
© 2005 The Institution of Engineering and Technology (IET)

Reprinted with permission.
Comparative survey of high-performance cryptographic algorithm implementations on FPGAs

K. Järvinen, M. Tommiska and J. Skyttä

Abstract: The authors present a comparative survey of private-key cryptographic algorithm implementations on field programmable gate arrays (FPGAs). The performance and flexibility of FPGAs make them almost ideal implementation platforms for cryptographic algorithms, and therefore the FPGA-based implementation of cryptographic algorithms has been widely studied during the past few years. However, a complete analysis of published implementations has not been presented previously. The authors analyse FPGA-based implementations of certain widely used cryptographic algorithms in terms of speed, area and implementation techniques. The algorithms studied in this article include the private-key cryptographic algorithms advanced encryption standard and international data encryption algorithm and certain hash algorithms. These algorithm implementations provide a good overview of the field of private-key cryptographic algorithm implementation.

1 Introduction

This article presents a thorough study of the state of private-key cryptographic algorithm implementation on field programmable gate arrays (FPGAs). As cryptographic algorithms become more widely used, the need for high-speed implementations of the algorithms increases. Software-based implementations of cryptographic algorithms fall short in performance in many applications, e.g. on heavily loaded servers. Therefore, an obvious need for high-speed implementations exists.

Reprogrammable hardware is almost ideal for cryptographic implementations because high speed can be achieved without significant reduction in flexibility. Flexibility, meaning that the design can be easily changed or modified, is of especially great importance in cryptographic implementations for the following reasons. First, a cryptographic algorithm can be considered secure only until proven otherwise. If a severe flaw in an algorithm is found, the algorithm must be replaced with a more secure one. Second, in many applications, a large variety of different algorithms are in use, and, therefore, it must be easy to change from one algorithm to another.

This article concentrates on implementations of the advanced encryption standard (AES), the international data encryption algorithm (IDEA) and certain hash algorithms. These algorithms represent both private-key cryptographic algorithms and hash algorithms, thus giving a good overview of the state of cryptographic algorithm implementation. All these algorithms have been implemented by the authors of this article in the Signal Processing Laboratory at Helsinki University of Technology, and these designs are here referred to as the SIG designs, e.g. SIG-AES.

Although FPGA-based cryptographic algorithm implementation has been widely studied during the past few years, a thorough comparative study of published implementations has not been presented, at least to the authors’ knowledge. The article by Wollinger et al. [1] included a review of implementations, but otherwise the article concentrated more on security questions of FPGAs as implementation platforms. In this article, implementations of cryptographic algorithms are compared in terms of speed, area and implementation techniques. Finally, certain conclusions on cryptographic algorithm implementation on FPGAs are presented.

2 Private-key cryptographic algorithms

Implementation of private-key cryptographic algorithms on reprogrammable hardware has been widely studied for several years. Block ciphers are well suited to hardware implementation, because parallelisation, unrolling and pipelining can usually be efficiently exploited.

Throughput can be increased by pipelining an unrolled design and then calculating a different encryption in each pipelined stage. Pipelining, unfortunately, restricts the use of feedback cipher modes which require the value of the previous ciphertext in the generation of the next one, e.g. cipher block chaining mode [2].

The data encryption standard (DES) and its variant 3DES were the most popular block ciphers for decades. Some recent papers on DES implementation have been published, e.g. [3] and [4]. However DES is currently being replaced by AES, and the role of DES will be marginal in the future. Thus, DES is not considered further.
2.1 Advanced encryption standard (AES)

AES is a NIST (National Institute of Standards and Technology) standard introduced in 2001 [5]. AES was developed by two Belgians, Vincent Rijmen and Joan Daemen, and it was originally named Rijndael. AES processes a 128-bit data block with a key of either 128, 192 or 256 bits. The different versions of AES are here referred to as AES-128, AES-192 and AES-256. The 128-bit data block is represented as $4 \times 4$ rectangular array of bytes called the State. Depending on the key size, AES consists of either 10, 12 or 14 rounds which include four transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey (the last round does not include MixColumns), where the rows and columns refer to the rows and columns of the State. Keys for every round are derived from the original cipher key using the KeyExpansion routine [5].

