A System Level Implementation of Rijndael on a Memory-slot based FPGA Card

Dennis Ka Yau Tong, Pui Sze Lo, Kin Hong Lee, Philip H.W. Leong

{kytong, pslo, khlee, phwl}@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong.

Abstract - This paper describes system level issues encountered in a high performance implementation of a Rijndael encryption core on a memory-slot based reconfigurable computing platform called Pilchard. The Rijndael algorithm was adopted in 2000 by the US National Institute of Standards and Technology (NIST) as the Advanced Encryption Standard (AES).

In the implementation of Rijndael, changing the number of unrolled rounds in the encryption core can affect the performance of the system. It is shown that for the design presented, the highest performance of 755 Mbit/sec was achieved by implementing a core with a single round. Although it is relatively easy to implement a high performance core on an FPGA, due to I/O bottlenecks, achieving high system level performance is more difficult. In order to optimize the performance of the host/FPGA interface, special instructions from the Intel Pentium III streaming SIMD extensions (SSE) along with write-combining memory operations were used. These features enabled the measured throughput of the AES core to reach 445 Mbit/sec which, although still slower than the AES core, was double that of an unoptimized interface.

1. INTRODUCTION

In September 1997 the National Institute of Standards and Technology (NIST) issued a request for possible candidates for a new Advanced Encryption Standard (AES) to replace the existing Data Encryption Standard (DES). In October 2000, Rijndael was selected to be the AES and will be used officially by U.S. Government organizations.

Although previous hardware implementations of the Rijndael algorithm have been reported with performance up to 7 Gb/s (reviewed in Section II), we are not aware of any reports of system level implementations which address issues of how a host can supply data at such a high rate. The main objective of this work was to explore system level issues associated with implementing an FPGA-based Rijndael encryption engine on a low-cost memory slot based reconfigurable platform [7] connected to a low-end PC.

The main contributions of this paper are as follows:

- The actual system level throughput of a high performance Rijndael encryption system was measured, reported and optimized. This implementation was optimized for small area rather than maximum speed since, as will be shown later, the performance of our system is limited by the speed of the host computer to FPGA interface rather than the speed of the Rijndael core.
- Comparisons of different degrees of unrolling for a Rijndael core were made to better understand possible tradeoffs between area and throughput in the core.
- It was shown that the transfer rate of a memory-slot based FPGA board such as Pilchard can be significantly improved by using a write-combining memory mode along with Intel Pentium III streaming SIMD extension (SSE) instructions.

The rest of this paper is organized as follows: in Section II, a review of previous work on software and hardware implementations of Rijndael is given. The Rijndael algorithm is described in Section III. In Section IV, the FPGA Rijndael core implemented in this work is described. In Section V, an unrolled design is presented. In Section VI, methods to improve transfer efficiency between the PC and FPGA are introduced by explaining different memory modes on the Pentium along with a brief description of the MMX and SSE instruction sets are given. Results are presented in Section VII and conclusions are drawn in Section VIII.

2. PREVIOUS WORK

Since the Rijndael algorithm made its appearance at the first AES Candidate Conference (AES1) in August 1998, many implementations of the algorithm have been reported. The fastest known software implementation of Rijndael was developed by Brian Gladman [2]. On a 933 MHz Pentium III processor, his 128-bit key design achieved a throughput of 325 Mbits/sec, the 192-bit key design reached 275 Mbits/sec and the 256-bit key design ran at 236 Mbits/sec.

An implementation of an electronic codebook (ECB) mode 128-bit key encryption core on a Xilinx Virtex-E XCV812E-8-BG560 device by McLoone and McCanny had a throughput of 7 Gbits/sec [3]. The same authors also made an implementation which could perform both encryption and decryption which was reported to have a throughput of 3239 Mbits/sec on a Xilinx Virtex-E XCV3200E-CG1156-8 device [3]. A partially unrolled design by Elbirt, Yip, Chetwynd and Paar on the Virtex XCV1000-BG560 FPGA achieved 1937.9 Mbits/sec [4]. A T-Box implementation of the Rijndael encryption was reported to give 750 Mbits/sec on an Altera APEX 1K400-1 by Fischer and Drutarovsky [5]. More previous work can be found at the NIST website on AES (http://www.nist.gov/aes).

3. RIJNDAEL ALGORITHM

Rijndael is an iterated block cipher which supports variable block length and key length. Both lengths can be independently specified as 128, 192 or 256 bits. Rijndael has a variable number of iterations: 10, 12 and 14 for key lengths of 128, 192 and 256 respectively. In this work, a 128 bit block and key length are assumed, although the design could be adapted without difficulty to other block and key lengths. Note that the AES standard specifies a 128 bit block.

In this section a brief description of the Rijndael algorithm is given. A more detailed description of the Rijndael algorithm can be found on the NIST website (http://www.nist.gov/aes/rijndael).

