A Bit–Serial Implementation of the International Data Encryption Algorithm IDEA

M.P. Leong, O.Y.H. Cheung, K.H. Tsoi and P.H.W. Leong
{mpleong,yhcheung,khtsoi,phwl}@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong

Abstract

A high–performance implementation of the International Data Encryption Algorithm (IDEA) is presented in this paper. Using a novel bit–serial architecture to perform multiplication modulo $2^{16}+1$, the implementation occupies a minimal amount of hardware. The bit–serial architecture enabled the algorithm to be deeply pipelined to achieve a system clock rate of 125MHz on a Xilinx Virtex XCV300–6, delivering a throughput of 500Mb/sec. With a XCV1000–6 device, the estimated performance is 2Gb/sec, three orders of magnitude faster than a software implementation on a 450MHz Intel Pentium II. This design is suitable for applications in on–line encryption for high–speed networks.

1 Introduction

Cryptography is concerned with the transfer of information between parties so that only the intended parties can read the data. Despite an assumption that an adversary may have full knowledge of the algorithms used, and has access to the media where data is transmitted, it is desired that the retrieval of data without knowledge of a secret piece of information called a key is intractable.

We believe that cryptography is an ideal application for Field–programmable Custom Computing Machines (FCCMs), since they offer the following advantages over VLSI technologies

- it is possible to use the same FCCM hardware for many different cryptographic protocols
- Moore's law continues to offer improved silicon technology at exponential rates which is available to FCCM designers without the costly manufacturing process required in VLSI
- it is possible to specialize the hardware to an extent not possible in VLSI devices to improve performance
- the reconfigurable nature makes it feasible to attempt designs employing more sophisticated algorithms which leads to an improvement in performance.

The Data Encryption Standard (DES) algorithm has been a popular secret key encryption algorithm and is used in many commercial and financial applications. Although introduced in 1976, it has proved resistant to all forms of cryptanalysis. However, its key size is too small by current standards and its entire 56 bit key space can be searched in approximately 22 hours [1].

In 1990, Lai and Massay introduced an iterated block cipher known as Proposed Encryption Standard (PES) [2]. The same authors, joined by Murphy, proposed a modification of PES called Improved PES (IPES) [3], which improves the security of the original algorithm against differential analysis and truncated differentials [4, 5, 6]. In 1992, IPES was commercialized and was renamed the International Data Encryption Algorithm (IDEA). Some believe that, to date, the algorithm is the best and the most secure block algorithm available to the public [7].

Although IDEA involves only simple 16–bit operations, software implementations of this algorithm still cannot offer the encryption rate required for on–line encryption in high–speed networks. Ascom's implementation of IDEA (Ascom are the holders of the patent on the IDEA algorithm) achieves 0.37×10^6 encryptions per seconds, or a equivalent encryption rate of 23.53Mb/sec, on an Intel Pentium II 450MHz machine. Our optimized software implementation running on a Sun Enterprise E4500 machine with twelve 400MHz Ultra–IIi processor, performs 2.30×10^6 en-

cryptions per second or a equivalent encryption rate of 147.13Mb/sec, still cannot be applied to applications such as encryption for 155Mb/sec Asynchronous Transfer Mode (ATM) networks.

Hardware implementations offer significant speed improvements over software implementations by exploiting parallelism among operators. In addition, they are likely to be cheaper, have lower power consumption and smaller footprint in embedded applications than a high speed software implementation. A paper design of an IDEA processor which achieves 528Mb/sec on four XC4020XL devices was proposed by Mencer et. al. [8]. The first VLSI implementation of IDEA was developed and verified by Bonnenberg et. al. in 1992 using a 1.5 μm CMOS technology [9]. This implementation had an encryption rate of 44Mb/sec. In 1994, VINCI, a 177Mb/sec VLSI implementation of the IDEA algorithm in 1.2 μm CMOS technology, was reported by Curiger et. al. [10, 11]. A 355Mb/sec implementation in 0.8 μm technology of IDEA was reported in 1995 by Wolter et. al. [12]. The fastest single chip implementation of which we are aware is a 424Mb/sec implementation of 0.7 μm technology by Salomao et. al. [13]. A commercial implementation of IDEA called the IDEACrypt coprocessor, developed by Ascom achieves 300Mb/sec [14].

In this paper, a Xilinx Virtex XCV300–6 based implementation of the IDEA algorithm is described with a throughput of 500Mb/sec. Furthermore, with a XCV1000–6 device, the estimated performance is 2Gb/sec. This design is faster than all VLSI implementations mentioned above. The implementation employs a novel bit–serial architecture which offers the following advantages

- high degree of fine–grain parallelism
- scalable so that throughput and area tradeoffs can be addressed
- high clock rate
- compact implementation.

