FPGA MONTGOMERY MULTIPLIER ARCHITECTURES – A COMPARISON

Ciaran McIvor, Máire McLoone, John V McCanny
The Institute of Electronics, Communications and Information Technology
Queen's University of Belfast, Northern Ireland
c.mcivor@ee.qub.ac.uk, maire.mcloone@ee.qub.ac.uk, j.mccanny@ee.qub.ac.uk

ABSTRACT

Novel FPGA architectures for the SOS, CIOS and FIOS Montgomery multiplication algorithms are presented. The 18×18-bit multipliers and fast carry look-ahead logic embedded within the Xilinx Virtex2 Pro family of FPGAs are used to perform the ordinary multiplications and additions required by these algorithms. A detailed analysis is given, highlighting the advantages and weaknesses of each of these architectures when implemented in hardware. This shows that the CIOS multiplier architectures perform best overall, with the performance gap between this and the other options increasing as the word size used decreases. In addition, the SOS multipliers outperform the FIOS multipliers for larger word sizes, but vice versa as the word size decreases. It is also shown that one can tailor the multiplier architectures to be area efficient, time efficient or a mixture of both, by choosing a particular word size.

1. INTRODUCTION

Modular multiplication is important in modern cryptography, as it forms an integral part in many modern public-key cryptographic primitives, such as those based on the integer factorization problem [1] or the elliptic curve discrete logarithm problem [2,3]. Montgomery [4], eliminating the need for trial division by the modulus, proposed a very efficient modular multiplication technique. Here, the operands are converted to a residue number system and modular multiplication is performed using a series of additions and divisions by a power of 2. Thus, this method is well suited to hardware implementation.

Koc *at al.* [5] developed this Montgomery multiplication technique and described a new class of algorithms, namely, the Separated Operand Scanning (SOS) method, the Coarsely Integrated Operand Scanning (CIOS) method, the Finely Integrated Operand Scanning (FIOS) method, the Finely Integrated Product Scanning (FIPS) method, and the Coarsely Integrated Hybrid Scanning (CIHS) method. These algorithms break up the modular multiplications into iterative series of ordinary word additions and multiplications. The word size can be chosen arbitrarily but should ideally reflect the properties of the target technology into which the algorithms are to be implemented. For instance, Koc *et al.* [5] presented software implementations and described detailed comparisons of these algorithms operating on an Intel Pentium-60 Linux system. In this case, they chose a word size of 32-bits. On their general-purpose processor, the CIOS method performed best overall, with the SOS and FIOS methods also performing well.

To the authors' knowledge, there have been no direct hardware implementations or comparisons of these algorithms. The motivation, therefore, of the work presented in this paper has been to investigate FPGA hardware implementations and undertake comparisons of the operand scanning (SOS, CIOS and FIOS) algorithms described. To achieve this, modern Xilinx Virtex2 Pro FPGAs have been used as the target technology, as these devices offer several key features, which we believe are well suited to the implementation of such algorithms. For example, the fast carry chains and 18×18-bit multiplier blocks embedded on these devices can be used to perform the ordinary word additions and multiplications required by these algorithms. These issues will be discussed further in Sections 2 and 3.

2. THE XILINX VIRTEX2 PRO FAMILY OF FPGAs

This section provides relevant information about some of the features of the Xilinx Virtex2 Pro family of FPGAs [6], which have been used to develop the SOS, CIOS and FIOS modular multiplier architectures described in Section 3. The Virtex2 Pro family has been developed using a 0.13 μm CMOS nine-layer copper process. The main processing elements in these FPGAs are the Configurable Logic Blocks (CLBs), which are arranged in a rectangular array across the face of the device. A CLB comprises four slices each of which contains two Look-Up Tables (LUTs), two flip-flops and fast carry look-ahead logic. Several columns of dedicated 18×18-bit embedded multiplier blocks and Block SelectRAM are also distributed across the device. Input/Output pins surround the array of CLBs. Some Virtex2 Pro devices also contain up to four PowerPC processor blocks.

For the modular multiplier architectures, we are mainly interested in the arithmetic functions on the Virtex2 Pro devices, namely, the fast look-ahead carry logic chains and the dedicated 18×18-bit multiplier blocks. Each CLB contains two separate carry chains and the height of these chains is two bits per slice. The carry chains run from the bottom to the top of a CLB column. Therefore, their length is limited to the height of a single CLB column. Each multiplier block can be reconfigured to support up to 18×18-bit unsigned multiplication or 17×17-bit signed multiplication. They can also be cascaded to create larger bit length multipliers. For the modular multiplier architectures described, 16×16-bit unsigned multipliers are cascaded to create larger multipliers, as discussed in Section 3.

