Reconfigurable Memory Based AES Co-Processor

Ricardo Chaves[1,2], Georgi Kuzmanov[2], Stamatis Vassiliadis[2], and Leonel Sousa[1]

[1]Instituto Superior Técnico/INESC-ID
http://sips.inesc-id.pt/
{ricardo.chaves, las}@inesc-id.pt

[2]Computer Engineering Lab, EEMCS, TUDelft
http://ce.et.tudelft.nl/
{G.Kuzmanov, s.vassiliadis}@ewi.tudelft.nl

Abstract

We consider the AES encryption/decryption algorithm and propose a memory based hardware design to support it. The proposed implementation is mapped on the Xilinx Virtex II Pro technology. Both the byte substitution and the polynomial multiplication of the AES algorithm are implemented in a single dual port on-chip memory block (BRAM). Two AES encryption/decryption cores have been designed and implemented on a prototyping XC2VP20-7 FPGA: a completely unrolled loop structure capable of achieving a throughput above 34 Gbits/s, with an implementation cost of 3513 slices and 80 BRAMs; and a fully folded structure, requiring only 515 slices and 12 BRAMs, capable of a throughput above 2 Gbits/s. To evaluate the proposed AES design, its has been embedded in a polymorphic processor organization, as a reconfigurable co-processor. Comparisons to state-of-the-art AES cores indicate that the proposed unfolded core outperforms the most recent works by 34% *in throughput and requires* 68% *less reconfigurable area. Experimental results of both folded and unfolded AES cores suggest over* 560% *improvement in the throughput/slice metric when compared to the recent AES related art.*

1 Introduction

In most of the current communication systems, privacy is a key requirement, which is typically achieved by the use of several encryption systems. In 2001, the National Institute of Standards and Technology (NIST) accepted the Rijndael algorithm as the Advanced Encryption Standard (AES) [12, 2]. This new AES has been introduced as the replacement for the old, but still used Data Encryption Standard (DES) [11]. Even though the AES is one of the most computationally efficient encryption algorithms, it is still very computationally demanding, and not able to achieve the throughput required by some applications when implemented in software. Motivated by the need of higher throughputs, several hardware designs of the AES algorithm have been proposed either for very high throughputs [5, 4, 8] or for more limited resource devices (achieving lower throughputs) [18, 17, 4]. However, these approaches implement the AES algorithm in a fine grain structure, requiring more hardware resources in a more complex structure, that reflects on a lower performance. This paper proposes a coarse grain AES design, employing the FPGA internal memories.

This research has been supported by the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC) project number IST-2003-004408.

Unlike other designs that use FPGA internal memories to implement only the byte substitution operation, in our proposal, we use these memory blocks to merge the byte substitution and the polynomial multiplication. This memory based structure allows an efficient AES encryption and decryption core implementation, and at the same time, potentially more resistent to DPA cryptanalyses attacks [7], due to the uniform power consumption of the memory blocks. More specifically, this paper presents two structures for the AES core: a fully folded one for area constrained implementations; and a fully unfolded structure meeting higher throughput requirements. Both AES cores have low hardware complexity and short critical paths. The design allows high throughput and low pipeline latency. The proposed AES core has been implemented within the reconfigurable co-processor of a Xilinx Virtex II Pro MOLEN prototype [15, 16]. The MOLEN polymorphic approach allows the core to be activated by a traditional software routine call, thus requiring practically no software development costs. More specifically, experimental results on the proposed standalone AES implementations indicate:

- High encryption/decryption efficiency and efficient hardware utilization:
  - 34 Gbit/s throughput AES core with 3513 slices and 80 BRAMs (9.9 Mbits per slice);
  - 2.3 Gbit/s throughput AES core with 515 slices and 12 BRAMs (4.6 Mbits per slice).
- Improvement to related-art:
  - 34% higher throughput;
  - 68% less reconfigurable area;
  - 560% improvement on the throughput/slice metric.

For the MOLEN polymorphic implementation, results suggest:

- Low FPGA utilization: just 10% occupation of a XC2VP20 device;
- Throughput of 1.2 Gbits/s;
- Minimal software integration costs.

The paper is organized as follows: Section 2 presents an overview on the AES algorithm as well as possible fine grain implementations of the several components of this algorithm. Section 3 describes the proposed BRAM implementation of the unfolded and folded versions of the AES core, as well as its polymorphic AES processor implementation. Section 4 presents the obtained experimental results and compares them to other state-of-the-art AES implementations. Section 5 concludes this paper with some final remarks.

