Tradeoffs in Parallel and Serial Implementations of the International Data Encryption Algorithm IDEA

O.Y.H. Cheung[1], K.H. Tsoi[1], P.H.W. Leong[1], and M.P. Leong[1]

Department of Computer Science and Engineering,
The Chinese University of Hong Kong,
Shatin, N.T.,
Hong Kong
{yhcheung,khtsoi,phwl,mpleong}@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk

Abstract. A high-performance implementation of the International Data Encryption Algorithm (IDEA) is presented in this paper. The design was implemented in both bit-parallel and bit-serial architectures and a comparison of design tradeoffs using various measures is presented. On an Xilinx Virtex XCV300-6 FPGA, the bit-parallel implementation delivers an encryption rate of 1166 Mb/sec at a 82 MHz system clock rate, whereas the bit-serial implementation offers a 600 Mb/sec throughput at 150 MHz. Both designs are suitable for real-time applications, such as online high-speed networks. The implementation is runtime reconfigurable such that key-scheduling is done by directly modifying the bitstream downloaded to the FPGA, hence enabling an implementation without the logic required for key-scheduling. Both implementations are scalable such that higher throughput is obtained with increased resource requirements. The estimated performances of the bit-parallel and bit-serial implementations on an XCV1000-6 device are 5.25 Gb/sec and 2.40 Gb/sec respectively.

Keywords: Cryptographic hardware, digital-design, reconfigurable-computing, performance-tradeoffs.

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, the aim of cryptography is to make it intractable to retrieve the data without knowledge of a secret piece of information called a key. Cryptography is an ideal application for custom computing machines since they offer the following advantages over VLSI technologies

- it is possible to use the same Field-Programmable Custom Computing Machine (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 [4–6]. 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 an equivalent encryption rate of 23.53 Mb/sec, on an Intel Pentium II 450 MHz machine. Implementation of IDEA using the Intel MMX multimedia instructions was proposed by Helger [8] and achieves 0.51×10^6 encryption per seconds or a equivalent encryption rate 32.9 Mb/sec, on an Intel Pentium II 233 MHz machine. Our optimized software implementation running on a Sun Enterprise E4500 machine with twelve 400 MHz Ultra-IIi processor, performs 2.30×10^6 encryptions per second or a equivalent encryption rate of 147.13 Mb/sec, still cannot be applied to applications such as encryption for 155 Mb/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, having lower power consumption and smaller footprint than a high speed software implementation. A paper design of an IDEA processor which achieves 528 Mb/sec on four XC4020XL devices was proposed by Mencer et. al. [9]. The first VLSI implementation of IDEA was developed and verified by Bonnenberg et. al. in 1992 using a 1.5 μm CMOS technology [10]. This implementation had an encryption rate of 44 Mb/sec. In 1994, VINCI, a 177 Mb/sec VLSI implementation of the IDEA algorithm in 1.2 μm CMOS technology, was reported by Curiger et. al. [11, 12]. A 355 Mb/sec implementation in 0.8 μm technology of IDEA was reported in 1995 by Wolter et. al. [13],

followed by a 424 Mb/sec single chip implementation of 0.7 μm technology by Salomao et. al. [14] was reported. In 2000, Leong et. al. proposed a 500 Mb/sec bit-serial implementation of IDEA on an Xilinx Virtex XCV300-6 FPGA which is scalable on larger devices [15]. Later, Goldstein et. al reported an implementation on the PipeRench FPGA which achieves 1013 Mb/sec [16]. A commercial implementation of IDEA called the IDEACrypt Kernel developed by Ascom achieves 720 Mb/sec [17] at 0.25 μm technology. The implementation dervied from the IDEACrypt Kernel, called the IDEACrypt Coprocessor, has a throughput of 300 Mb/sec [18].

In this paper, two Xilinx Virtex Field Programmable Gate Array (FPGA) based implementations of the IDEA algorithm are described. On an XCV300-6 device, the bit-parallel implementation offers a 1166 Mb/sec encryption rate, while the bit-serial implementation has a throughput of 600 Mb/sec. The implementation is scalable so that throughput and area tradeoffs can be addressed. Applications of these designs include virtual private networks (VPNs) and embedded encryption/decryption devices. To illustrate various design tradeoffs, an analysis on both of the designs in terms of area, latency, throughput and other design measures was carried out.