3. OPERAND SCANNING MODULAR MULTIPLICATION COMPARISONS

3.1 The SOS Algorithm Architectures

The SOS method was described by Koc *et al.* [5] as one of a number of flexible alternatives to the ordinary Montgomery

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 2, 2008 at 16:29 from IEEE Xplore. Restrictions apply.

multiplication algorithm [4]. The SOS algorithm calculates the Montgomery product $A*B*r^{-1}(mod\ n)$ of two n-residues A and B, where n is the k-bit modulus and $r=2^k$. The SOS algorithm is summarised below. Here w is the word size, s is the number of words, $W=2^w$ and $(r*r^{-1}–n*n`)=1$.

Algorithm 1: *SOS Montgomery Multiplication (A, B, n, n`(0))*

```
t = 0;
for i = 0 to s–1  loop
   C = 0;
   for j = 0 to s–1  loop
      (C, S) = t(i+j) + A(j)*B(i) + C; t(i+j) = S;
   end loop;
   t(i+s) = C;
end loop;
for i = 0 to s–1  loop
   C = 0; m = t(i)*n`(0) mod W;
   for j = 0 to s–1  loop
      (C, S) = t(i+j) + m*n(j) + C; t(i+j) = S;
   end loop;
   ADD [t(i+s), C];
end loop;
for j = 0 to s loop
   t(j) = t(j+s);
end loop;
```

Here, the operands A, B, n and $n`$ take the form of $s$ $w$-bit words. Only the least significant word of $n`$ is needed. The variable (C, S) consists of two w-bit words of which C is the most significant and S is the least significant word. The variable t, which is used to accumulate the partial products of $A*B$ and $m*n$, comprises $2s+1$ w-bit words. A conditional subtraction is normally required after the algorithm has terminated if t is greater than or equal to n. However, since this final subtraction is identical for all the algorithms described, it is omitted here in order to obtain a better analysis of the advantages and weaknesses of the algorithms when implemented in hardware.

The main arithmetic functions to be performed in Algorithm 1 include word size multiplications and several variable size full additions. The additions are computed using the fast carry chains. To perform the multiplications, we need to develop different word size multipliers, by cascading numerous 16×16-bit unsigned multipliers. The larger multipliers are developed in a systematic fashion. For instance, the partial products of the 32×32-bit multiplier are calculated using the 16×16-bit unsigned multiplier blocks. These partial products are then added together using the fast look-ahead carry chains to obtain the 64-bit final product. This process is continued until the desired multiplier size is attained. The calculation and addition of the partial products is a fully pipelined process, designed to take full advantage of the small critical path delay of the 16×16-bit multiplier blocks and the fast carry chains. Therefore it takes 3, 5, and 7 clock cycles to complete a 32, 64, and 128 bit multiplication respectively.

The word size w can be chosen arbitrarily. For our 128-bit and 256-bit SOS, CIOS and FIOS multiplier architectures, we have chosen w as 16, 32, 64 and 32, 64, 128 respectively. For small w, relatively smaller multipliers and carry chains can be used to calculate the multiplications and additions in Algorithm 1, implying a smaller silicon area usage. For instance, for a 128-bit SOS multiplier using a 32-bit word size, we use the 32×32-bit multiplier, as shown in Figure 1. However, for small w a greater amount of clock cycles are required to compute a SOS multiplication, as the number of words s will be larger. Therefore, more iterations of the *for* loops in Algorithm 1 are needed, implying a lower data throughput rate. For larger w, less clock cycles are needed and a higher throughput rate can be attained, albeit at the expense of a larger silicon area.

By using the cascaded multipliers described to carry out the word size multiplications, we have been able to develop 128-bit and 256-bit SOS multiplier architectures for varying word sizes. Figure 1 shows a block diagram of the 128-bit SOS multiplier architecture using a 32-bit word size. The inputs are registered into the FPGA 32-bits per clock cycle over 4 cycles (8-bits per cycle over 4 cycles for $n`$). The control unit then determines the order in which the multiplications and additions are performed, according to Algorithm 1. Here, the maximum addition needed is 128-bits. This is due to the *ADD[t(i+s),C]* function in Algorithm 1.