2 The AES algorithm

The AES is the new NIST standard chosen to replace DES [11], it uses the Rijndael encryption algorithm with cryptography keys of 128, 192, 256 bits, the 128 bit key being the most commonly used. As in most of the symmetrical encryption algorithms,the AES algorithm manipulates the 128 bits of the input data, disposed in a 4 by 4 bytes matrix, with byte substitution, bit permutation and arithmetic operations in finite fields, more specifically, addition and multiplications in the Galois Field 2^8 (GF(2^8)). Each set of operations is designated by round. The round computation is repeated 10, 12 or 14 times depending on the size of the key (128, 192, 256 bits respectively).

AES Encryption: The coding process includes the manipulation of a 128-bit data block through a series of logical and arithmetic operations. In the computation of both the encryption and decryption, a well defined order exists for the several operations that have to be performed over the data block. The encryption process is depicted in Figure 1.

The following describes in detail the operation performed by the AES encryption in each round, introduced in Figure 1. The State variable contains the 128-bit data block to be encrypted.

SubBytes() transformation

The replacement of one set of bits by another is a non linear transformation, and is one of the most common operations in symmetrical encryption algorithms. In the Rijndael algorithm, this replacement is performed over a set of 8 bits. This replacement can be described by an affine transformation (over $GF(2)$), as presented in (1):

$$b_i' = b_i \oplus b_{(i+4)mod_8} \oplus b_{(i+5)mod_8} \oplus b_{(i+6)mod_8} \oplus b_{(i+7)mod_8} \oplus c_i \; ; \; 0 \le i < 8, \quad (1)$$

where b_i is the i-th bit of byte b(x) obtained from the State array. This byte substitution is performed over each byte individually. The c_i is the i-th bit of the value {01100011}. The byte substitution operation is usually implemented in hardware by a 256 bytes lockup table, with an 8 bit input and an 8 bit output.

ShiftRows()

The bytes in each row of the state matrix, are shifted to the left by $0, 1, 2$ or 3 byte positions, depending on the row where they are located, as depicted in Figure 2. For example $S_{1,0}S_{1,1}S_{1,2}S_{1,3}$ is transformed to $S_{1,1}S_{1,2}S_{1,3}S_{1,0}$. Since this operation contains no calculations, it can be implemented simply by routing the appropriate byte from the output of the previously described lockup table to the corresponding input of the MixColumns unit.

MixColumns() transformation

In this transformation, each column is treated as a four-term polynomial over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by:

$$a(x) = 03x^3 + 01x^2 + 01x + 02 \quad (2)$$

The resulting new column is thus calculated with the previous values of only that column. The calculation of the new column, presented in (3), is performed over the GF(2^8), where the multiplications ($\bullet$) are performed by AND operations and the additions and subtractions by XOR ($\oplus$) operations.

$$\begin{aligned} S_{0,c}' &= (02 \bullet S_{0,c}) \oplus (03 \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S_{1,c}' &= S_{0,c} \oplus (02 \bullet S_{1,c}) \oplus (03 \bullet S_{2,c}) \oplus S_{3,c} \\ S_{2,c}' &= S_{0,c} \oplus S_{1,c} \oplus (02 \bullet S_{2,c}) \oplus (03 \bullet S_{3,c}) \\ S_{3,c}' &= (03 \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (02 \bullet S_{3,c}) \quad (3) \end{aligned}$$

Since the multiplication of two bytes results in a double byte number, the result is replaced by the remainder

```
State = in

AddRoundKey(State, key[0 to Nb−1])
for round= 1, round<Nr, round=round+1 do
  SubBytes(State)
  ShiftRows(State)
  MixColumns(State)
  AddRoundKey(State,key[round×Nb to (round+1)×Nb−1])
end for

SubBytes(State)
ShiftRows(State)
AddRoundKey(State,key[Nr×Nb to (Nr+1)×Nb−1])

out = State
```

Figure 1. Pseudo Code for AES Encryption.

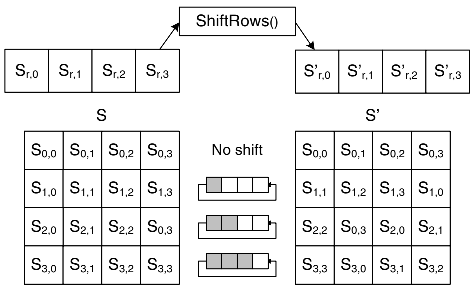

Figure 2. AES ShiftRows.

polynomial, that in the case of Rijdael is given by the irreducible polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{4}$$

This calculation can be performed by subtracting the $m(x)$ polynomial (value $1b$) whenever the result of each partial multiplication is bigger than FF. Finally, the addition of the four coefficients of the polynomial can be performed by XOR gates.