Applications of this design include Virtual Private Networks (VPNs) and embedded encryption/decryption devices.

This paper is organized as follows. In Section 2 the IDEA algorithm as well as algorithms for multiplication modulo $2^n + 1$ are described. In Section 3 the bit–serial implementation of IDEA is presented. In Section 4 results are given. Conclusions are drawn in Section 5.

2 The IDEA Algorithm

IDEA belongs to a class of cryptosystems called secret–key cryptosystems which is characterized by the symmetry of encryption and decryption processes, and the possibility of implying the decryption key from the encryption key and vice versa. IDEA takes 64–bit plaintext inputs and produces 64–bit ciphertext outputs using a 128–bit key.

The design philosophy behind IDEA is mixing operations from different algebraic groups including XOR, addition modulo 2^{16}, and multiplication modulo the Fermat prime $2^{16} + 1$. All these operations work on 16–bit sub–blocks.

The IDEA block cipher [7] (depicted in Figure 1) consists of a cascade of eight identical blocks known as rounds, followed by a half–round or output transformation. In each round, XOR, addition and modular multiplication operations are applied. IDEA is believed to be of strong cryptographic strength because its primitive operations are of three distinct algebraic groups of 2^{16} elements, multiplication modulo $2^{16} + 1$ provides desirable statistical independence between plaintext and ciphertext, and its property of having iterative rounds made differential attacks difficult.

The encryption process is as follows, the 64–bit plaintext is divided into four 16–bit plaintext sub–blocks, X_1 to X_4. The algorithm converts the plaintext blocks into ciphertext blocks of the same bit–length, similarly divided into four 16–bit sub–blocks, Y_1 to Y_4. 52 16–bit subkeys, $Z_i^{(r)}$, where i and r are the subkey number and round number respectively, are computed from the 128–bit secret key. Each round uses six subkeys and the remaining four subkeys are used in the output transformation. The decryption process is essentially the same as the encryption process except that the subkeys are derived using a different algorithm [7].

The algorithm for computing the encryption subkeys (called the key schedule) involves only logical rotations. Order the 52 subkeys as $Z_1^{(1)}, \ldots, Z_6^{(1)}, Z_1^{(2)}, \ldots, Z_6^{(2)}, \ldots, Z_1^{(8)}, \ldots, Z_6^{(8)}, Z_1^{(9)}, \ldots, Z_4^{(9)}$. The procedure begins with partitioning the 128–key secret key Z into eight 16–bit blocks and assigning them directly to the first eight subkeys. Z is then rotated left by 25 bits, partitioned into eight 16–bit blocks and again assigned to the next eight subkeys. The process continues until all 52 subkeys are assigned. The decryption subkeys $Z_i'^{(r)}$ can be computed from the encryption subkeys with reference to Table 1.

In electronic codebook (ECB) mode [7], the data

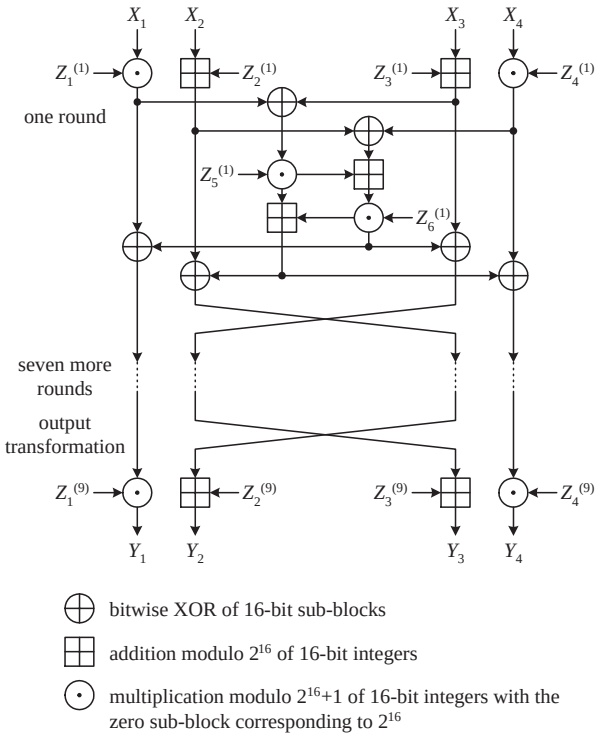

Figure 1: Block diagram of the IDEA algorithm.