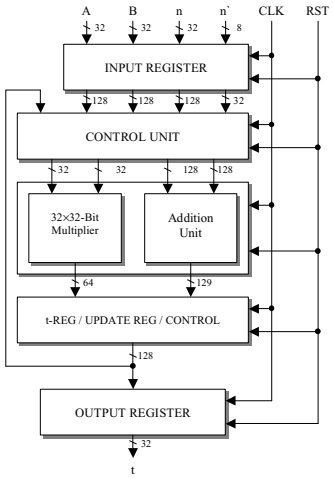

Figure 1: 128-Bit SOS Architecture

The *t-REG/UPDATE REG/CONTROL* component stores the intermediate values of the variable t and the results of the other multiplications and additions, which are then fed back into the control unit to be re-used as inputs to the 32×32-bit multiplier or Addition Unit. The *t-REG/UPDATE REG/CONTROL* component also performs the trivial *mod* and *right shift* operations in Algorithm 1. Once the loops in Algorithm 1 have terminated, t is output from the chip 32-bits at a time over 4 clock cycles.

The architectures described have been captured in VHDL and then used to create implementations using the Xilinx Virtex2 Pro family of FPGAs. Table 1 provides performance results for each of these. The 128-bit and 256-bit SOS multipliers have been implemented in the XC2VP50-7-ff1517 and XC2VP125-7-ff1696 FPGAs respectively.

The performance results given in Table 1 highlight the time-area trade-offs by using different word sizes. For instance, by using a 32-bit rather than a 64-bit word size for the 128-bit SOS architecture, then there is a corresponding fall of 28.5% and 58.5% in the silicon area and data throughput rate respectively. This trend is similar for the 256-bit architecture. Therefore, a larger word size would be preferable for high throughput applications and a smaller word size for a lower silicon area solution.

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 2, 2008 at 16:29 from IEEE Xplore. Restrictions apply.

Words×Wordsize (s×w)	Clock (MHz)	No. Slices	No. Mult 18×18	No. Clock Cycles	Throughput Rate (Mb/s)
2×64 Bits (128)	128.83	1,701	16	73	225.90
4×32 Bits (128)	129.63	1,216	4	177	93.75
8×16 Bits (128)	135.14	1,022	1	449	38.52
2×128 Bits (256)	76.17	4,663	64	93	209.66
4×64 Bits (256)	79.66	2,743	16	249	81.90
8×32 Bits (256)	83.13	2,070	4	609	34.95

Table 1: Performance Results for 128-bit and 256-bit SOS Multiplier Architectures

Also, the critical path delay through the multiplier architectures is somewhat dependent on the maximum addition to be performed in Algorithm 1. As mentioned, for the 128-bit architecture this addition is 128-bits in length independent of the word size chosen. This is reflected in the critical delay through the multiplier, which falls by only 4.7% using a 16-bit rather than a 64-bit word size for the 128-bit SOS architecture. The corresponding fall in the delay is 8.4% using a 32-bit rather than a 128-bit word size for the 256-bit SOS architecture. The maximum addition in this case is 256-bits. As will be shown, the critical delay through the CIOS and FIOS multiplier architectures is more responsive to changes in the word size used, as the maximum additions needed in those architectures are dependent on the word size chosen.

3.2 The CIOS Algorithm Architectures

The CIOS algorithm [5] is given as Algorithm 2 below. Again w is the word size, s is the number of words, and $W=2^w$.

Algorithm 2: *CIOS Montgomery Multiplication (A, B, n, n`(0))*

```
t = 0;
for i = 0 to s–1  loop
   C = 0;
   for j = 0 to s–1  loop
      (C, S) = t(j) + A(j)*B(i) + C; t(j) = S;
   end loop;
   (C, S) = t(s) + C; t(s) = S; t(s+1) = C; C = 0;
   m = t(0)*n`(0) mod W; (C, S) = t(0) + m*n(0);
   for j = 0 to s–1  loop
      (C, S) = t(j) + m*n(j) + C; t(j–1) = S;
   end loop;
   (C, S) = t(s) + C; t(s–1) = S; t(s) = t(s+1) + C;
end loop;
```

Here the variable t is only $s+2$ words in length, leading to a reduction in the silicon area of the CIOS multipliers over the SOS multiplier architectures. The additions and multiplications are again carried out using the fast carry chains and the cascaded multipliers described, respectively. This led us to develop CIOS multiplier architectures similar to that shown in Figure 1. However, here, the maximum addition needed is dependent on the word size and corresponds to the additions $(C,S)=t(j)+A(j)*B(i)+C$, $(C,S)=t(0)+m*n(0)$ and $(C,S)=t(j)+m*n(j)+C$ in Algorithm 2. So for a 16-bit word size the maximum addition is 32-bits in length in comparison to a maximum addition of 128-bits for a 64-bit word size.

Again, the CIOS architectures have been captured in VHDL and then used to create implementations using the Xilinx Virtex2 Pro family of FPGAs. Table 2 provides performance results for these architectures. The 128-bit and 256-bit CIOS multipliers have been implemented in the XC2VP50-7-ff1517 and XC2VP125-7-ff1696 FPGAs respectively.