AddRoundKey()

The final operation to be performed in each round is the addition (XOR in $GF(2^8)$) of the respective round Key to each column of the State matrix. Each round Key consists of 4 32-bit words from the expanded Key (xK). The formalized operation is:

$$\begin{aligned}[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = &[S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus \\ &[xK_{round \times N_b + c}] \;\; ; \;\; 0 \leq c < N_b\end{aligned} \tag{5}$$

AES decryption: The decryption process is identical to that of the encryption (see the pseudo code in Figure 1). The main differences in the decryption computation lays on the byte substitution and on the polynomial equation used in the column mix. The byte substitution transformation for the decryption has the same structure as the encryption, only differing in the values of the look up table, presented in [2]. The row shifting is also identical, with the only difference that the rotation of the byte is performed to the right and not to the left, as depicted in Figure 2. In the inverse column mix transformation, the computation is exactly the same as in the encryption, differing only in the polynomial coefficient values. In the encryption, these coefficients result in a small hardware structure, since at most, only 2 bits are equal to 1 (the multiplications constants are $1, 2$ and 3). In the inverse column mix, the coefficients $(9, b, d, e)$ have 3 bits equal to 1.

3 Memory based implementation of AES

In a fine grain implementation of the AES algorithm, both the byte substitution (SBox) and the column multiplication would require specific and distinct hardware structures. According to (3), four structures are required per column. Implementing this structure in a LUT based architecture has a significant cost, not only due to the computational units, but also in the resources used in the interconnection. In programable devices, a significant improvement can be achieved by merging all computations into a single (on-chip) memory block [3]. Such a course grain solution is possible due to the fact that the individual resulting coefficients of the polynomial depend only on one of the bytes of the data block (or state matrix). In devices embedding true dual port memory blocks, such as the BRAM in the Xilinx FPGAs, 2 byte substitutions and 2 full multiplications can be mapped in a single memory block, as depicted in Figure 3. Each full multiplication multiples one byte ($S_{i,c}$) by a set of constants: $\{1, 1, 2, 3\}$ for the encryption and $\{9, e, b, d\}$ for the decryption.

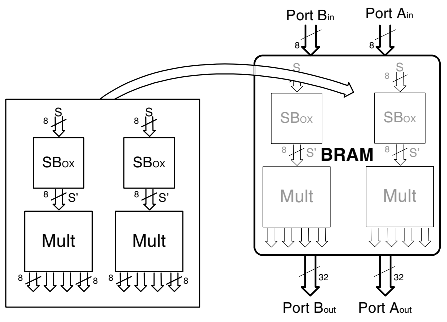

Figure 3. Coarse grain column computation using BRAM

Since the byte permutation required in the row shift, is a fixed operation, it is performed simply by routing the values to the respective memory block. Only the remaining additions have to be performed in a fine grain organization (e.g. in LUTs). For each byte in a given column, 4 additions have to be performed: 3 to add the polynomial coefficients and 1 for the key addition. It should be noted that because the byte substitution operation does not depend on the bytes position in the state matrix, the ShiftRow operation can be performed before the SubBytes (see pseudo-code in figure 1).

Last round calculation: The last round of the AES computation has the particularity of not computing the polynomial multiplication, as illustrated in the pseudo-code in Figure 1. This can be computed with the memory structure previously presented, which only performs the byte substitution operation. Also the output value is directly added to the key, since no polynomial addition has to be performed. For reconfigurable area efficiency, the last round is computed using the same memory blocks as the inner rounds. In the encryption, the byte substitution can be obtained directly since this value is equal to the multiplication by 1, given by the memory computation. In the decryption, however, all four results are multiplied by coefficients that are different from 1 (i.e. $\{9, e, b, d\}$). In order to obtain the original value before the multiplication, the logical operation $1 = b \oplus d \oplus e \oplus 9$ can be performed on the four 8-bit outputs of the memory blocks, at the cost of three 8-bit XOR gates.

Encryption and decryption AES rounds: As previously mentioned, the two major differences between the encryption and decryption algorithms are the byte substitution operation and the polynomial multiplication coefficient values. In our memory based implementations, these two differences are located in the memory blocks. Thus, by changing the lookup values in the memories, the computation can change either to encryption or to decryption. For better hardware efficiency, both encryption and decryption can be merged into a single memory block [10]. The encryption/decryption memory block requires 2048 addressable bytes ($2 \times 32 \times 2^8 = 1024$ bytes). The new memory address is given by a byte, of the state matrix, and an additional bit, indicating whether the operation is encryption or decryption, as depicted in Figure 4.