State, Cipher Key and Number of Iterations

Transformations in Rijndael operate on an intermediate result, called the *State*. The State can be pictured as a rectangular array of bytes. This array has 4 rows. The number of columns is denoted by N_b and is equal to the block length divided by 32. Transformations in Rijndael treat the AES standard 128-bit data block as a 4 column rectangular array of 4-byte vectors. A 128-bit plaintext has 16 bytes (B_0, B_1, B_2, …, B_{14}, B_{15}) and it is interpreted as a State.

The cipher key is also considered to be a rectangular array with four rows, the number of column N_k being the key length divided by 32. The number of rounds are denoted by N_r and depends on the values N_b and N_k. For 128 bit blocks, N_b=4, N_k=4 and N_r = 10.

128-bit Key Rijndael Encryption

The 128-bit key Rijndael encryption algorithm consists of an initial data/key addition, then 9 round transformations followed by a final round.

The Key Schedule expands the key entering the cipher so that a different *round key* is created for each iteration, as shown in Fig. 3.1 below.

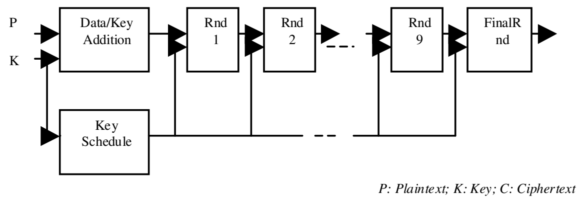

Fig. 3.1 Illustration of 128-bit Key Rijndael Encryption Algorithm

Round Transformation

The Rijndael round transformation consists of four different operations. They are a *ByteSub Transformation*, a *ShiftRow Transformation*, a *MixColumn Transformation* and a *Round Key Addition*. In pseudocode, a round transformation is:

```
Round(State, RoundKey)  {
        ByteSub(State);
        ShiftRow(State);
        MixColumn(State);
        AddRoundKey(State, RoundKey);
}
```

The final round is similar to a round except that it does not have the MixColumn(State) transformation:

```
FinalRound(State, RoundKey)     {
        ByteSub(State);
        ShiftRow(State);
        AddRoundKey(State, RoundKey);
}
```

The *ByteSub* Transformation is a byte substitution operated on each of the State bytes independently. The lookup table for substitution, i.e. the S-Box, is constructed by finding the multiplicative inverse of each byte in GF(2^8). An affine transformation is then applied, which inverses multiplying the result by a matrix and adding to the hexadecimal number '63'. The inverse of ByteSub is a byte substitution using the inverse table.

The *ShiftRow* Transformation shifts the rows of the State cyclically by a row dependent amount. Row 0 is not shifted. Row 1 is shifted over C1 bytes, row 2 over C2 bytes and row 3 over C3 bytes where for N_b=4, C1, C2 and C3 are 1, 2 and 3 respectively. The inverse of ShiftRow is a cyclic shift of the 3 bottom rows over N_b – C1, N_b – C2, and N_b – C3 bytes respectively so that the byte at position j in row i moves to position ($j + N_b - Ci$) mod N_b.

The *MixColumn* Transformation operates on the columns of the State. Each column is considered as a polynomial over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial c(x), where, c(x) = '03'x^3 + '01' x^2 + '01'x + '02'. The inverse of MixColumn is similar to MixColumn. Every column is transformed by multiplying it with, instead of c(x), a specific polynomial d(x), where d(x) = '0B'x^3 + '0D' x^2 + '09'x + '0E'

In *Round Key addition*, a Round Key is applied to the State by a simple bitwise EXOR operation. The Round Key is derived from the Cipher Key by means of the key schedule operation, which would be described in detail later. The length of Round Key is equal to the block length N_b. *AddRoundKey* is its own inverse. This means when a State is EXORed with a round key to give a new State, the original State can be recovered by EXORing the new State with the same round key.

Key Schedule

The Round Keys, Ki, are derived from the Cipher Key by means of the Key Schedule and are different for each round number. The Key Schedule consists of two parts: *Key Expansion and Round Key Selection*.

Key Expansion is a process of expanding the Cipher Key into a linear array of 4-byte words. The length of this array is determined by the length of data block N_b, multiplied by the number of rounds N_r plus 1, i.e., $N_b * (N_r + 1)$. Thus for N_b=4 and N_r=10, the length is 44.