Words×Wordsize (s×w)	Clock (MHz)	No. Slices	No. Mult 18×18	No. Clock Cycles	Throughput Rate (Mb/s)
2×64 Bits (128)	114.33	1,535	16	68	215.20
4×32 Bits (128)	126.66	936	4	162	100.08
8×16 Bits (128)	163.53	717	1	310	67.52
2×128 Bits (256)	67.90	4,376	64	88	197.53
4×64 Bits (256)	91.62	2,140	16	234	100.23
8×32 Bits (256)	101.87	1,433	4	582	44.81

Table 2: Performance Results for 128-bit and 256-bit CIOS Multiplier Architectures

The fact that the maximum addition needed in Algorithm 2 is dependent on the word size is reflected in the critical path of the multipliers. For example, there is a drop of 30.1% in the critical delay through the 128-bit CIOS multiplier using a 16-bit rather than a 64-bit word size. Also, there is a drop in the delay through the 256-bit CIOS multiplier architecture of 33.3% using a 32-bit rather than a 128-bit word size. This leads to a higher data throughput rate for the CIOS multipliers over the SOS multipliers for smaller word sizes, as shown in Figure 2. Also, because the variable t is only $s+2$ words in length, then there is a notable saving in silicon area over the SOS architectures, as shown in Figure 2.

3.3 The FIOS Algorithm Architectures

The FIOS algorithm [5] is given as Algorithm 3 below. Again w is the word size, s is the number of words, and $W=2^w$.

Algorithm 3: *FIOS Montgomery Multiplication (A, B, n, n`(0))*

```
t = 0;
for i = 0 to s–1  loop
   (C, S) = t(0) + A(0)*B(i); ADD [t(1), C]; m = S*n`(0) mod W;
   (C, S) = S + m*n(0);
   for j = 1 to s–1  loop
      (C, S) = t(j) + A(j)*B(i) + C; ADD [t(j+1), C];
      (C, S) = S + m*n(j); t(j–1) = S;
   end loop;
   (C, S) = t(s) + C; t(s–1) = S; t(s) = t(s+1) + C; t(s+1) = 0;
end loop;
```

By using the cascaded multipliers described and the fast carry chains to carry out the arithmetic functions in Algorithm 3, we have been able to develop FIOS multiplier architectures similar to that shown in Figure 1. Table 3 provides performance results for these architectures. The 128-bit and 256-bit FIOS multipliers have been implemented in the XC2VP50-7-ff1517 and XC2VP125-7-ff1696 FPGAs respectively.

Again, the variable t is stored in $s+2$ words here. Therefore, as in the CIOS multiplier case, there is a lower silicon area usage over the SOS architectures, as shown in Figure 2. However, the maximum addition required here is over and above that needed in Algorithms 1 and 2. This maximum addition is dependent on the *ADD[t(1),C]* function in Algorithm 3 and is dependent on the word size chosen.

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 2, 2008 at 16:29 from IEEE Xplore. Restrictions apply.

Words×Wordsize (s×w)	Clock (MHz)	No. Slices	No. Mult 18×18	No. Clock Cycles	Throughput Rate (Mb/s)
2×64 Bits (128)	87.55	1,727	16	70	160.09
4×32 Bits (128)	105.35	1,054	4	166	81.24
8×16 Bits (128)	111.67	836	1	318	44.95
2×128 Bits (256)	55.66	4,770	64	90	158.32
4×64 Bits (256)	65.22	2,491	16	238	70.15
8×32 Bits (256)	66.97	1,709	4	590	29.06

Table 3: Performance Results for 128-bit and 256-bit FIOS Multiplier Architectures

For a 128-bit FIOS multiplier using a 64-bit word size, a 192-bit maximum addition is required. This falls to a 144-bit addition for a 16-bit word size, which is still larger than that required in Algorithms 1 and 2. Thus, there is a higher critical delay through the FIOS multipliers and hence a lower data throughput rate is achievable here, as shown in Figure 2.

In order to provide a general analysis of which of the multiplier architectures performs best overall, the Area/Throughput Rate product has been calculated, as shown in Figure 2. Smaller values indicate better performance. For larger word sizes the SOS architectures outclass the FIOS architectures, but vice versa as the word size decreases.