The last difference between the encryption and the decryption process resides on the byte permutation

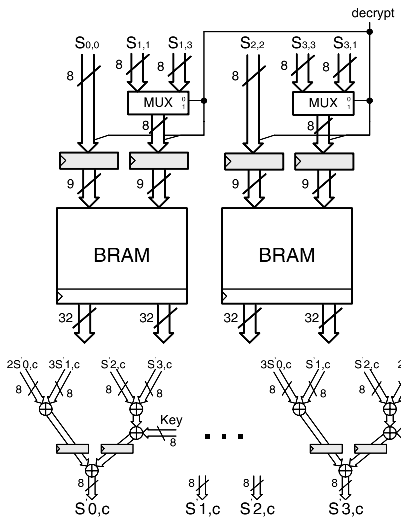

Figure 4. AES partial Encryption and decryption round

performed by the shift row transformations, as depicted in Figures 5. The corresponding byte permutation can be selected by multiplexing the two possible values. After the byte permutation only half of the new byte positions are different (depicted by the shaded rows in Figure 5). Thus, each resulting column calculation only requires the multiplexing of half of the byte values to select between encryption and decryption.

The AES core: To compute the AES algorithm, the round calculations have to be performed a predefined number of times depending on the key size. This can be done either by having one hardware structure per each AES round or by reusing the hardware of the rounds. The first approach is refereed to as unfolded loop and the second one, a fully unrolled loop approach by folded loop.

Unfolded loop AES core: When a high throughput is required, pipelined versions of the fully unrolled approach can be used (referred to as AES unfolded core). In the the AES unfolded core, each round is computed in it's own dedicated structure. Such a structure can be implemented by connecting the output of one round to the input of the following round, as depicted in Figure 6. Since the reconfigurable devices internal memory blocks are synchronous and have registered outputs, the inter-round pipeline consists of the synchronous memory blocks output. Additional pipelining can be introduced, by inter-round registers. These registers

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{0,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Original

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{0,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Encryption

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,3}$	$S_{1,0}$	$S_{1,1}$	$S_{1,2}$
$S_{2,2}$	$S_{0,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,0}$

Decryption

Figure 5. Byte permutation in the row shifting

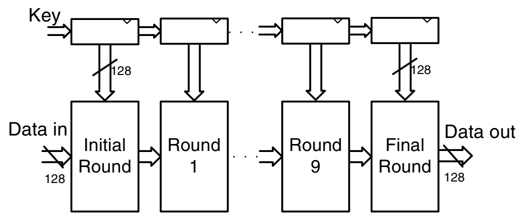

Figure 6. AES unfolded core

can be easily introduced after the memory blocks and anywhere in the additional logic, as depicted in Figure 6 by the shadowed registers. *Folded loop AES core:* When the throughput requirements are not critical or the available hardware is limited, a folded versions of the core can be used. In the fully folded design, the throughput is significantly lower, due to the reuse of the hardware structure and due to a longer critical path. The longer critical path is imposed by the additional multiplexer unit required to reinsert the value of the previously calculated round or the new data block, as depicted in Figure 7. The primary design tradeoff is between performance and hardware resources. The rolled structure can also be advantageous when other commonly used encryption modes are used, such as Cipher-Block Chaining (CBC) and Cipher Feedback (CFB) [14]. In these modes, the next data block can only be encrypted after the previous one has been coded. For these modes, the existence of a pipeline (e.g. in the unfolded structures) becomes inefficient, since it would be almost always empty. The folded structure, on the other hand, has the round hardware always in use. Figure 8 depicts a variation of the folded design to encrypt and decrypt in Electronic CodeBook (ECB) and CBC modes (where the Initialization Vector (IV) is used). Since the difference in the AES algorithm for the 128, 192 and 256 is the number of times the rounds are executed, the folded AES core is able to cipher in these 3 modes, simply by switching the reset value of

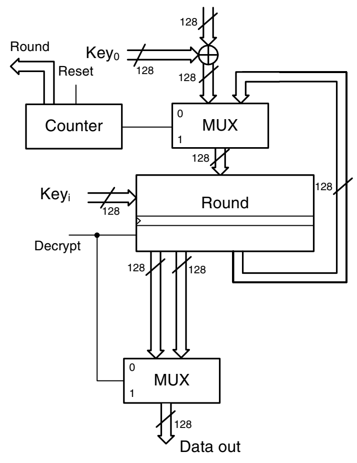

Figure 7. AES folded core

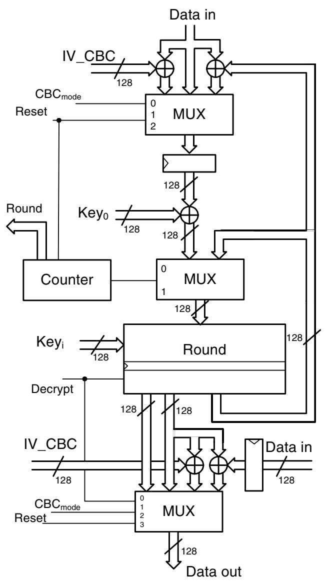

Figure 8. AES folded core with ECB and CBC

the internal counter (to 10, 12 or 14). In the folded design, the expanded key is only accessed in blocks of 128 bits, which is the corresponding size for each round. In these cases the key register can be accessed as an addressable register bank, which can be implemented with memory blocks. To better use the dual encryption/decryption capability of the AES, the key register (implemented with memory blocks) can store both the encryption and the decryption keys. Like in the round calculation, the differentiation of the cipher mode is done by adding the encryption/decryption signal to the register address. In devices with memory blocks having less than 128-bit output ports, the memory based register has to be implemented with several memory blocks. Figure 9 depicts such an implementation for a XILINX FPGA using 32-bit output BRAMs.