	$r = 1$	$2 \le r \le 8$	$r = 9$
$Z'^{(r)}_1$	$(Z_1^{(10-r)})^{-1}$	$(Z_1^{(10-r)})^{-1}$	$(Z_1^{(10-r)})^{-1}$
$Z'^{(r)}_2$	$-Z_2^{(10-r)}$	$-Z_3^{(10-r)}$	$-Z_2^{(10-r)}$
$Z'^{(r)}_3$	$-Z_3^{(10-r)}$	$-Z_2^{(10-r)}$	$-Z_3^{(10-r)}$
$Z'^{(r)}_4$	$(Z_4^{(10-r)})^{-1}$	$(Z_4^{(10-r)})^{-1}$	$(Z_4^{(10-r)})^{-1}$
$Z'^{(r)}_5$	$Z_5^{(9-r)}$	$Z_5^{(9-r)}$	N/A
$Z'^{(r)}_6$	$Z_6^{(9-r)}$	$Z_6^{(9-r)}$	N/A

Table 1: IDEA decryption subkeys $Z'^{(i)}_r$ derived from encryption subkeys $Z_r^{(i)}$. $-Z_i$ and Z_i^{-1} denote additive inverse modulo 2^{16} and multiplicative inverse $2^{16}+1$ of Z_i respectively.

dependencies of the IDEA algorithm have no feedback paths. Additionally, in practice, latencies of order of microseconds are acceptable. These feature make the algorithm suitable to be implemented as a deep bit–serial pipeline.

2.1 Multiplication Modulo $2^n + 1$

Of the basic operations used in the IDEA algorithm, multiplication modulo $2^{16}+1$ is the most complicated and occupies most of the hardware. Curiger et. al. [15] described and compared several VLSI architectures for multiplication modulo 2^n+1 and found that an architecture proposed by Meier and Zimmerman [16], using modulo 2^n adders with bit–pair recoding offers the best performance.

The pseudocode for the modular multiplication operation by module 2^n adders using bit–pair recoding is as follows.

```
1  uint16 mulmod(uint16 x, uint16 y)
2  {
3      uint32 t;
4      x = (x - 1) & 0xFFFF;
5      y = (y - 1) & 0xFFFF;
6      t = (uint32) x * y + x + y + 1;
7      x = t & 0xFFFF;
8      y = t >> 16;
9      x = (x - y) + (x <= y);
10     return x;
11 }
```

This algorithm requires a total of six additions and subtractions, one 16–bit multiplication and one comparison. However, in IDEA one of the operands of a modular multiplication operation is always a subkey, so the second subtraction can be eliminated if the associated subkeys are pre–decremented.

3 Implementation

3.1 Bit–Serial Architecture

Bit–serial architectures are characterized by the property that operators perform their computations in a bitwise fashion and communications between operators are multiplexed in time over a single wire. Dataflow begins with either the least significant bit or the most significant bit, but the former is more commonly used due to its compatibility with two's complement arithmetic. In a typical bit–serial implementation, each variable is associated with a control signal which is set high only when the first bit

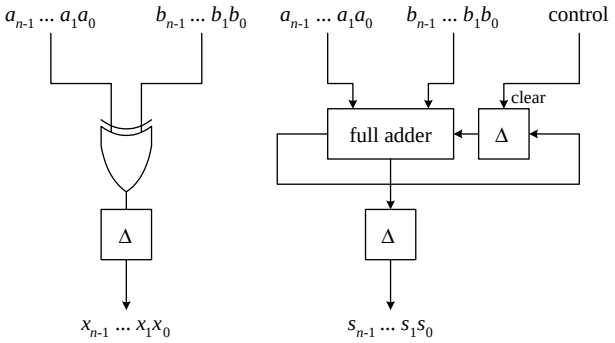

Figure 2: Bit–serial XOR and addition operators.

is transferred along associated data bus. To reduce area, control signals can be shared among the variables. Since bit–serial operators usually require the first bits of their operands to enter the operators on the same clock cycle, appropriate stage latches must be inserted for time–alignment [17].

Two of the primitive operators used in IDEA, namely XOR and addition modulo 2^{16}, can be implemented in a bit–serial fashion using the circuits shown in Figure 2. These two operators have latencies of one clock cycle and are capable of taking consecutive bit–serial operands. The multiplication modulo $2^{16}+1$ operator has a latency of 35 clock cycles. The corresponding pipelined datapath for one round of IDEA is illustrated in Figure 3. For the best area–efficiency, stage latches and constants are implemented using Virtex SRL16E primitives [18, 19]. More specifically, a constant is implemented as a SRL16E primitive, with its output connected to its input to form a cyclic shift register.

3.2 Multiplication Modulo $2^{16}+1$

As described in Section 2.1, multiplication modulo $2^{16}+1$ is the most critical operation in the IDEA algorithm. Choosing a suitable multiplier is therefore a crucial design issue.