Key Expansion starts with the original key being the first N_k words, say, W_0 to W_3 for N_k = 4. Then W_0 to W_3 are expanded to generate the next 4 words, W_4 to W_7. W_8 to W_{11} are expanded from W_4 to W_7 as shown in Fig. 3.2. The iterations continue until the final 4 words W_{41} to W_{43} are generated. Each word W_i is the EXOR of the previous word W_{i-1} with the word 4 positions earlier, i.e. W_{i-4}. Additional operations are performed prior to expansion in word W_i when i is the precessor of multiples of 4. Such words need to undergo the ByteSub, ByteRot and RCons transformations. The ByteSub transformation is the same as described in the Round Transformation where individual bytes in the word are replaced according to S-Box. The ByteRot transformation is simply a rotation of bytes in a word from (x0, x1, x2, x3) to (x1, x2, x3, x0). The RCons transformation produces output which is the EXOR result of the input with a predetermined constant. The predetermined constant is dependent on the round number of the current key.

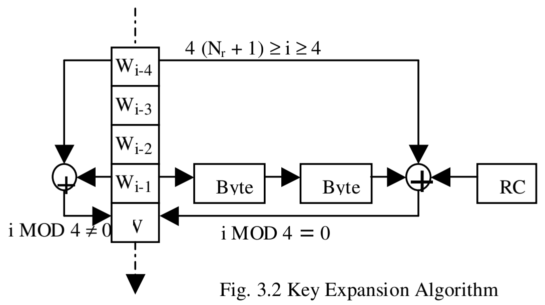

Fig. 3.2 Key Expansion Algorithm

The Round Key K_i is selected from the expanded key for $W[N_b * i]$ to $W[N_b * (i + 1)]$. A 10-round design requires 11 round keys (44 words). Round Key 0 is W_0 to W_3 and is used in the initial "Data/Key Addition" as shown in Fig. 3.1. Round key 1 is W_4 to W_7 and used in round 1, round key 2 is W_8 to W_{11} and used in round 2 and so on. Finally, round key 10, W_{40} to W_{43}, is used in the final round.

4. FPGA-BASED IMPLEMENTATION

The Rijndael algorithm can be implemented using a looping structure where data is iteratively passed through a round transformation. Several architectural options were considered. The term *Iterative Looping* is used for the case where the core consists of only one round implemented as a single combinatorial logic block and the cipher must iterate up to the total number of rounds to perform an encryption. This approach has a low register-register delay but requires a larger number of clock cycles to perform an encryption. In this section, an Iterative Looping design is described.

Development Environment

The FPGA device used for the Rijndael implementation was a Xilinx Virtex-E XCV1000-6. It is composed of a 64 x 96 array of lookup table based Configurable Logic Blocks (CLBs), each of which acts as a 4-bit element comprised of two 2-bit slices for a total of 12288 CLB slices. In addition, the Virtex-E FPGA Series provides dedicated blocks of on-chip, dual-read/write port synchronous RAM, with 4096 memory cells, known as Block SelectRAMs (BRAMs). Each port of the BRAM can be independently configured as a read/write port, a read port, a write port, and can also be configured to a specific data width.

The Pilchard board [7] used for interfacing the FPGA with a host personal computer (PC) uses the PC's DIMM memory slot. The DIMM interface offers higher bandwidth, simpler interface and lower latency than a traditional PCI interface. From earlier measurements with Pilchard [7], the read/write transfer rate of Pilchard using an "Uncacheable" memory type and "movq" instructions was 35 MB/sec (280 Mbit/sec). The "Uncacheable" memory type guarantees that all reads and writes would appear on the system bus in the same order as the program. Compared with the read/write transfer rate of 40 Mbit/sec offered by traditional PCI bus (without DMA), Pilchard can provide a higher transfer rate and thus a higher System Throughput. As a result, the Pilchard based Rijndael encryption on Pilchard can attain higher data rates than a similar implementation on traditional PCI bus interface.

Rijndael Encryption Core Design

The major components of the Rijndael Core are the Control Unit, the Round Transformation Unit and the Key Schedule Unit, as shown in Fig. 4.1.

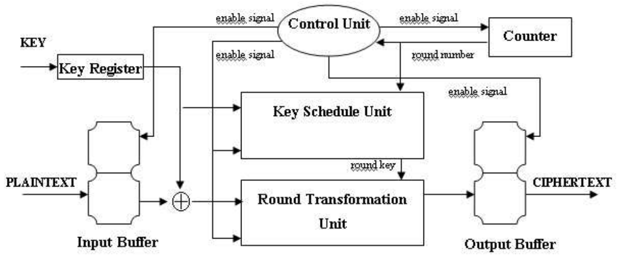

Figure 4.1 Overview of Rijndael Encryption Core

The Control Unit is a Finite State Machine (FSM) that controls other components of the core. Recalling the 0th round is an EXOR operation between the plaintext and the key, the Round Transformation accepts this result as its input. It receives the round key from the Key Schedule Unit, in which the round key are generated in the same clock cycle as the round transformation. After 11 cycles the Control Unit will signal that an encryption has been completed and ciphertext will be output.