AES polymorphic processor prototype: In order to evaluate the AES core in real applications, a prototype has been developed and embedded in the MOLEN polymorphic processor [16]. The MOLEN paradigm [15] is based on the coprocessor architectural paradigm. More specifically, a reconfigurable coprocessor is configured for an application specific operation, while the main program is executed on a General Purpose Processor (GPP). The prototyping platform technology is the XILINX Virtex II Pro, which embeds a PowerPC as a GPP. In order to use this core,

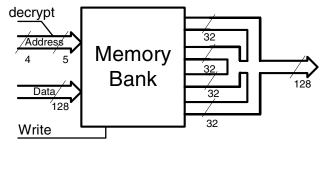

Figure 9. AES folded key register

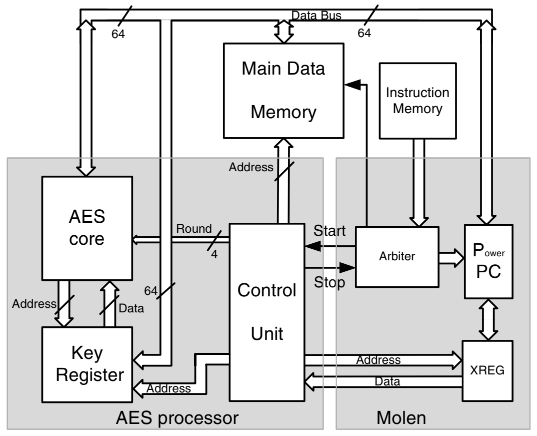

Figure 10. AES polymorphic processor

the pre-existing software applications are compiled by a specialized compiler [13]. Thus, for the software programmer, the usage of the reconfigurable coprocessor is transparent, it is used as if the function were implemented in software. This capability allows the coprocessor to be used in any existing application with minor modifications to the already existing software applications. Figure 10 illustrates the MOLEN organization with the AES polymorphic prototype. Unlike other existing AES coprocessors [6, 9], the proposed coprocessor allows to speedup any application that uses AES encryption or decryption with virtually no software development costs. While in software functions the parameter passing is done through the stack, in the MOLEN processor these parameter are passed through a dedicated exchange register file, designated by XREG, depicted in Figure 10. Identically to a software function, the data to be ciphered by the coprocessor, is accessed directly from the main data memory.

The specialized compiler transfers the function parameters, either from the stack or from the XREG, depending if the functions is implemented in software or hardware. During runtime the arbiter depicted in Figure 10, decides if the instruction is to be executed in the main processor or in the co-processor. The control unit is responsible for retrieving the initialization data and the data blocks from and to the main data and the XREG. This control unit reads the memory pointers from the XREG and addresses the main data memory in order to initialize the AES core internal registers. The initialization values are the key addresses and the begin and end address of the data blocks to be ciphered. The control unit has been described in a fully parameterizable fashion, allowing structural modifications to be performed simply by altering a set of constants. These structural modifications include: the use of a main data memory with different dimensions, influencing the way the data is read; distinct memory architectures types that require different timing; different levels of loop unfolding, increasing the latency of the data throughput.

The proposed implementation uses a fully folded loop version of the AES core capable of encrypting and decrypting data blocks in both ECB and CBC modes for all key sizes, i.e., for 10, 12 or 14 rounds. The operation modes as well as the Initialization Vector for the CBC mode are passed as additional parameters in the XREG.

4 Performance analysis and related work

This section evaluates the proposed AES implementations and the polymorphic AES co-processor. It also presents comparative analysis to related work. The prototype systems were implemented in a Xilinx Virtex II Pro (xc2vp20-7) on an Alpha Data: (ADM-XPL) development board using the ISE (6.3) tools from Xilinx. The system is using the embedded PowerPC, capable of running at a maximum frequency of 300 MHz. The used main memory has a maximum working frequency of 100MHz.

Implementation results: Table 1 presents the implementations results (after Place & Route) of the unfolded AES, with intra-pipelining (intra-pp) and without intra-pipelining(inter-pp), and the folded AES core. Note that all the proposed cores are capable of performing both encryption and decryption.