Key-scheduling in both implementations is achieved by modifying the bitstream downloading to the FPGA, in a manner similar to that described by Patterson in an implementation of DES [19]. Instead of doing this using the JBits Applications Programming Interface (API), a technique for the direct modification of the binary bitstream was used. The approach is advantageous because dedicated logic for key-scheduling is not required in the designs hence leaving more logic resources for performing computation.

This paper is organized as follows. In Section 2 the IDEA algorithm as well as algorithms for multiplication modulo $2^n + 1$ are described. The bit-parallel and bit-serial implementations of IDEA are presented in Section 3 and 4 respectively. In Section 5 the methodology to achieve runtime reconfigurability is described. In Section 6 results are given. Conclusions are drawn in Section 7.

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 to mix 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 possess strong cryptographic strength because

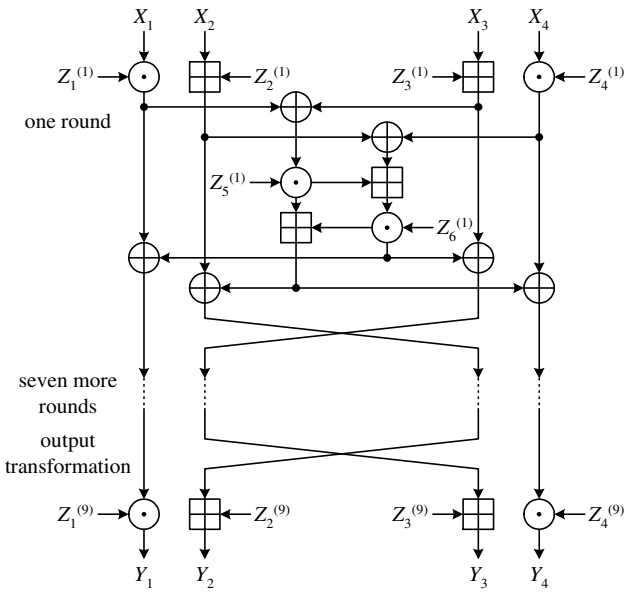

Fig. 1. Block diagram of the IDEA algorithm.

- 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
- 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 by 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 $Z'^{(r)}_i$ can be computed from the encryption subkeys with reference to Table 1.

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

In Electronic Codebook (ECB) mode [7], the data dependencies of the IDEA algorithm have no feedback paths. Additionally, in practice, latencies of order of microseconds are acceptable. These features make deeply pipelined implementations possible.

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. [20] described and compared several VLSI architectures for multiplication modulo $2^n + 1$ and found that an architecture proposed by Meier and Zimmerman [21], using modulo 2^n adders with bit-pair recoding offers the best performance.