The SubBytes operation is the most crucial for both the speed and area requirements of an AES implementation [6]. It operates independently on each byte of the State, and it consists of finding a multiplicative inverse in the Galois field $GF(2^8)$ followed by an affine transformation [5]. It operates on each byte of the State, and therefore a total number of 200 S-boxes are required in a fully unrolled key agile AES-128 implementation. If S-boxes are implemented on Xilinx FPGAs, 100 BlockRAMs are needed, because one dual-port BlockRAM can implement two S-boxes. BlockRAM-based S-boxes have been used in many publications, e.g. [13, 15, 17–23].

Different S-boxes are used in SubBytes, and InvSubBytes, which makes the combination of encryption and decryption difficult without doubling the BlockRAM need. McLoone and McCann presented in [20] AES-128, AES-192 and AES-256 implementations combining encryption and decryption. They introduced two ROMs including S-box values for encryption and decryption. The BlockRAMs implementing SubBytes and InvSubBytes were programmed using values from ROMs every time encryption was changed to decryption or vice versa [20]. Another solution was presented by Rodriguez-Henriquez et al. in [16]. They combined SubBytes and InvSubBytes so that the inversion in $GF(2^8)$ (utilised by both) was implemented in BlockRAMs, but the affine transformations were implemented with logic. This allowed the same BlockRAMs to be utilised in encryption and decryption.

In addition to (Inv)SubBytes, the (Inv)MixColumns transformation can also be performed using the LUT approach. An LUT combining (Inv)SubBytes and (Inv)MixColumns is called the T-box. Fischer and Drutarovsky studied implementation techniques based on S-boxes and T-boxes on an Altera FPGA in [24]. They concluded that slightly faster performance was attained with the T-box approach, but the memory need increased [24]. In [25], McLoone and McCann presented an AES-128 encryption implementation utilising T-boxes. Their implementation had high throughput and occupied only a small number of slices, but it required a very large number of BlockRAMs. A device with a large amount of embedded memory, e.g. the Virtex-E Extended Memory [26], is therefore required.

In the above implementations, SubBytes was implemented as a LUT. Another approach is to calculate the multiplicative inverse and the affine transformation using combinatorial logic. Inversion in $GF(2^8)$ can be reduced into an inversion in $GF(2^4)$ or in $GF(2^2)$ accompanied by Galois field field additions and multiplications. That is, the problem is mapped from $GF(2^8)$ to another representation of the field, in which cases is either $GF(2^4)^2$ or $GF((2^2)^2)^2$. This approach is here referred to as combinatorial implementation of SubBytes, and certain implementations using such methods are considered next.

Several publications have presented studies of unrolling and pipelining, e.g. [6, 13–19]. The previously mentioned AES finalist algorithm implementations also considered unrolling and pipelining [9, 11]. Very high throughput can be achieved by pipelining unrolled rounds of the algorithm, but, as mentioned, pipelining cannot be efficiently used in feedback modes. To the authors’ knowledge, the fastest published FPGA-based implementation of AES was presented by Zambreno et al. in [19]. They used aggressive pipelining and achieved throughput of 23.57 Gbps on a Xilinx Virtex-II XC2V4000.

SubBytes can be implemented as an S-box (LUT) which includes precalculated values of the transformation. One $256 \times 8$-bit S-box is required for each byte of the State, and therefore 16 parallel S-boxes are required if SubBytes is performed for the entire State at once. Thus, the total number of S-boxes, without KeyExpansion, is 16 times the number of unrolled rounds, and a fully unrolled AES-128 (10 rounds) requires 160 S-boxes. KeyExpansion requires four additional S-boxes per round, and therefore a total number of 200 S-boxes are required in a fully unrolled key agile AES-128 implementation. If S-boxes are implemented on Xilinx FPGAs, 100 BlockRAMs are needed, because one dual-port BlockRAM can implement two S-boxes. BlockRAM-based S-boxes have been used in many publications, e.g. [13, 15, 17–23].