Due to the usage of the internal BRAMs, it is possible to obtain very high throughputs for a relatively low FPGA area utilization, namely a throughput above 34 Gbits/s for an 36% and 80 BRAMs occupation of a XC2VP20 FPGA. With the folded design a throughput of above 2 Gbits/s is obtained for a occupation of only 5% and 12 BRAMs, which already includes the additional logic and the 4 BRAMs used to implement the expanded key register. With different synthesis and routing tools some aspects of the implementations might be further improved. In Figure 4, the intra-register pipeline that precedes the BRAM block can be included in the BRAM itself, since these components can be configured to have registered inputs. However the ISE 6.3, reported worst implementation results for this solution. Since the unfolded AES core is implemented as a sequence of registered rounds, the maximum frequency of the unfolded core without intra-pipelining should be at least as high as the folded AES core.

The proposed prototype of the polymorphic AES processor was implemented with a fully unfolded loop AES core, capable of encryption and decryption in both ECB and CBC encryption modes. The implementation of this system requires 1130 slices (12%) and 12 BRAMs with a maximum frequency of 100 MHz. The maximum frequency is imposed by the main data memory and not by the AES core. An additional hardware round can be included at a cost of 160 slices and 8 BRAMs. Table 2 presents the throughput results using a 128-bit key, for the full software and the hybrid software/hardware implementations. The speedup obtained with the AES core, when compared to the software implementation of the AES algorithm is also presented.

Table 1. AES implementation results

	Slices	BRAM	(MHz) Freq.	(Gbps) ThrPut.	(cycles) Lat.
folded	515(5%)	12	182	2.33	10
unfolded (inter-pp)	3168(32%)	80	156	19.95	10
unfolded (intra-pp)	3513(36%)	80	271	34.7	30

Table 2. AES polymorphic performances

	Hardware		Software		
Bits	Cycles	(Mbps) ThrPut	Cycles	(Mbps) ThrPut	Kernel SpeedUp
128	646	59	24216	1.59	43
4k	4366	281	738952	1.66	169
128k	31246	1258	23610504	1.67	751

The obtained speedup is just 43 when only one 128-bit data block is encrypted, due to the overhead to transfer the expanded key (1408 bits). A speedup of 571x is accomplished for a file with 16kBytes. Note however, that the overhead of the expanded key transfer is already not significant, especially when considering the encryption of large files. The last column of Table 2, contains the calculated speedup for the ciphering kernels. The number of cycles is the same for encryption and decryption, for both the software and hardware implementations. Note that speedups are for the ciphering subroutine (kernel) only and not of the entire application. In a practical application, even if only one data block is ciphered in every function call, the expanded key remains the same. Since the AES core internal registers are not cleared each time a new encryption is executed, they work as static variables. This means that the expanded key has to be transferred only once. In this case, for the ciphering of a single 128-bit data block per function call, a hardware throughput of 89 Mbits/s is obtained, resulting in a local kernel speedup of 56x. Finally, a speedup above 750x is obtained with small resource utilization (1130 slices and 12 BRAMs) and at practically no software development costs.

Related work: In order to analyze the obtained results, the following compares the obtained figures for the fully unfolded design, the fully folded and the hybrid AES processor to other related work.

Folded AES core

In Table 3, the figures of the proposed folded AES core are compared to the state-of-the-art research and the most recent commercial products. In order to compare our design to the semi unfolded core proposed in [17], ours was unfolded once to obtain an identical throughput. Even though our design requires 2k-Byte RAMs and focused on a different FPGA technology (a Virtex II Pro), it was implemented in an FPGA with 512-byte internal RAMs. Even in this less favorable case our AES core outperforms [17] by 560% on the Throughput/Slice metric.

When compared to the Helion [4] folded core implemented on a Virtex II Pro, our core outperforms it by 91%. However, the encryption/decryption AES core from Helion, is designed to perform the encryption and decryption calculation, with independent inputs and outputs, which make it less usable as a single AES core. If these separate inputs/outputs are combined to make a *real* encryption/decryption AES core, the oc-

Table 3. AES folded core performance comparisons

Architecture	Jhing [17]	Ours	CAST [1]	Helion [4]	Ours
Cipher	Enc./Dec.	Enc./Dec.	Enc./Dec.	Enc.&Dec.	Enc./Dec.
Device used	XCV812E	XCV812E	XC2VP20	XC2VP20	XC2VP20
Speed grade	n.a.	-8	-7	-7	-7
Number of slices	3046	431	928	1016	515
Number of BRAM	280	32	8	18	12[1]
Max. frequency (MHz)	61	67	122	198	182
Latency (cycles)	11	10	11	11	10
Throughput (Gbps)	1952	1718	1415	2304	2332
Throughput/Slice (Mbps/s)	**0.6**	**4.0**	**1.5**	**2.3**	**4.6**

cupation and throughput will most likely be affected, as shown in [8]. It should be noticed that the Helion core has the key expansion in the core itself, while in our approach the key expansion is performed outside the core.

