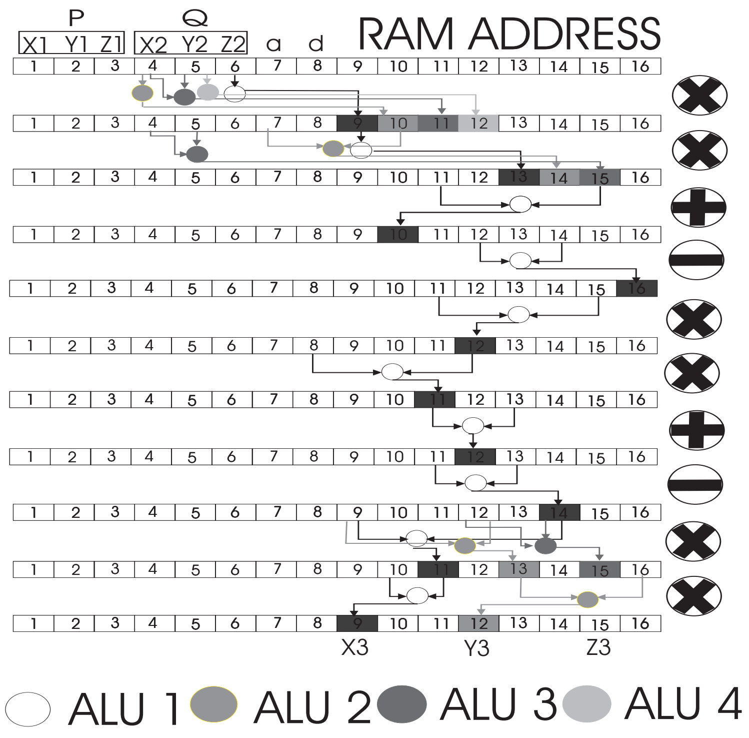

Fig. 4. twisted Edwards Strongly Unified 4ALU Point Operation

used for the point doubling and point addition, there is only one stage where all four ALUs can be used in parallel, while there are two stages where only one ALU can be in use. When this is compared against the twisted Edwards design which uses three ALUs, shown in Table 3, it results in approximately the same

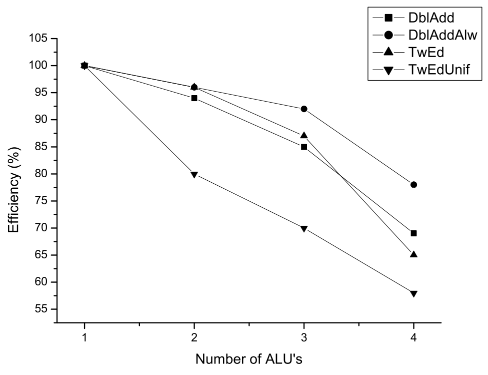

Fig. 5. Efficiency of varying numbers of ALUs.

completion time, for a much larger increase in circuit area. The efficiency for one to four ALUs for each of the four formulae is shown in Figure 5. From the graph it is clear that all of the formulae result in a drop in efficiency as more ALUs are added, with the Unified twisted Edwards having the worst case, and the Double-and-Add-Always making the best use of parallelism.

4.3 Area Time Product

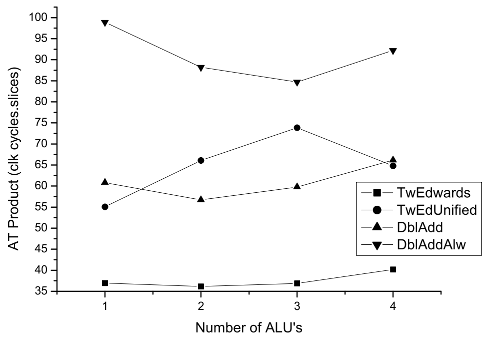

Fig. 6. Area-Time Product

The area-time (AT) product was calculated to get a representation of any speed increase against the increase in size, as shown in Figure 6. 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 be 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.

Acknowledgements

This material is based upon works supported by the Science Foundation Ireland under Grant No. 06/MI/006.

References

1. Victor S Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
2. Neal Koblitz. Elliptic curve cryptosystems. In *Mathematics of Computation*, volume 48, pages 203–209, 1987.
3. R. Rivest, A. Shamir, and L. M. Adleman. Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
4. Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In *PKC '00: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, pages 446–465, London, UK, 2000. Springer-Verlag.
5. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.
6. Paul Kocher, Joshua Ja E, and Benjamin Jun. Differential power analysis. pages 388–397. Springer-Verlag, 1999.
7. Eric Brier and Marc Joye. Weierstraßelliptic curves and side-channel attacks. In *PKC '02: Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, pages 335–345, London, UK, 2002. Springer-Verlag.

8. H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393 –422, 2007.
9. Daniel Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In *Progress in Cryptology AFRICACRYPT 2008*, pages 389–405, 2008.
10. Donald Ervin Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms of *Addison-Wesley series in computer science and information processing*. Addison-Wiley, third edition, 2001.
11. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 292–302, London, UK, 1999. Springer-Verlag.
12. I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.
13. Gerardo Orlando and Christof Paar. A scalable gf(p) elliptic curve processor architecture for programmable hardware. In *Lecture Notes In Computer Science*, volume 2162, pages 348 –363, 2001.
14. Siddika Berna rs, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an fpga first experimental results. In *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, volume 2279 of *Lecture Notes in Computer Science*, pages 35–50. Springer-Verlag, 2003.
15. A. Byrne, E. Popovici, and W.P. Marnane. Versatile processor for gf(pm) arithmetic for use in cryptographic applications. In *Computers & Digital Techniques, IET*, volume 2, pages 253–264, July 2008.
16. P. Montgomery. Modular multiplication without trial division. In *Mathematics of Computation*, volume 44, pages 519–521, 1985.
17. Robert A. Walker and Samit Chaudhuri. Introduction to the scheduling problem. volume 12, pages 60–69, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.