In [27], we presented a design called SIG-AES which implements SubBytes combinatorially as suggested in [28]. Because this approach requires mappings from $GF(2^8)$ to $GF(2^4)^2$ and vice versa, other transformations were also mapped to $GF((2^2)^2)$ in order to reduce area and latency. Hodjat and Verbauwhede explored the optimal number of pipelined stages in the combinatorially implemented SubBytes in [15], and they compared combinatorial implementation with implementation using BlockRAMs. Zhang and Parhi presented a careful analysis of the combinatorial implementation of SubBytes and introduced highly optimised implementations, one of which exceeds 20 Gbps on a Virtex-E FPGA [29]. The high efficiency was attained through detailed analysis and careful implementation using combinatorial SubBytes in $GF((2^2)^2)$. 

During the selection process Dandalis et al. [8], Elbirt et al. [9], Fischer [10], Gaj and Chodowiec [11] and Mroczkowski [12] published FPGA implementations on both Altera and Xilinx devices. Their studies concluded that Rijndael and Serpent had the highest throughputs [8, 9, 11], while Twofish and RC6 provided compact implementations with medium speed [11]. Mars clearly had the worst hardware characteristics [11].

Since the selection of Rijndael as AES, an enormous number of FPGA-based implementations have been published. Certain trends in these publications are considered next.
The largest benefit of the combinatorial implementation is that the SubBytes can be pipelined and thus higher throughput can be attained. This does, however, increase the latency of the implementation. The slice requirements also increase compared with BlockRAM-based implementations, because SubBytes is implemented with logic. In many applications, it is more important to minimise area than to maximise throughput. Therefore, several implementations with small logic requirements have been published. Pramstaller and Wolkerstorfer presented a compact implementation of AES encryption and decryption with all key lengths using a novel State representation, which solves the problem of accessing both rows and columns of the State [30]. A very compact implementation was presented by Chodowiec and Gaj in [31]. They efficiently exploited the structure of FPGA and were able to fit AES-128 encryption and decryption into 222 slices and three BlockRAMs on a low-cost Xilinx Spartan-II XC2S30-5/6. The design achieved a throughput of 166 Mbps on an XC2S30-6 [31]. At least to the authors’ knowledge, the most compact AES implementation published in the literature so far was presented by Rouvoy et al. in [32]. They were able to fit AES-128 with KeyExpansion into only 163 slices and three BlockRAMs on a Xilinx Spartan-3 XC3S50-4 and achieved a throughput of 208 Mbps. They also implemented the same design on a Virtex-II, and it has been included in the comparison below. The key to lower area consumption compared with [31] was the combination of SubBytes and MixColumns transformations [32].

The comparison of different AES implementations is hard for many reasons. First, the large variety of different devices makes a fair comparison difficult. Second, many authors do not specify their devices well enough to ensure easy comparison, e.g. the size or the speed grade of the device has not been provided. Third, comparison of area requirements is difficult because both slices and embedded memory, i.e. BlockRAMs in the Xilinx devices, are used.

Xilinx Virtex-family FPGAs (i.e. the Virtex [36], Virtex-E [37], Virtex-E Extended Memory (Virtex-EM) [26] and Virtex-II [38]) are clearly the most used implementation platforms for the published designs. Therefore, this comparison concentrates on designs implemented on these devices. Performances on different devices should not be compared, because the device greatly determines the performance of an implementation. Devices are therefore clearly differentiated in the tables and figures in this paper.