An $N \times N$–bit multiplier generates a $2N$–bit result, and requires $2N$ cycles to complete. Thus, throughput of bit–serial multipliers are restricted because the minimum interval between consecutive multiplications must be at least $2N$ cycles. In the IDEA algorithm one of the operands of every modular multiplication is a subkey and treated as a constant.

Recall in the modular multiplication algorithm that the intermediate result `t` is divided into two portions (lines 6 to 8, in Section 2.1). The two portions are

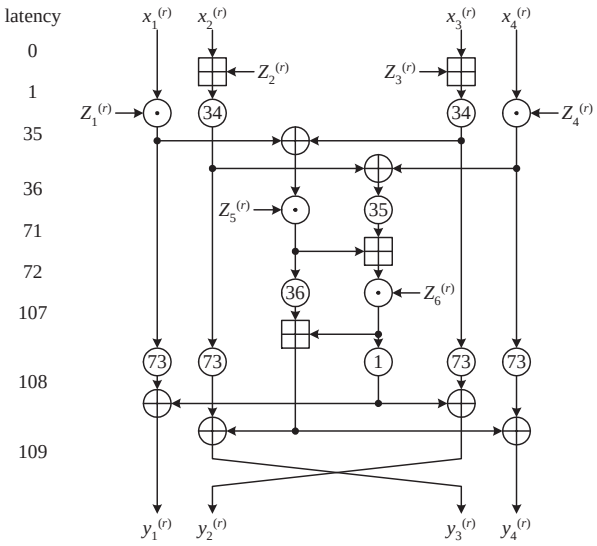

Figure 3: Pipelined datapath for one round of IDEA.

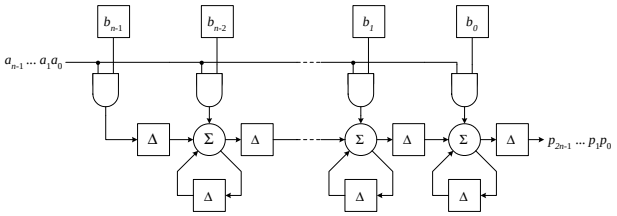

Figure 4: Lyon's serial–parallel multiplier.

respectively the upper and lower 16 bits of the double–word, which are operands to subsequent operations. A design that computes the upper and lower words of `t` independently is desirable, allowing all the inputs, outputs and intermediate variables of the operator to be 16–bit long. Using this scheme and duplicating hardware, the throughput of a modular multiplication operation can be doubled.

A modified version of Lyon's serial–parallel multiplier [20] was developed which addresses this problem. The original design of Lyon's multiplier is shown in Figure 4. To generate two 16–bit results in 16 cycles, the throughput of the multiplier must be doubled. We achieved this by duplicating the hardware for multiplication, as illustrated in Figure 5. Registers storing the constant are shared among the two multiplication pipelines. The outputs p and q correspond to the results of two consecutive multiplications, where the two 32–bit long variables have a time–difference of 16 cycles. The control signal, which is high one clock cycle

before the least significant bit enters the module, toggles the control register. The vector of input variables $a_{n-1} \ldots a_1 a_0$ is consequently redirected into the two multiplication pipelines alternately. While the vector is being redirected to one pipeline, logic zero enters the other pipeline carrying out zero–padding. A timing diagram of the modified multiplier is shown in Figure 6.

To obtain the time–aligned upper and lower words of `t`, a 16 stage shift register is required. The input and output of the shift register are the upper and lower words of `t` respectively, 16 cycles after `t` is valid. In the implementation the shift register is implemented as a SRL16E [18] primitive. The complete architecture for the modular multiplication operation is shown in Figure 7. Upon initialization, the subkey associated with the operator is passed into the operator bit–serially. The pre–decremented subkey is shifted into the registers of the multiplier, and at the same time stored into the SRL16E primitive responsible for key storage.

Utilizing the idea of multiple pipelines, the modular multiplication operation offers a throughput of 16 cycles, even though a 32–bit intermediate result is computed. This scheme doubles the throughput but since sharing of the b registers can occur, the hardware cost is less than double.

3.3 IDEA Core

The core implementation of IDEA is obtained by cascading eight identical rounds of operations shown in Figure 3, followed by a output transformation. For convenient interfacing, four parallel–to–serial converters are inserted before the first round and four serial–to–parallel converters are appended after the output transformation. The core takes one 64–bit plaintext once every 16 cycles, yielding an effective encryption rate of $f \times 64 \div 16$ Mb/sec at a system clock rate of f MHz.