In the Round Transformation Unit and Key Schedule Unit, operations defined in Section III are performed respectively. The ByteSub transformation employed in both the Round Transformation and Key Schedule was implemented as a look-up table (LUT). Since State bytes were operated on individually, each Rijndael round for a 128-bit block required sixteen 256 x 8 bit LUTs. In the Key Schedule, LUTs were also used when words were passed through the S-Box. A single BRAM could be configured into two S-Boxes, hence, eight BRAMs were used in each round. For the Key Schedule, since Round Keys were computed on-the-fly, two BRAMs were dedicated to Round Key generation. As a result, ten BRAMs were required for the Iterative Looping architecture.

ShiftRow in the Round Transformation and ByteRot in the Key Schedule require similar operations. They are both simply hardwired as no logic is involved. The MixColumn transformation is a matrix multiplication operation. Since the values in the square matrices are constant elements, this multiplication can be replaced by several EXOR operations that are simple to implement in the FPGA. Indeed, the operation

$Y = 03 \bullet X$, for X, Y in GF(2^8)

is implemented as below:

$y^7 = x^7$ EXOR x^6 $y^3 = x^7$ EXOR x^3 EXOR x^2
$y^6 = x^6$ EXOR x^5 $y^2 = x^2$ EXOR x^1
$y^5 = x^5$ EXOR x^4 $y^1 = x^7$ EXOR x^1 EXOR x^0
$y^4 = x^7$ EXOR x^4 EXOR x^3 $y^0 = x^7$ EXOR x^0

Also, the operation

$Y = 02 \bullet X$, for X, Y in GF(2^8)

is implemented as below:

$y7 = x^6$ $y^3 = x^7$ EXOR x^2
$y6 = x^5$ $y^2 = x^1$
$y5 = x^4$ $y^1 = x^7$ EXOR x^0
$y4 = x^7$ EXOR x^3 $y^0 = x^7$

I/O Buffer Design

From Fig. 4.1, it can be observed that an input buffer and output buffer were included in the design. Both of them were composed of 16 BRAMs and can store 512 128-bit data blocks. When a data block is written to Pilchard, it is placed in the input buffer which is configured as a circular buffer with separate read and write counters. As long as these two counters do not equal each other, the Control Unit can signal the Rijndael core to continue encryption. This process stops when all the data in the input buffer were read, thus implementing a first-in-first-out (FIFO) buffer. The writing and reading within the same RAM from two ports without conflict was supported by the dual port property of the BRAM. This feature increased the efficiency of the interface since transfers from the PC can be overlapped with computation in the Rijndael core.

Each ciphertext generated was stored in the output buffer. There was no flag to indicate the buffer is full and thus the software was responsible for reading the output buffer as soon as the ciphertext was ready. Similar to the input buffer, the Rijndael core could write data to the output buffer even when data was being read from the buffer.

5. UNROLLED DESIGN

Loop Unrolling is where n rounds are implemented as a combinatorial logic block. This method requires smaller number of clock cycles to perform an encryption, but results in a higher register-to-register delay, resulting in a lower clock frequency. The Rijndael core was implemented with different degrees of unrolling to compare the performance. Unrolling of degree one is the same as Iterative Looping. Unrolling of higher degree requires more hardware resources but reduces the number of cycles used to perform an encryption. It also introduces larger register-to-register delay, thus results in lower clock frequency.

In the Iterative Looping implementation of Rijndael described in the previous section, there were 11 rounds of iteration. The first round was an EXOR operation between plaintext and the key and was different from the other 10 rounds. As a result, only 10 rounds can be unrolled. In order that cycles have similar critical paths, the degree of unrolling must be divisible by 10. Hence, the possible degrees are 1, 2, 5 and 10.

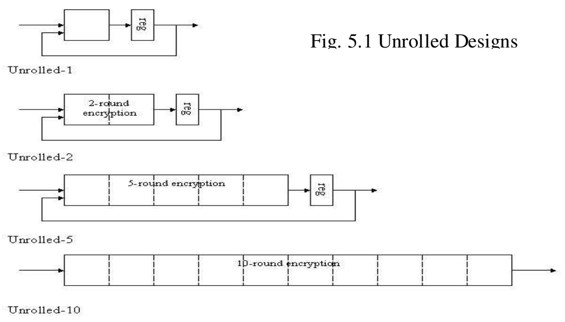

Fig. 5.1 Unrolled Designs

Figure 5.1 shows the flow of data in an unrolled design. The rectangular blocks represent one combinatorial round implemented in hardware. When the unrolling degree is one (Unrolled-1), only one round of iteration is implemented in hardware. Intermediate ciphertext is passed through the same hardware 10 times to perform one Rijndael encryption. When unrolled degree is two (Unrolled-2), two rounds of iteration are implemented in hardware. Ciphertext is passed through the same hardware 5 times to perform one Rijndael encryption. In an Unrolled-10 design, all the rounds are implemented and no iteration is required.