The C code for the multiplication modulo $2^{16} + 1$ operation by modulo 2^{16} adders using bit-pair recoding is as follows.

```
 1   uint16 mulmod(uint16 x, uint16 y)
 2   {
 3      uint16 xd, yd, th, tl;
 4      uint32 t;
 5      xd = (x - 1) & 0xFFFF;
 6      yd = (y - 1) & 0xFFFF;
 7      t = (uint32) xd * yd + xd + yd + 1;
 8      tl = t & 0xFFFF;
 9      th = t >> 16;
10      return (tl - th) + (tl <= th);
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 Bit-Parallel Implementation

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

Modulo multiplication is the bottleneck in the IDEA algorithm. In a single round of the algorithm there are four modular multiplications so a well-designed multiplication modulo $2^{16} + 1$ operator is crucial since it directly affects the system performance both in terms of area and throughput.

The modular multiplication algorithm described in Section 2.1 was used in our design, but instead of taking x and y as inputs, the operator takes x and y_d as inputs. As one of the operands is a subkey which is regarded as a constant, the modification eliminates one subtraction operator by taking the advantage of pre-decremented subkeys (Section 2.1, pseudocode line 6).

In order to implement a well-designed multiplication modulo $2^{16} + 1$ operator, the throughput of the operator is maximized by introducing more pipeline stages. In our design, 16-bit mulitplier used in Section 2.1 (pseudocode line 7) is constructed by Xilinx CORE Generator [22] which has a latency of 4 cycles. And the multiplication modulo $2^{16} + 1$ operator pipeline has a latency of 7 cycles.

3.2 Bit-Parallel IDEA Core

The IDEA algorithm is a cascade of eight identical rounds of operations, followed by a output transformation. By instantiating building blocks, that is, additions, XORs and modular multiplications, and inserting appropriate stage latches for time-alignment, a module for one round of computation is formed. For the best area-efficiency, stage latches are constructed by Virtex SRL16E primitives [23, 24].

Due to limited hardware resources, each round of the algorithm shares the same physical resource, but with different key-schedules. Output transformation also reuses the resource. In our implementation the key-schedules are stored inside ROM primitives. The architecture of the bit-parallel IDEA core is shown in Figure 2.

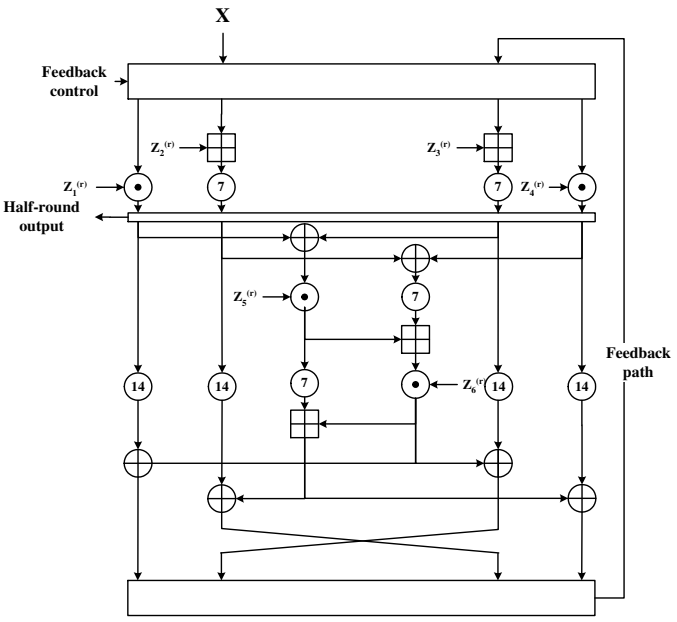

Fig. 2. Architecture of the bit-parallel IDEA core.

As mentioned earlier, for ECB mode operations, data dependencies of the IDEA algorithm have no feedback paths. This property enabled the round architecture to take input values until the pipelined is filled, and output values are redirected to the input of the pipeline subsequently. In an IDEA round, the data passes through three multiplication modulo $2^{16}+1$ operators, each of which has a latency of 7 cycles. Thus the full round pipeline has a latency of

21 cycles For an output transformation, the data must pass through a single multiplication modulo $2^{16}+1$ operator with pipeline latency of 7 cycles. Therefore the core has a total latency of $21 \times 8 + 7 = 175$ cycles. The core takes 21 64-bit plaintexts per $21 \times 9 = 189$ cycles, equivalently performing encryption at $(21 \div 189) \times 64 \times f$ Mb/sec with a system clock rate of f MHz. For instance, at a 82 MHz clock rate, the core delivers an encryption rate of 583 Mb/sec with a latency of 2.134 μs.

4 Bit-Serial Implementation

The bit-serial implementation mentioned below is an improved implementation of [15]. By register reordering and register duplication, the improved implementation offers an encryption throughput of 600 Mb/sec, 20% faster than the original implementation.

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 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 [25].