All the shift registers for key storage are linked during initialization cycles. Upon initialization, the pre–computed 52 16–bit subkeys (a total of 832 bits) are passed bit–serially into the core via the shift registers. This key scheduling mechanism is advantageous for its minimum routing and logic requirements. To further optimize area, the control circuitry of all the modules is extracted and is replaced by a global 16–bit one–hot encoding state machine. The dataflow diagram of the IDEA core is shown in Figure 8.

Each round has a latency of 109 cycles. The output transformation has a latency of 35 cycles. Each serial–to–parallel converter at the outputs has a latency of 16 cycles. Therefore, the IDEA core has an overall latency of $109 \times 8 + 35 + 16 = 923$ cycles. At a 125MHz system clock rate, the equivalent latency is 7.384 μs, which is acceptable for many applications.

3.4 Scalability

Given more resources, the bit–serial implementation of IDEA can be efficiently scaled up to achieve higher encryption rate. This is achieved by instantiating multiple IDEA core instances, but having the control signals shifted with respect to every other instance. With a single core, the implementation can be scaled up to 16 times to achieve 16 times the original encryption rate without affecting latency. A maximally scaled version of the implementation is illustrated in Figure 9. The timing diagram in Figure 10 illustrates the mechanism of input data forwarding and output data merging of a maximally scaled implementation.

4 Results

The bit–serial IDEA processor was verified with Synopsys VHDL Simulator, and was synthesized using Synopsys FPGA Express 3.3 and Xilinx Foundation Series 2.1i, with a Xilinx Virtex XCV300–4 as the target device. The fully–pipelined implementation requires 2801 Virtex slices, accounting for 91.18% of the total 3072 slices on an XCV300 device.

The basic building block of the Virtex FPGA is the the logic cell (LC). A LC includes a 4–input function generator, carry logic and a storage element. Each Virtex CLB contains four LCs, organized in two slices. The 4–input function generator are implemented as 4–input look–up tables (LUTs). Each of them can provide the functions of one 4–input LUT or a 16×1–bit synchronous RAM (called "distributed RAM"). Furthermore, two LUTs in a slice can be combined to create a 16×2–bit or 32×1–bit synchronous RAM, or a 16×1–bit dual–port synchronous RAM.

Our implementation of IDEA was successfully implemented on Annapolis Micro Systems Wildcard Reconfigurable Computing Engine [21]. The device is a Type II PCMCIA Card with a 33MHz 32–bit CardBus interface, consisting of a Xilinx Virtex XCV300 FPGA as Processing Element (PE) and two $64\text{k} \times 32$–bit SDRAMs. The design was found to be operational at room temperature up to 125MHz and this clock rate was used for all performance tests. Reliable operation over the full commercial temperature range would be expected if a XCV300–6 was used.

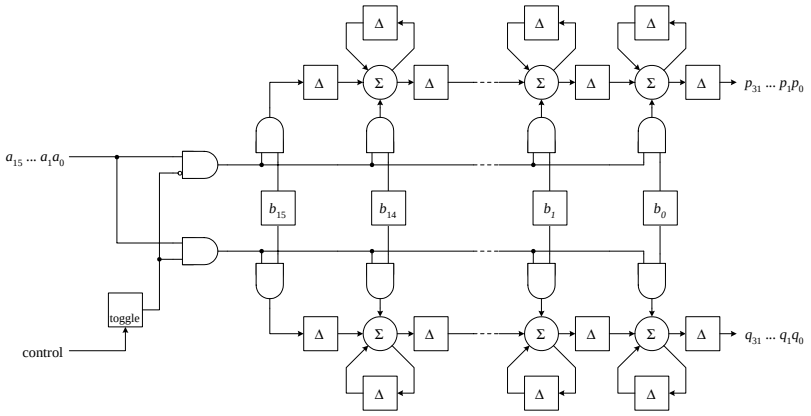

Figure 5: Parallel–serial multiplier modified for increased throughput.

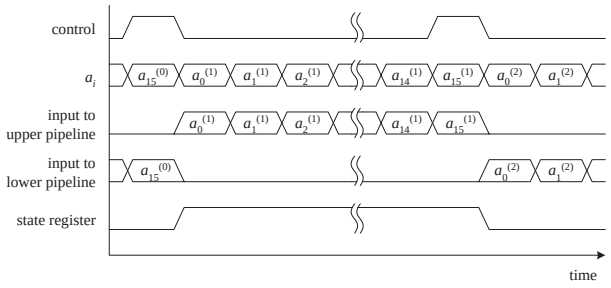

Figure 6: Timing diagram of the modified multiplier.