6. TRANSFER OPTIMIZATION

Pilchard, utilizing a DIMM memory slot in a PC, is treated as a memory device by the computer. Movement of data between Pilchard and CPU is identical to that between the real memory and the CPU. The efficiency of data transfers between memory and CPU depends on both the memory mode and the instruction set. The original Pilchard design used the MMX instructions in the Pentium processor to perform 64 bit transfer operations [7]. In this section, memory modes and the SSE instruction set are briefly reviewed.

Memory Mode

In Intel processors Memory Type Range Registers (MTRRs) are used to control processor access to memory. Different memory modes have different properties. Some common modes are Uncacheable, Write-combining, Write-through and Write-back. In this work, their properties were explored and the Rijndael system was tested using these memory modes.

According to IA-32 Intel Architecture Software Developers' Manual, there is no caching of system memory locations in Uncacheable mode. All reads and writes appear are executed in program order. There is no speculative memory access. This type of cache-control is useful for memory-mapped I/O devices. It is the default memory mode used in Pilchard but can be overridden by programming the MTRRs.

Similar to Uncacheable mode, the Write-Combining mode does not cache system locations. There are speculative reads, which involves reading the contents of target address as well as that of nearby addresses. It enhances performance when contents of continuous address are to be read. Writes may be delayed and combined in the write combining buffer until the next occurrence of a serializing event. This control mechanism greatly reduces memory accesses and can be used in application where order of writes is unimportant.

In Write-through mode, writes and reads of system memory locations are cached. There are also speculative reads. Writes are written to a cache line and through to system memory. Write combining is allowed. This type of cache control is appropriate when there are devices on the system bus that access system memory, but do not perform snooping of memory accesses. It enforces coherency between caches in the processors and system memory.

Similar to Write-through mode, there is caching of writes and reads of system memory locations, speculative read and write-combining in Write-back mode. The write-back memory mode reduces bus traffic by eliminating unnecessary writes to system memory. Writes to a cache line are accumulated in the cache. The modified cache lines are written to system memory when a write-back operation is performed. Write-back operations are triggered when cache lines need to be deallocated, such as when new caches are being allocated in a cache that is already full. They are also triggered by the mechanisms used to maintain cache consistency. This type of cache control provides best performance, but it requires all devices that access system memory on the system bus be able to snoop memory accesses to insure system memory and cache coherency.

The normal memory in a PC is set as Write-back, which gives best performance. However this memory type is not applicable to the Rijndael system. In normal memory, data retrieved from memory is exactly the same value that was written. As a result, caching can reduce access to memory. In the case of Pilchard, encryption is to be performed as a side effect of a memory access. Caching will eliminate necessary writes to Pilchard. When the user issues a read operation to get the ciphertext, the original plaintext which is not encrypted will be returned to user from the cache. Therefore both Write-through and Write-back type which cache read and write operations cannot be applied to the Rijndael system.

The Rijndael core was tested using Uncacheable and Write-Combining modes. Write-combining gives better performance because it combines write operations and performed speculative reads. The Rijndael core reads a buffer of input data and outputs a buffer of ciphertext for each encryption, so speculative reads will be correct if done after the whole output buffer is filled. The performance of the two memory types will be presented in detail in the "Results" section..

MMX Technology

In 1997 MMX Technology was launched by Intel to increase their processor's power to process multimedia application. MMX Technology introduces several new instructions based on a technique known as Single Instruction, Multiple Data (SIMD). SIMD means that a single instruction operates on multiple pieces of data in parallel. This allows programmers to pack several small chunks of data into a 64-bit register and then use a single instruction to command the CPU to perform a specific operation on each of those data elements. The MMX state consists of eight 64-bit registers (MM0 through MM7). Among all MMX instruction categories, only data transfer instructions were used. The MMX registers have two access modes, 64-bit access and 32-bit access. To transfer 128-bit data block, 64-bit access mode requires two memory accesses and 32-bit access mode requires four. Therefore the former was used to obtain better performance.

In this work, the instruction Move Quadword (*movq*) was used to transfer 64 bit data between memory and MMX registers, or among MMX registers.

SSE Technology

In 1999 when the Pentium III processor was launched, the most important enhancement was 70 unique instructions, called Streaming SIMD Extensions. Similar to MMX Technology, the Streaming SIMD Extensions use single-instruction, multiple-data (SIMD) capability to manipulate multiple pieces of data in parallel. Moreover, SSE improves performance by streamlining cache and memory access. The new instructions enable applications to pre-fetch specific data into the L2 cache from main memory. By doing so, applications can sidestep costly cache misses that can waste as many as 50 processor clock cycles while the system goes out to main memory after checking the L2 cache store. In addition, pre-fetching hides memory latency by allowing the system to perform pre-fetches even as data is being read into the processor. The SSE extensions introduced 8 new 128-bit XMM registers (XMM0 through XMM7) and 70 new instructions to the instruction set.