Two of the primitive operators used in IDEA, namely XOR and addition modulo 2^{16}, can be easily implemented bit-serially. 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. As in the parallel implementation, stage latches and constants are implemented using SRL16E primitives. Additionally, constants are also implemented as SRL16E primitives, with its output connected to its input to form a cyclic shift register.

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

The modular multiplication algorithm described in Section 2.1 was directly applied in the bit-serial implementation of the algorithm. The operator optimization used in the bit-parallel implementation, described in Section 3.1, was not applied in the bit-serial implementation because comparisons in bit-serial architectures are not efficient in terms of latency.

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 (Section 2.1, pseudocode line 7-9). The two portions, t_h and t_l, are 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 parallel-serial multiplier [26] was developed which addresses this problem. 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 3. 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.

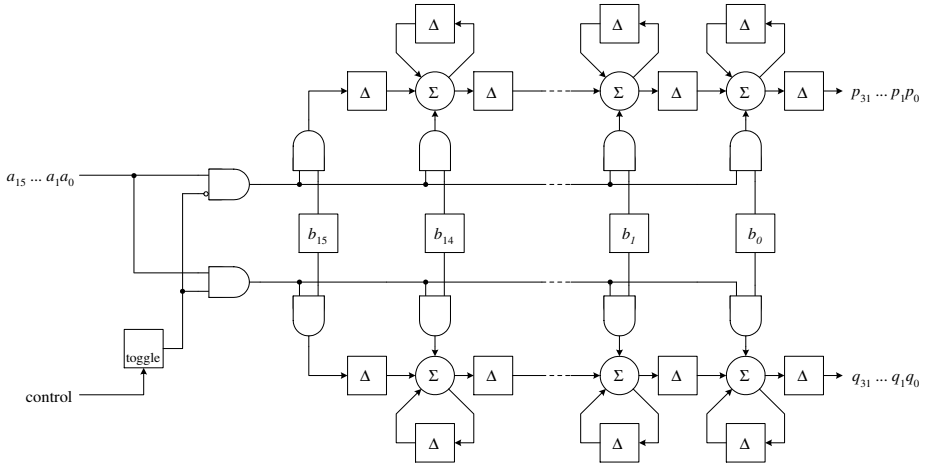

Fig. 3. Parallel-serial multiplier modified for increased throughput.

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 [23] primitive. The complete ar-

chitecture for the modular multiplication operation is shown in Figure 4. 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.

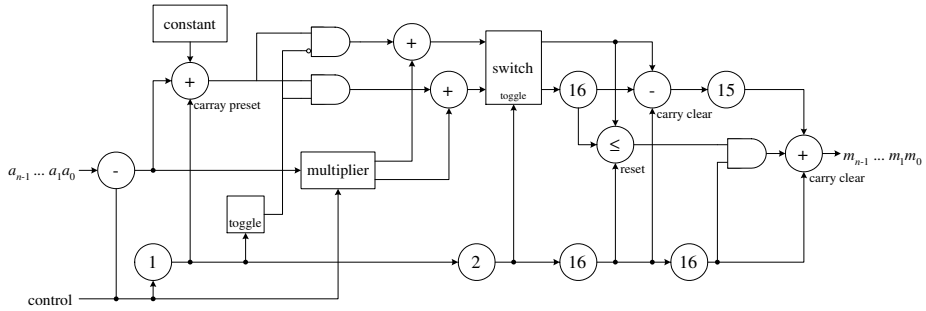

Fig. 4. Bit-serial architecture for multiplication modulo $2^{16} + 1$ operations.

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.

4.2 Bit-Serial IDEA Core

The core implementation of IDEA is obtained by cascading eight identical rounds of operations shown in Figure 5, followed by a 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. At 150 MHz, for example, the performance of the core is 600 Mb/sec.

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 150 MHz system clock rate, the equivalent latency is 6.153 μs.

5 Bitstream Reconfiguration

The basic building block of the Virtex FPGA is the logic cell (LC). A LC includes a 4-input function generator, carry logic and a storage element. Each Virtex Configurable Logic Blocks (CLB) contains four LCs, organized in two slices. The 4-input function generator are implemented as 4-input LUTs. Each of them

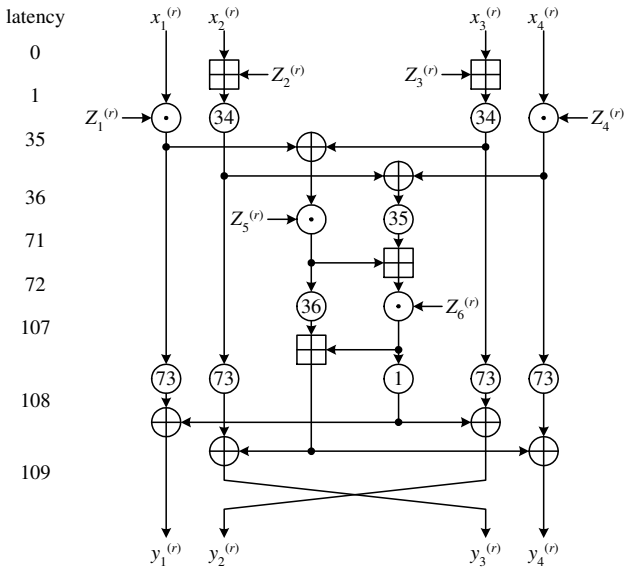

Fig. 5. Bit-serial architecture for one round of IDEA algorithm.

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.

The contents of the LUTs upon initialization are encoded in the bitstream. Xilinx disclosed the format of bitstream for Virtex series FPGA [27, 28], hence it is possible to edit the bitstream and alter the contents of LUTs. Our approach to achieve runtime reconfigurability is to build all configurable blocks from LUTs and later modify the bitstream.

More specifically, the key-schedule is stored only inside ROM or SRL16E primitives which are implemented as LUTs. After technology-mapping, placement and routing, a circuit description (with a *.ncd* extension) is generated. Using the *ncdread* tool provided by Xilinx, the contents of the circuit description can be converted into a human-readable format. It is possible to extract the physical location of individual LUTs from the output of *ncdread*.

We have developed software to customize bitstreams for different key-schedules. In the first step (which need only be performed once for a given design), information concerning the physical location of individual LUTs which are used in the key schedule is extracted from the *.ncd* file and written to a location file (`locfile`). To modify a bitstream, the LUTs are directly modified by a program to use a given key-schedule. The pseudocode below describes the technique that was used.