A summary of open-literature FPGA-based AES implementations on Xilinx Virtex-family devices is presented in Table 1, and it includes implementations published in [6, 8, 9, 11, 13–23, 25, 27, 29, 30, 32, 33, 39, 40]. In order to compare the area requirements of BlockRAM-based implementations with those of implementations which use only slices, a method introduced in [17] is used. Because a dual-port 256×8-bit BlockRAM can be replaced by distributed memory consisting of 256 LUTs, one BlockRAM can be replaced by 128 slices [17]. Thus, the area value in Table 1 was calculated using the following formula:

\[
\text{area} = \text{slices} + 128 \times \text{BlockRAMs}
\]  

The performance-area relationship is studied using two different metrics. The first one is the traditional throughput per slice (TPS) value [9]. The other, which also takes into account the BlockRAM utilisation, is called the throughput per area (TPA) value and is calculated using the area value obtained using (1). TPA offers a better impression of the performance-area relationship than TPS, which neglects the usage of BlockRAMs. An extreme example of this is the implementation presented in [25], which attains an extremely high TPS value but nonetheless requires a large target device because of the large number of BlockRAMs. However, in such extreme cases, the TPA method yields estimates that are too pessimistic and, therefore, TPA should be used together with TPS to ensure fair comparison.

Throughput-slice and throughput-area scatter plots are presented in Figs 1 and 2, respectively. There exists only little correlation between slice usage and throughput in Fig. 1. In Fig. 2, however, there is a significantly higher correlation, which again validates the use of TPA.

As stated earlier, the fastest reported AES implementation achieves a throughput of 23.57 Gbps with 16938 slices on a Virtex-II XC2V4000 [19]. Although it is the fastest implementation, it is not the most efficient of the high-throughput implementations, if TPS and TPA are considered as the efficiency metrics. As can be seen from Fig. 2, implementations by Hodjat and Verbauwhede [15] and Zhang and Parhi [29] achieve almost the same level of throughput with fewer logic resources. Considering the high TPS and TPA values as well as the slower Virtex-E device compared with the Virtex-II devices used in [15] and [19], Zhang’s design can be considered the most efficient fully unrolled and pipelined AES-128 implementation published so far.

Combinatorial implementation of SubBytes results in higher TPA values than LUT-based implementation. This is due to the fact that SubBytes can be pipelined and therefore very high throughput can be achieved. Combinatorial SubBytes also results in moderate area requirements. This can be seen in Fig. 2, where the combinatorial implementations, i.e. [15, 27, 29], are situated in the upper left corner.

If embedded memory can be used, a considerable reduction in slice requirements can be achieved by using BlockRAM-based S-boxes. Also the latency of S-boxes is shorter than that of the combinatorial SubBytes. The T-box approach seems infeasible if TPA is considered, but a very high TPS can be achieved. T-boxes are therefore very inviting if the slice requirement needs to be minimised in a high-throughput design. The approach presented in [25], however, requires a Virtex-EM XC8V812E FPGA because, with the exception of the new Virtex-4 FPGAs [41], no other device in the Xilinx Virtex family contains enough BlockRAMs (244) [26, 36–38].

Many implementations with an area in the range 1600–2000 have been published [17–19, 33, 40]. These compact implementations achieve relatively high throughputs of >1 Gbps on Virtex-E and Virtex-II FPGAs. The implementation by Standaert et al. [33] has the highest TPA and throughput among these implementations, as it achieves throughput of 2.085 Gbps with only 1769 slices and no BlockRAMs.
on a Virtex-E XCV3200E-8, and thus has a TPA (and TPS) of 1.179 Mbps/area. If even smaller area consumption is required, one must tolerate a large slowdown in throughput. The smallest implementations have throughputs measured in hundreds of Mbps; e.g. Rouvoy et al. achieved 358 Mbps with only 146 slices and three BlockRAMs (area 530) on a Virtex-II XC2V40-6 [32]. This survey has shown that various different methods have been presented to implement AES. It is impossible to point out the absolutely best method, because all methods have their advantages and disadvantages. As a conclusions to the AES study it is stated that AES can be efficiently implemented on FPGAs for applications with various requirements. Both very high performance and low area requirements can be efficiently achieved using the methods presented in the literature.