In this project, instructions for cache control were used to transfer data between Pilchard and the CPU. In particular, the Move Unaligned Four Packed Single-FP (movups) and Move Aligned Four Packed Single-FP Non Temporal (movntps) instructions were used. Both commands move four packed Single-FP from an XMM register directly to a location in memory. Movntps will bypass the cache hierarchy. These two commands were used to move data between an XMM register and a memory location, adapting the MMX-based technique described in [10] to use the SSE instructions and registers.

7. RESULTS

In this section, resource utilization and throughput of different unrolled designs are presented and the performance of our Rijndael implementation for different memory modes is compared. All results presented in this section were obtained using timing analysis and implementation reports generated by Xilinx Foundation Series 3.1i software.

Core Implementation

Figure 7.1 shows the increase in resource utilization as the degree of unrolling is increased. Note that the relationship is linear. For the Unrolled-10 case, the design was too large to fit on our target XCV1000 chip, but a mapping report could still be generated. For this case, no timing report could be generated so the performance of that design is not reported.

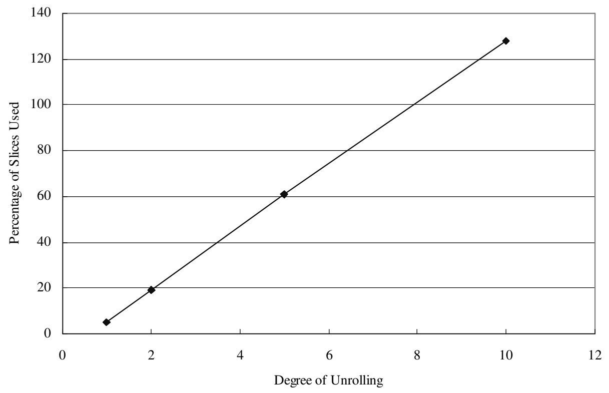

Fig. 7.1 Relationship between Slice Utilization and Degree of Unrolling

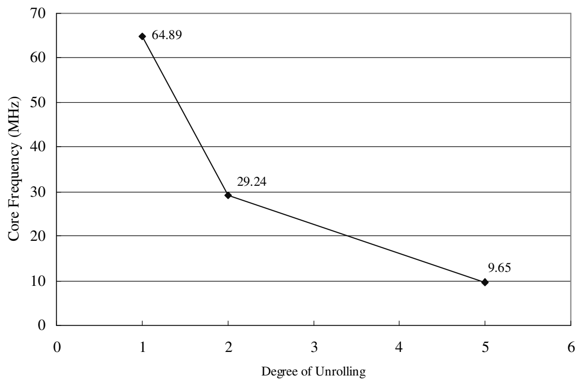

Fig. 7.2 Relationship between Core Frequency and Degree of Unrolling

As can be seen in Figure 7.2, as the degree of unrolling is increased, more combinatorial delays are introduced, resulting in a lower maximum frequency.

Figure 7.3 shows the throughput of each design. Although Unrolled-1 required 11 cycles to produce the ciphertext, it yielded the highest throughput amongst all the designs. Unrolled-2 and Unrolled-5 had similar throughput as the reduced number of cycles of Unrolled-5 was balanced by a reduction in operating frequency. Therefore, the Unrolled-1 implementation of the Rijndael algorithm achieves the highest throughput using the least amount of FPGA resources.

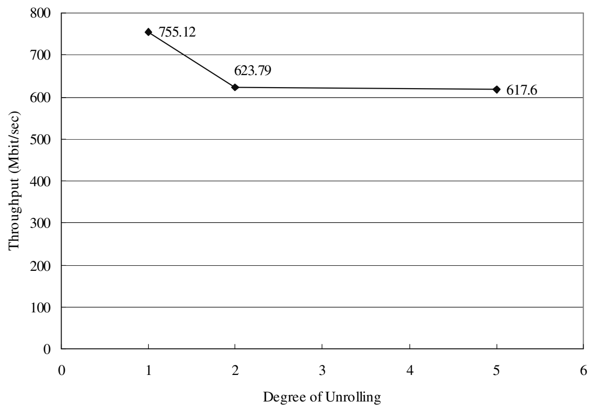

Fig. 7.3 Relationship between Throughput and Degree of Unrolling (with BRAM)

The maximum core frequency was 65 MHz. Since each round was finished in one clock cycle, 11 clock cycles were needed in a 128-bit key design. As a result, the throughput of the Rijndael core was 755 Mbit/sec. Throughput calculated by other researches is listed below in Table 7.1.