When compared to one of the leading (real) encryption/decryption AES cores on the market, the CAST [1], implemented on a Virtex II Pro, our AES folded core is over 200% more efficient, in the Throughput/Slice metric.

Unfolded AES core

Table 4 presents the figures for the fastest hight throughput AES cores, including the proposed unfolded AES core. Note that some of the considered cores are only able to perform AES encryption and not AES decryption. In [8], the AES encryptrion/decryption core is implemented by one encryption unit and another decryption unit. Thus, although not explicitly presented in the paper, the required hardware resources have to be at least twice those of the single encryption core ($2 \times 5408 = 10816$ slices).

Our intra-pipelined AES core (Ours-P30) proposal is capable of achieving a 34% faster throughput, than the Giga AES core from Helion [4], which in turn is capable of a maximum throughput of 25 Gbits/s. No further comparisons with [4] are be made, since no more figures have been made available by Helion. In implementations, where the output latency is critical, the unfolded AES core without intra-pipelining (Ours-P10) is able to achieve a frequency 24% faster than [8], with only 17% of the output latency of [8]. Even when compared to the architectures than only perform encryption, this AES encryption/decryption core outperforms any of the existent cores in speed and in area occupation, resulting in an improvement of the Throughput/Slice metric of almost 100%.

When compared to the existing cores, reported in [8], that are also capable of performing both encryption and decryption, our intra-pipelined core achieves a throughput more than 115% higher, with 68% less reconfigurable area. Thus, achieving a 560% better Throughput/Slice metric by more than , while the output latency is reduced by half.

AES polymorphic processor

The developed hybrid processor is compared to a recently published, computationally identical processor. The figures presented in Table 5 show an improvement on the area occupation while maintaining an identical

Table 5. AES hybrid implementation

Architecture	Lu [9]	Ours
Cipher	Enc./Dec.	Enc./Dec.
Operation modes	ECB, CBC	ECB, CBC
Device used	XC2VP100	XC2VP20
Main processor	PowerPC	PowerPC
Speed grade	n.a.	-7
Number of slices	1700[2]	847
Number of BRAM	44	12
Max. frequency (MHz)	196-179	100[3]
Throughput AES-128	1197 Mbps	1258 Mbps
Throughput AES-256	778 Mbps	905 Mbps
Throughput/slice (Mbps/s)	0.7	1.5

throughput for the 128-bits key cipher mode. However, for the ciphering using a 256-bit key, the implementation proposed in [9] suffers from a frequency degradation. On the contrary, in our proposal, the hybrid processor has already been implemented in a design capable of cipher with all key sizes at the same frequency. For the AES ciphering with a 256-bit key, our proposal obtains a 16% higher throughput. The AES version for 256-bit keys in [9] implies an area increase and a frequency decrease, in regard to their 128-bit AES core, thus reducing the Throughput/slice performance metric. The slice number value presented in Table 5 depicts only the figures for the AES core (with ECB and CBC) for both processing units. The interface hardware overhead is excluded form the presented overall synthesis results for both processors. The Throughput/Slice ratio given in Table 5 relates to the most advantageous case for the processor proposed in [9]. Even in this case, this metric is 114% higher for our co-processor. For the 256-bit key ciphering, our Throughput/Slice efficiency is even higher. Overall, the performance of our core is clearly advantageous when compared to the existing art.

5 Conclusion

We proposed a new approach to the implementation of the AES algorithm in hardware, where by merging the ByteSub operation and the polynomial multiplication operations and computing them in a single lookup

[1]This 12 BRAMs includes the 4 BRAMs used to implement the expanded key register.

[2]A estimated value for the slice utilization has been used, for a ratio of 0.62 Slices per LUT. The authors in [9] only give the number of utilized LUTs. Our AES core has a 0.62 Slice/LUT ratio.

[3]100 MHz is the maximum frequency imposed by the main data memory and not of the AES core itself, that is capable of running at higher frequencies.