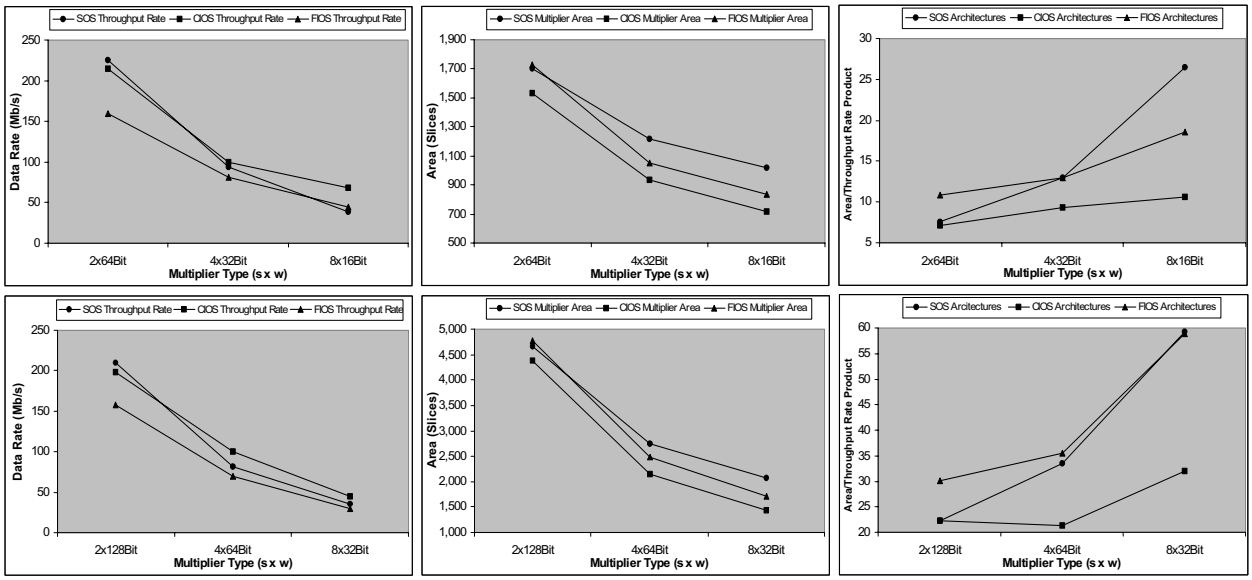

Figure 2: Architecture Comparisons

However, it can clearly be seen that the CIOS architectures outperform both the SOS and FIOS multipliers, especially as the word size decreases.

4. CONCLUSION

In this paper we have presented new FPGA architectures for the SOS, CIOS and FIOS Montgomery multiplication algorithms. The embedded 18×18-bit multipliers and fast carry look-ahead logic located on the Xilinx Virtex2 Pro family of FPGAs were used to perform the ordinary multiplications and additions required by these algorithms. A detailed analysis was given of these architectures, highlighting the advantages and weaknesses of each of the aforementioned algorithms when implemented in hardware. The results obtained show that the CIOS multiplier architectures performed best overall with the performance gap increasing as the word size used decreased. Also, the SOS multipliers outperformed the FIOS multipliers for larger word sizes but vice versa as the word size decreased. It was also found that one can tailor the multiplier architectures to be area efficient, time efficient or a mixture of both, by choosing a particular word size. These multiplier architectures can be used in elliptic curve cryptosystems over GF(p), which require modular field multiplication to perform elliptic curve point addition and doubling. They can also be readily migrated to other FPGA families such as Altera's Stratix devices [7], which possess similar arithmetic functions to the Virtex2 Pro devices. In this case, we would expect to achieve analogous results to those detailed here.

ACKNOWLEDGMENT

Amphion Semiconductor Ltd. and a Northern Ireland Department of Learning postgraduate studentship in the form of a CAST award have funded this research.

REFERENCES

[1] Rivest, R.L., Shamir, A., Adleman, L.: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM, 21(2): 120-126, February 1978.

[2] Miller, V.S.: "Use of Elliptic Curves in Cryptography". Proc. Advances in Cryptology (Crypto' 85), pp. 417-426, 1986.

[3] Koblitz, N.: "Elliptic Curve Cryptosystems". Math. Computing, Vol. 48, pp. 203-209, 1987.

[4] Montgomery, P.L.: "Modular Multiplication without Trial Division". Math. Computation, Vol. 44, pp. 519-521, 1985.

[5] Koc, C.K., Acar, T., Kaliski, B.S.: "Analyzing and Comparing Montgomery Multiplication Algorithms". IEEE Micro, Vol. 16, No. 3, pp. 26-33, June 1996.

[6] Xilinx, Inc.: http://www.xilinx.com, "Xilinx Virtex2 Pro Data Sheets".

[7] Altera Corporation: http://www.altera.com, "Stratix FPGA Literature".

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 2, 2008 at 16:29 from IEEE Xplore. Restrictions apply.