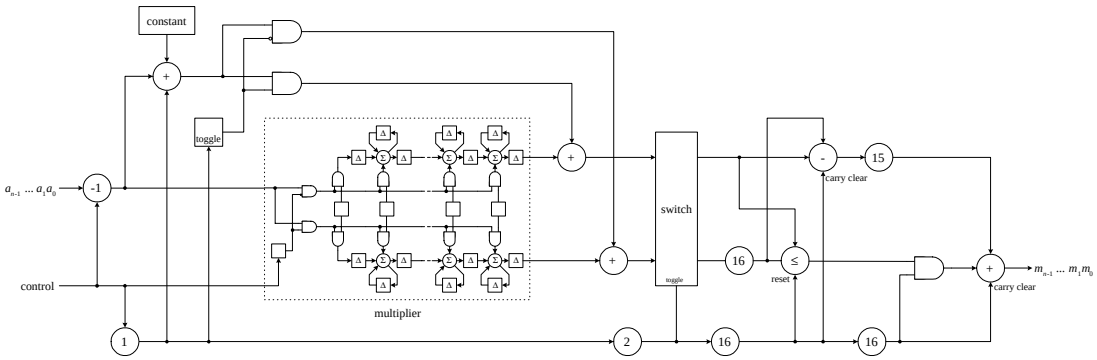

Figure 7: Bit–serial architecture for multiplication modulo $2^{16} + 1$ operations.

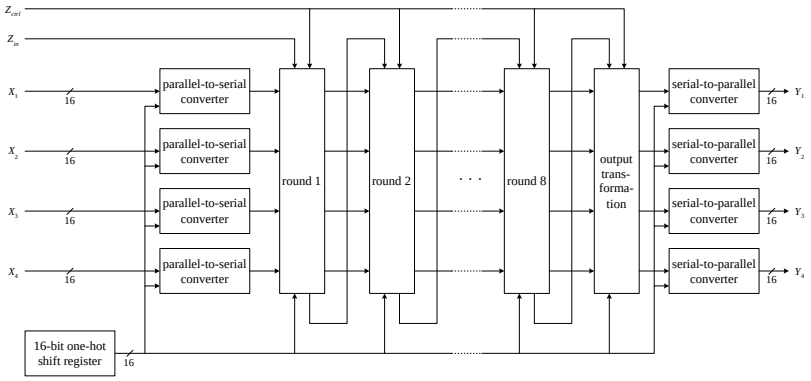

Figure 8: Dataflow diagram of the IDEA core.

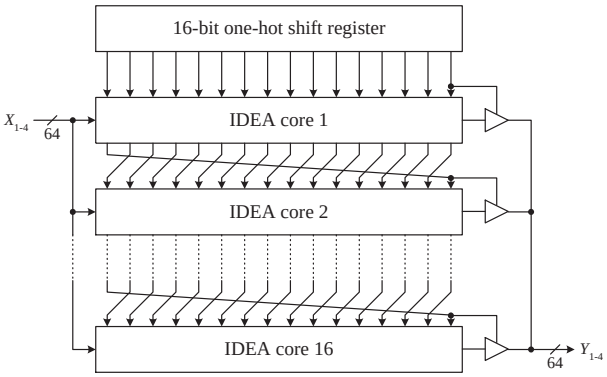

Figure 9: Maximally scaled bit–serial implementation of IDEA.

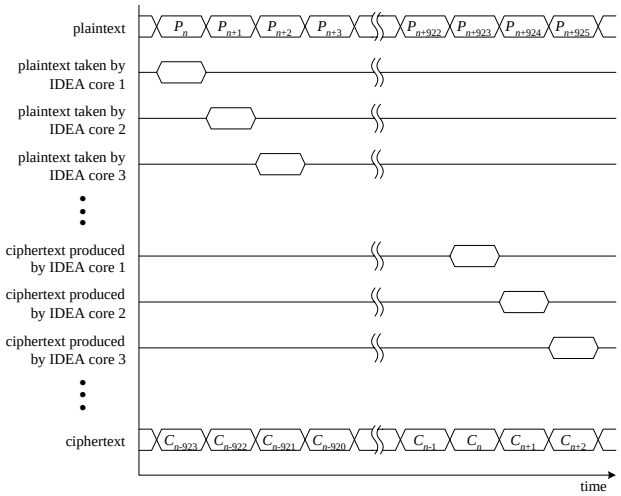

Figure 10: Timing diagram of a maximally scaled implementation, showing the instantiations of IDEA cores which take plaintext and produce ciphertext in a round–robin order.

Speed grade	–4	–5	–6
Reported clock rate (MHz)	106.378	116.229	125.202
Encryptions per second ($\times 10^6$)	6.648	7.264	7.825
Encryption rate (Mb/sec)	425.5	464.9	500.8
Latency (μs)	8.677	7.941	7.372

Table 2: Performance of IDEA core on devices of different speed grades.