	Type	Device	Area (CLB)	Throughput (Mbit/sec)
Gaj [8]	IL	XCV1000BG560-6	2902	331.5
Dandalis [9]	IL	XCV1000	5673	353.0
the authors	IL	XCV1000HQ240-6	702	755.1
Elbirt [4]	P	XCV1000BG560-4	10992	1937.9
McLoone [3]	P	XCV3200EBG560-8	7576	3239.0
McLoone [3]	P	XCV812EBG560-8	2222	6956.0

[KEY]
IL = Iterative Looping
P = Pipelining

Table 7.1 Comparison of 128-bit Rijndael Encryption Implementations.

In Table 7.1 the throughput of this design is compared with other implementations. The authors' implementation was faster than that of Gaj's work because Gaj also implemented the decryptor in the chip. The author's design was also faster than a software implementation by Brian Gladman, which achieved 325 Mbit/sec on a 933MHz Pentium III processor. Pipelined designs have much higher performance than iterative designs. However, they cannot be used to implement feedback modes of operation such as cipher block chaining (CBC) in which a block is EXORed with the ciphertext of the previous block before being encrypted, creating a data dependency. CBC mode offers better security since the same plaintext is mapped to different a ciphertext depending on its context.

System Throughput

This throughput of the Rijndael core presented in the previous subsection was calculated from the maximum frequency specified in the timing analysis report. It is merely the highest speed of the encryptor core and is different to the System Throughout which takes I/O and software overheads into account.

The System Throughput of the Rijndael core on Pilchard was measured by encrypting data from a host PC. The testing environment was consisted of an Asus CUSL2 motherboard (Intel 815EP chipset) with 933MHz Pentium III processor and a Pilchard board.

When the software test program was started, all data blocks were are loaded into memory. Next, the initial time was recorded and data blocks were written to Pilchard. After filling the whole input buffer of the Rijndael core, the program read a buffer of ciphertext from Pilchard. Once the reading of the ciphertext was completed, the finish time was recorded and the time elapsed calculated. System Throughput was calculated by dividing the size of the data transfer by the time elapsed and was expressed in Mbit/sec.

The host processing interface was first implemented with MMX instructions. MMX Technology uses MMX registers which are 64 bits wide. This enables the transfer of 64-bit data between Pilchard and the PC. Plaintext of different block sizes was tested to give an average result as illustrated in Fig. 7.4 below.

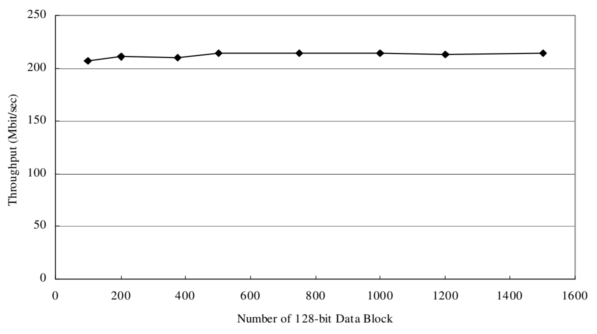

Fig. 7.4 Throughput of Rijndael Algorithm on Pilchard using MMX instruction

The System Throughput was approximately 210 Mbit/sec for data sizes from 1.5KB to 23KB. This throughput was much lower than that of the Rijndael encryption core and comparatively closed to Pilchard's read/write transfer rate in Uncacheable memory mode, which was 298 Mbit/sec. System Throughput was bottlenecked by the transfer rate of Pilchard in Uncacheable mode.

In order to improve transfer rate of Pilchard, the Write-combining memory mode was used. In addition, SSE instructions were used to read data from Pilchard to CPU to compare performance of MMX and SSE Technology. Two data blocks with size 1.5KB and 15KB were tested. The resulted System Throughput was depicted in the following graphs.

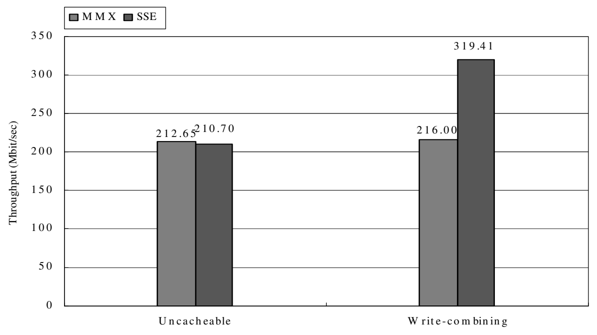

Fig. 7.5 System Throughput in different memory range, plain text size 1.5KB

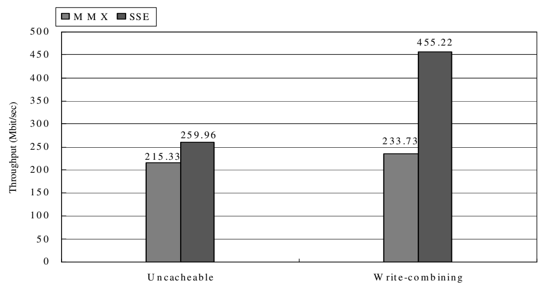

Fig. 7.6 System Throughput in different memory mode, plain text size 15KB

It can be observed from Figure 7.5 that in Uncacheable memory mode, MMX instruction and SSE instruction gave similar throughput of 210 Mbit/sec for small file sizes. However SSE instructions gave higher throughput for larger transfers whereas throughput remains the same for the MMX case.