```
changekeys(locfile, bitstream)
{
   locdb = read(locfile);
   foreach bit in the key
   {
      Find location of the LUT using locdb;
      Modify the value of the bit in the LUT;
   }
   Recompute CRC for the bitstream;
   Write bitstream;
}
```

On an Intel Pentium III 866 MHz machine, the reconfiguration process requires the modification of 6×16 LUTs and changing a key takes approximately 0.12 seconds.

In some applications, runtime reconfiguration may not be desirable e.g. if the bitstream is placed in a ROM or in an Application-Specific Integrated Circuit (ASIC) implementation. For these cases, shift registers can be employed for the key-schedule. The shift registers are linked to form a large shift register when key-schedules are being fetched. This long shift register breaks down into the original shift registers after initialization. This method requires minimal logic and routing resources.

6 Results

Both the bit-parallel and bit-serial IDEA processor was verified with Synopsys VHDL Simulator, and was synthesized using Synopsys FPGA Express 3.5 and Xilinx Foundation Series 3.1i, with Xilinx Virtex XCV300-6 as target device.

Our serial and parallel implementations of IDEA were successfully implemented on Annapolis Micro Systems Wildcard Reconfigurable Computing Engine [29]. The device is a Type II PCMCIA Card with a 33 MHz 32-bit CardBus interface, consisting of an Xilinx Virtex XCV300–6 FPGA as Processing Element (PE) and two $64\text{k} \times 32$-bit SDRAMs. A single core parallel implementation was also tested using a Pilchard card [30] which uses a memory slot interface instead of a CardBus interface.

6.1 Performance of IDEA Core

For the bit-parallel implementation, a single core/round of the algorithm requires 1178 Virtex slices. An XCV300 device can accommodate two rounds of the algorithm, accounting for 2444 slices (including extra logic required for scaling), or 79.56% of the total 3072 slices.

For the bit-serial implementation, the fully-pipelined implementation (8 rounds plus output transformation), with parallel-to-serial converters at inputs and serial-to-parallel converters at outputs, requires 2878 Virtex slices which occupies 93.68% of CLB resources.

It was observed that the building blocks offer faster computations in the stand-alone configuration, but performance degrades when they are being used as components in the hierarchical design. Hence, core performance improvement may be obtained by floorplanning, such that inter-component routing is minimized. The performance of the cores (assuming a high-bandwidth interface to the data sources and sinks) is summarized in Table 2.

	Bit-parallel	Bit-serial
Number of Cores	2	1
Clock rate (MHz)	82.0	150.0
Encryptions per second ($\times 10^6$)	18.220	9.375
Encryption rate (Mb/sec)	1166.08	600.0
Latency (μs)	2.134	6.153

Table 2. Summary of performance for the two implementations on an Virtex XCV300-6 device.

In an attempt to explore tradeoffs between performance and area, the core was generated for FPGAs of different capacities. Since there are no data dependencies, the implementations can be easily scalable by instantiation of multiple cores. The designs were maximally scaled within the resource limitation of each device to produce the results summarized in Table 3.

	Bit-parallel			Bit-serial		
Device (speed grade -6)	XCV300	XCV600	XCV1000	XCV300	XCV600	XCV1000
Scaling	2×	5×	9×	1×	2×	4×
Number of slices	2444	6368	11602	2878	5756	11512
Device utilization	79.56%	92.13%	94.42%	93.68%	83.28%	93.68%
Encryptions per second ($\times 10^6$)	18.220	45.551	81.991	9.375	18.750	37.500
Encryption rate (Mb/sec)	1166.1	2915.3	5247.4	600.0	1200.0	2400.0

Table 3. Tradeoffs between performance and area of the IDEA cores on different devices.

6.2 Performance

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

Data from host is written directly to the core using a burst mode transfer of 1024 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 [31] 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 BlockRAM. The decryption process is similar except the ciphertext is written to the IDEA core and the plaintext appears in the BlockRAM.

The interface between host and IDEA core on Wildcard requires approximately an additional 160 slices, resulting in a total of 2606 slices (84.83%) and 3039 slices (98.93%) utilization of the XCV300 for the bit-parallel and bit-serial implementations respectively.