Device (XCV)	300–6	600–6	1000–6
Scaling	1×	2×	4×
Number of slices	2801	5602	11204
Device slices utilization	91.18%	81.05%	91.18%
Clock rate (MHz)	125.202	125.202	125.202
Encryptions per second ($\times 10^6$)	7.825	15.650	31.300
Encryption rate (Mb/sec)	500.8	1001.6	2003.2

Table 3: Tradeoffs between performance and area of the IDEA core.

4.1 Performance of IDEA Core

Bit–serial architectures facilitate high system clock rates compared with traditional bit–parallel implementations. The performance of the core (assuming a high bandwidth interface to the data sources and sinks) is summarized in Table 2. Reported clock rate refer to the clock frequencies reported by timing analysis.

In an attempt to explore tradeoffs between performance and area, the core was generated for FPGAs of different capacities. The core was maximally scaled within the resource limitation of each device using the method described in Section 3.4. Results are summerized in Table 3.

It is estimated that a maximally scaled implementation requires $2801 \times 16 = 44816$ slices, which can produce an encryption rate of $500 \times 16 = 8$ Gb/sec at a 125MHz clock rate.

4.2 Performance on the Wildcard Platform

On the Wildcard implementation, the time taken to complete a transaction between the FPGA and host is dominated by operating system overheads. When designing the interface between the IDEA core and the host, it is crucial that the number of discrete CardBus read and write transactions is minimized and the amount of data transferred per transaction is maximized.

A block diagram of the interface is shown in Figure 11. Data is written directly to the core using a burst mode transfer of 512 64–bit plaintext blocks. After the latency period, the ciphertext is written to consecutive locations in the BlockRAM. For XCV300 devices, there are eight 256×32–bits BlockRAM [22] on the chip and they are all used in the host/IDEA interface. The results are read by the host from the IDEA processor by doing a burst mode transfer of the contents of the block RAM. The decryption process is similar except the ciphertext is written to the IDEA core and the plaintext appears in the BlockRAM.

The maximum transfer rate of CardBus is 33MHz $\times$ 32-bits = 1056 Mb/sec, but the bit–serial core, clocked at 125MHz, has an encryption rate of approximately $125 \times 64 \div 16 = 500$ Mb/sec. In order to match the bandwidth of the IDEA core, the host inserts three blank 32–bit words between every two 64–bit plaintext double–words. The maximum data rate is therefore $1056 \times 0.4 = 422.4$ Mb/sec. The interface between host and IDEA core on Wildcard requires an additional 238 slices, resulting in a total of 3039 slices, or 98.93% utilization of the XCV300.

Although the CardBus has a 1056Mb/sec maximum transfer rate, its actual data transfer rate using programmed I/O is degraded due to very large operating system overheads in setting up a CardBus transaction. The implementation achieves a measured performance of 0.61×10^6 encryptions per second (39Mb/sec). The situation could be improved by using Direct Memory Access (DMA) but the DMA interface requires an additional 400 slices and would not fit on an XCV300. The DMA interface was tested in a stand-alone configuration and measured performance for a write of 5120 words ($2048 \div 0.4$) followed by a read of 2048 words was 142Mb/sec. A larger device which can accommodate both the IDEA core and the DMA

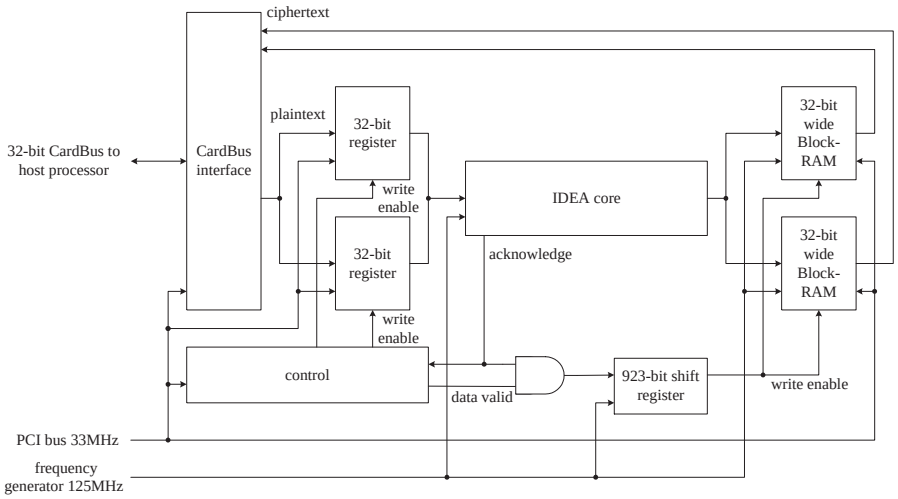

Figure 11: Interface between host processor and IDEA core via CardBus.

interface could achieve this performance. In addition, utilizing the two 64k $\times$ 32–bits SDRAMs on Wildcard could provide a larger buffer for ciphertext storage, hence reducing the number of CardBus transactions and improve the overall performance. Of course, if a 500Mb/sec data source could be connected to the FPGA (for example, directly via the I/O pins), the full bandwidth of the IDEA core presented could be achieved.