Table 4. AES unfolded core performance comparisons

Architecture	Hodjat [5]	Kotturi [8]	Kotturi [8]	Ours-P10	Ours-P30
Cipher	Encryption	Encryption	Enc./Dec.	Enc./Dec.	Enc./Dec.
Device used	XC2VP20	XC2VP70	XC2VP70	XC2VP20	XC2VP20
Speed grade	-7	-7	-7	-7	-7
Number of slices	5177	5408	10816	3168	3513
Number of BRAM	84	200	400	80	80
Max. frequency (MHz)	168	233	126	156	272
Latency (cycles)	41	60	60	10	30
Throughput (Gbps)	**21.54**	**29.77**	**16.08**	**19.95**	**34.76**
Throughput/Slice (Mbps/s)	**4.2**	**5.5**	**1.5**	**6.3**	**9.9**

memory bank significant improvements were achieved. This approach can also be used in other reconfigurable technologies as well as in ASIC implementations. The simplicity of the proposed implementations suggests a high improvement over existing state-of-the-art AES cores, in terms of area and critical path improvement. The later directly reflects to an increase of the core throughput. Since the major differences between the encryption and the decryption computations are located in the memory blocks, an AES core employing both encryption and decryption can be implemented at a reduced hardware cost when compared to cores that only implement either encryption or decryption. Another advantage of the proposed design is that due to the uniformity, simplicity and regularity of the design, the power consumption is lower and more uniform. This paper also proposed a polymorphic AES processor capable of speeding up the AES ciphering kernel by 751x, at the cost of 1130 slices and 12 BRAMs. The polymorphic computational approach, allows the AES co-processor to be used at a minimum software development costs. To our best knowledge, the proposed core achieves a maximum throughput 34% faster than any existing AES core we know about, and requires 68% less reconfigurable hardware and half the pipeline output latency of equivalent co-processors. These result suggest an improvement on the Throughput/Slice metric of more than 200% for the folded core, when compared to the state-of-the-art research and leading commercial products. For the unfolded structures, the Throughput/Slice metric is improved 560%, achieving a throughput higher than 34 Gbits/s.

Evaluation prototype

An evaluation prototype for the XUP and AD-XPL prototyping boards of the hybrid AES processor is available for download at http://ce.et.tudelft.nl/MOLEN/aplications/AES/

References

[1] CAST. AES128-P Programmable Advanced Encryption Standard Core. http://http://www.cast-inc.com/, 2005.

[2] J. DAEMEN and RIJMEN. The design of rijndael. AES-the advanced encryption standard. *Springer-Verlag*, 2002.

[3] V. Fischer and M. Drutarovsk. Two methods of rijndael implementation in reconfigurable hardware. In *Cryptographic Hardware and Embedded Systems, CHES 2001: Third International Workshop*, volume 2162, pages 77–92, January 2001.

[4] HELION. High Performance AES (Rijndael) cores for Xilinx FPGA. http://www.heliontech.com/, 2005.

[5] A. Hodjat and I. Verbauwhede. A 21.54 Gbits/s fully pipelined AES processor on FPGA. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 308 – 309, April 2004.

[6] A. Hodjat and I. Verbauwhede. Interfacing a high speed crypto accelerator to an embedded cpu. In *Proc. 38th Asilomar Conference on Signals, Systems, and Computers*, volume 1, pages 488–492, November 2004.

[7] P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks, 1998.

[8] D. Kotturi, S.-M. Yoo, and J. Blizzard. AES Crypto Chip Utilizing High-Speed Parallel Pipelined Architecture. In *IEEE International Symposium on Circuits and Systems*, pages 4653 – 4656, May 2005.

[9] J. Lu and J. Lockwood. IPSec Implementation on Xilinx Virtex-II Pro FPGA and Its Application. In *Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium*, pages 158b – 158b, April 2005.

[10] S. Morioka and A. Satoh. A 10 gbps full-aes crypto design with a twisted-bdd s-box architecture. In *ICCD*, pages 98–103. IEEE Computer Society, 2002.

[11] NIST. Data encryption standard (DES), FIPS 46-2 ed. Technical report, National Institute of Standards and Technology, December 1993.

[12] NIST. Announcing the advanced encryption standard (AES), FIPS 197. Technical report, National Institute of Standards and Technology, November 2001.

[13] E. M. Panainte, K. Bertels, and S. Vassiliadis. The PowerPC Backend Molen Compiler. In *14th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 434–443, September 2004.

[14] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Jonh wiley & Sons, inc, second edition, 1996.

[15] S. Vassiliadis, S. Wong, and S. D. Cotofana. The Molen $\rho\mu$-coded Processor. In *11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, pages 275–285, August 2001.

[16] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte. The Molen polymorphic processor. *IEEE Transactions on Computers*, pages 1363– 1375, November 2004.

[17] J.-F. Wang, S.-W. Chang, P.-C. Lin, and C. Kung. A novel round function architecture for AES encryption/decryption utilizing look-up table. In *IEEE 37th Annual 2003 International Carnahan Conference on Security Technology*, pages 132– 136, October 2003.

[18] S.-S. Wang and W.-S. Ni. An efficient FPGA implementation of advanced encryption standard algorithm. In *Proceedings of the 2004 International Symposium on Circuits and Systems*, volume 2, pages 597–600, May 2004.