The burst transfer rate of CardBus is $33 \times 32 = 1056$ Mb/sec. However, due to large overheads in the CardBus transactions, both the implementations achieve a measured performance of 0.61×10^6 encryptions per second (39 Mb/sec) on a 300 MHz Intel Pentium II laptop computer. The situation could be improved by using Direct Memory Access (DMA) channels. In addition, utilizing the two $64\text{k} \times 32$-bits SDRAMs on Wildcard could provide a larger buffer for ciphertext storage hence reducing the number of transactions.

6.3 Pilchard

In an attempt to improve the PC to FPGA data transfer rate, the bit-parallel implementation was ported to a Pilchard card [30] which utilizes a memory slot interface for improved performance over a CardBus interface. The Pilchard card used the same XCV300–6 device as in the Wildcard. The implementation uses only a single IDEA core/round and requires a total of 1319 slices (42.93% utilization). Pilchard offers a higher bandwidth between the PC and FPGA and the implementation achieved a measured encryption performance of 146 Mb/sec on an Intel Pentium IIII 866 MHz desktop PC.

7 Conclusion

Two high-performance runtime reconfigurable implementations of the IDEA algorithm were presented in this paper. In both designs, the bitstream is customize for a particular key and this procedure saved hardware resources in our design. In implementations on the same XCV300-6 part, the bit-parallel version achieved an encryption rate of 1166 Mb/sec using an 82 MHz clock, whereas the bit-serial

implementation achieved a 600 Mb/sec throughput at a clock rate of 150 MHz. The bit-parallel implementation achieved a higher throughput with lower latency than the bit-serial implementation, while the bit-serial implementation permits a minimal area fully-pipelined design.

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 Software 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. Schneider, *Applied Cryptography*. John Wiley & Sons, second ed., 1996.
8. H. Lipmaa, "Idea: A cipher for multimedia architectures?," in *Selected Areas in Cryptography '98*, pp. 253–268, August 1998.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. M. P. Leong, O. Y. H. Cheung, K. H. Tsoi, and P. H. W. Leong, "A bit-serial implementation of the international data encryption algorithm (IDEA)," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 122–131, April 2000.

16. S. C. Goldstein, H. Schmit, M. Budiu, M. Moe, and R. R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, pp. 70–77, April 2000.
17. Ascom, *IDEACrypt Kernel Data Sheet*, 1999. (http:// www.ascom.ch/ infosec/ downloads/ IDEACrypt_Kernel.pdf).
18. Ascom, *IDEACrypt Coprocessor Data Sheet*, 1999. (http:// www.ascom.ch/ infosec/ downloads/ IDEACrypt_Coprocessor.pdf).
19. C. Patterson, "High performance DES encryption in Virtex FPGAs using JBits," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 113–121, April 2000.
20. 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.
21. C. Meier and R. Zimmerman, "A multiplier modulo $(2^n + 1)$," Diploma thesis, Institut für Integrierte Systeme, ETH, Zürich, Switzerland, February 1991.
22. Xilinx, Inc., *Xilinx Coregen Reference Guide*, 2000. Version 3.1i.
23. Xilinx, Inc., *Xilinx Libraries Guide*, 1999.
24. M. George and P. Alfke, *Linear Feedback Shift Registers in Virtex Devices.* Xilinx, Inc., August 1999. Application Note XAPP210, Version 1.0.
25. R. Hartley and K. K. Parhi, *Digit-Serial Computation.* Kluwer Academic Publishers, 1995.
26. R. F. Lyon, "Two's complement pipeline multipliers," *IEEE Transactions on Communications*, vol. 12, pp. 418–425, April 1976.
27. S. Kelem, *Virtex Configuration Architecture Advanced Users' Guide.* Xilinx, Inc., September 1999. Application Note XAPP151, Version 1.2.
28. C. Carmichael, *Virtex FPGA Series Configuration and Readback.* Xilinx, Inc., September 1999. Application Note XAPP152, Version 1.2.
29. Annapolis Micro Systems, Inc., *Wildcard Reference Manual*, 1999. Revision 1.1.
30. P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and K. H. Lee, "Pilchard - a reconfigurable computing platform with memory slot interface," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (to appear)*, April 2001.
31. Xilinx, *The Programmable Logic Data Book*, 2000.