5 Conclusion

A high performance bit–serial implementation of the IDEA algorithm was presented in this paper. The architecture minimizes resource requirements, offers high levels of fine–grain parallelism and allows a high clock rate. The implementation achieves an encryption throughput of 500Mb/sec at a 125MHz system clock rate on a Xilinx Virtex XCV300–6 FPGA.

References

[1] Electronic Frontier Foundation, "DES challenge III broken in record 22 hours," January 1999. (http:// www.eff.org/ pub/ Privacy/ Crypto_misc/ DESCracker/ HTML/ 19990119_deschallenge3.html).

[2] X. Lai and J. Massay, "A proposal for a new block encryption standard," in *Advances in Cryptology, Proceedings of Eurocrypt 1990*, pp. 389–404, 1990.

[3] X. Lai, J. Massay, and S. Murphy, "Markov ciphers and differential cryptanalysis," in *Advances in Cryptology, Proceedings of Eurocrypt 1991*, pp. 17–38, 1991.

[4] M. Hellman and S. Langford, "Differential–linear cryptanalysis," in *Advances in Cryptology, Proceedings of Eurocrypt 1994*, pp. 26–36, 1994.

[5] L. R. Knudsen, "Truncated and higher order differentials," in *Proceedings of the Second International Workshop on Fast Softward Encryption*, pp. 196–211, 1995.

[6] J. Borst, "Differential–linear cryptanalysis of IDEA," ESAT–COSIC Technical Report 96–2, Department of Electrical Engineering, Katholieke Universiteit Leuven, February 1997.

[7] B. Schneier, *Applied Cryptography*. John Wiley & Sons, second ed., 1996.

[8] O. Mencer, M. Morf, and M. J. Flynn, "Hardware software tri-design of encryption for mobile communication units," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. 3045–3048, May 1998.

[9] H. Bonnenberg, A. Curiger, N. Felber, H. Kaeslin, and X. Lai, "VLSI implementation of a new block cipher," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computer and Processors*, pp. 501–513, 1991.

[10] A. Curiger, H. Bonnenberg, R. Zimmerman, N. Felber, H. Kaeslin, and W. Fichtner, "VINCI: VLSI implementation of the new secret–key block cipher IDEA," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 15.5.1–15.5.4, 1993.

[11] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner, "A 177Mb/sec VLSI implementation of the international data encryption algorithm," *IEEE Journal of Solid–State Circuits*, vol. 29, pp. 303–307, March 1994.

[12] S. Wolter, H. Matz, A. Schubert, and R. Laur, "On the VLSI implementation of the international data encryption algorithm IDEA," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 397–400, 1995.

[13] S. L. C. Salomao, V. C. Alves, and E. M. C. Filho, "HiPCrypto: A high–performance VLSI cryptographic chip," in *Proceedings of the Eleventh Annual IEEE ASIC Conference*, pp. 7–11, 1998.

[14] Ascom, *IDEACrypt Coprocessor Data Sheet*, 1999. (http:// www.ascom.ch/ infosec/ downloads/ IDEACrypt_Coprocessor.pdf).

[15] A. V. Curiger, H. Bonnenberg, and H. Kaeslin, "Regular VLSI architectures for multiplication modulo 2^n+1," *IEEE Journal of Solid–State Circuits*, vol. 26, pp. 990–994, July 1991.

[16] C. Meier and R. Zimmerman, "A multiplier module $2^n + 1$," Diploma thesis, Institut für Integrierte Systeme, ETH, Zürich, Switzerland, February 1991.

[17] R. Hartley and K. K. Parhi, *Digit–Serial Computation*. Kluwer Academic Publishers, 1995.

[18] Xilinx, Inc., *Xilinx Libraries Guide*, 1999.

[19] M. George and P. Alfke, *Linear Feedback Shift Registers in Virtex Devices*. Xilinx, Inc., August 1999. Application Note XAPP210, Version 1.0.

[20] R. F. Lyon, "Two's complement pipeline multipliers," *IEEE Transactions on Communications*, vol. 12, pp. 418–425, April 1976.

[21] Annapolis Micro Systems, Inc., *Wildcard Reference Manual*, 1999. Revision 1.1.

[22] Xilinx, *The Programmable Logic Data Book*, 1999.