In Write-combining memory mode, for both file sizes, MMX instructions had higher throughput than in Uncacheable mode. SSE instructions had much higher throughput, achieving 319 Mbit/sec in Uncacheable mode and 455 Mbit/sec in Write-combining mode.

From Figures 7.5 and 7.6, it can be seen that data transfer is always faster in Write-combining mode than in Uncacheable mode and that SSE instructions have generally higher performance than MMX instructions. The difference is larger for large data sizes.

From this test, it could be deduced that configuring Pilchard in Write-combining memory mode using SSE instruction yields the highest throughput. As a result, this configuration was used to produce Figure 7.7 which compares the SSE plus write combining mode with the original MMX plus uncacheable mode for different block sizes.

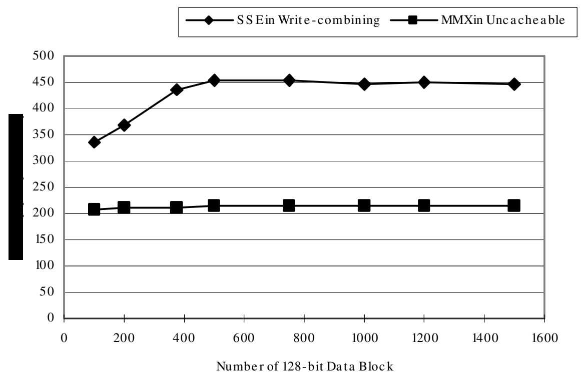

Fig. 7.7 System Throughput of Rijndael (SSE+WC vs MMX+uncacheable)

From Figure 7.7, it can be observed that throughput increases with the size of the data block. When data block size was larger than 400 128-bit values, throughput saturates at around 450 Mbit/sec. This was very close to the transfer rate of Pilchard in Write-combining mode, which is 483 Mbit/sec. This was also the highest system throughput achieved by this project and is significantly faster than the 280 Mbit/sec achieved by our previous DES implementation using Pilchard which employed MMX instructions and the uncacheable memory mode [7]. Note that this figure is still lower than throughput of Rijndael encryption core of 755 Mbit/sec and hence remains the bottleneck in our system.

8. CONCLUSIONS

The aim of this project was to develop a high performance system for data encryption using the Rijndael algorithm. An FPGA-based Rijndael core operating at a maximum frequency of 64 MHz was implemented. It was shown that placing a single round and iterating over the rounds (unrolling with a degree of one) gave higher throughput than unrolled implementations which complete in fewer cycles. Unrolling with degree one has the added advantage of requiring the least hardware resources.

To improve the system throughput of Rijndael encryption, which was bottlenecked by PC to FPGA transfers at 280 Mbit/sec, the Intel Pentium MMX and SSE technologies were compared. By using SSE instructions together with the write-combing memory mode, the measured system throughput was increased from 280 Mbit/sec to 450 Mbit/sec.

REFERENCES

[1] B.Schnier: Applied Cryptography. New York, New York, USA: John Wiley & Sons Inc., 2nd ed., 1996.

[2] B. Gladman: The AES Algorithm (Rijndael) in C and C++: URL: http://fp.gladman.plus.com/crytography_technology/rijndael/index.htm, 2001.

[3] M. McLoone, J.V McCanny: High Performance Single-Chip FPGA Rijndael Algorithm Implementations, CHES 2001, pp. 65-76.

[4] A.J. Elbert, E. Yip, B. Chetwynd, C. Paar: An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists, IEEE Transactions on VLSI, August 2001, vol. 9, no. 4, pp. 545-557.

[5] V. Fischer, M. Drutarovsky: Two Methods of Rijndael Implementation in Reconfigurable Hardware, CHES 2001, pp. 77-92.

[6] J. Daemen, V. Rijnmen: The Rijndael Block Cipher AES Proposal, Document version 2, September 03, 1999 (available from http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf).

[7] P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, K. H. Lee: Pilchard - A Reconfigurable Computer Platform with Memory Slot Interface, FCCM 2001 (to appear).

[8] K. Gaj, P. Chodowiec: Comparison of the Hardware Performance of the AES Candidates using Reconfigurable Hardware: The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, USA.

[9] A. Dandalis, V. K. Prasanna, J. D. P. Rolim: A Comparative Study of Performance of AES Candidates Using FPGAs: The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, USA.

[10] SGI, "Optimizing CPU to Memory Accesseson the SGI Visual Workstations 320 and 540", http://www.sgi.com/developers/technology/irix/resources/-asc_cpu.html