2.2 International Data Encryption Algorithm
IDEA was introduced by Lai and Massay in 1990 [42] and modified the following year [43]. IDEA is considered highly secure, and no published attack

<table>
<thead>
<tr>
<th>Authors</th>
<th>Key</th>
<th>Device</th>
<th>Slices</th>
<th>BRAM</th>
<th>Area</th>
<th>Throughput (Gbps)</th>
<th>TPS (Mbps/slice)</th>
<th>TPA (Mbps/area)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chodowiec and coworkers [13, 14]</td>
<td>Cho</td>
<td>Virtex 1000-6</td>
<td>12 600</td>
<td>80</td>
<td>22 940</td>
<td>12.16</td>
<td>0.965</td>
<td>0.532</td>
</tr>
<tr>
<td>Chodowiec et al. [13]</td>
<td>Cho</td>
<td>Virtex 1000-6</td>
<td>2 057</td>
<td>8</td>
<td>3 081</td>
<td>1.265</td>
<td>0.615</td>
<td>0.411</td>
</tr>
<tr>
<td>Chodowiec and coworkers [13, 14]</td>
<td>Cho</td>
<td>Virtex 1000-6</td>
<td>2 507</td>
<td>0</td>
<td>2 507</td>
<td>0.414</td>
<td>0.166</td>
<td>0.165</td>
</tr>
<tr>
<td>Dandalis et al. [8]</td>
<td>Dan</td>
<td>Virtex 6</td>
<td>5 673</td>
<td>0</td>
<td>5 673</td>
<td>0.353</td>
<td>0.062</td>
<td>0.062</td>
</tr>
<tr>
<td>Elbirt et al. [6, 9]</td>
<td>Elb</td>
<td>Virtex 1000-4</td>
<td>10 992</td>
<td>0</td>
<td>10 992</td>
<td>2.291</td>
<td>2.291</td>
<td>1.352</td>
</tr>
<tr>
<td>Elbirt et al. [6, 9]</td>
<td>Elb</td>
<td>Virtex 1000-4</td>
<td>1 871</td>
<td>0</td>
<td>1 871</td>
<td>0.949</td>
<td>0.195</td>
<td>0.195</td>
</tr>
<tr>
<td>Gaj and Chodowiec [11]</td>
<td>Gaj</td>
<td>Virtex 1000-6</td>
<td>2 902</td>
<td>0</td>
<td>2 902</td>
<td>0.332</td>
<td>0.114</td>
<td>0.114</td>
</tr>
<tr>
<td>Hodjat and Verbauwhede [15]</td>
<td>Hod</td>
<td>Virtex-II VP20-7</td>
<td>9 446</td>
<td>0</td>
<td>9 446</td>
<td>21.64</td>
<td>2.291</td>
<td>1.352</td>
</tr>
<tr>
<td>Holmström et al. [27]</td>
<td>Holm</td>
<td>Virtex-II 2000-5</td>
<td>10 750</td>
<td>0</td>
<td>10 750</td>
<td>17.8</td>
<td>1.656</td>
<td>1.656</td>
</tr>
<tr>
<td>Labbé and Pérez [39]</td>
<td>Labb</td>
<td>Virtex 1000-4</td>
<td>2 151</td>
<td>4</td>
<td>2 663</td>
<td>0.394</td>
<td>0.183</td>
<td>0.148</td>
</tr>
<tr>
<td>Labbé and Pérez [39]</td>
<td>Labb</td>
<td>Virtex 1000-4</td>
<td>3 543</td>
<td>4</td>
<td>4 055</td>
<td>0.796</td>
<td>0.225</td>
<td>0.196</td>
</tr>
<tr>
<td>Labbé and Pérez [39]</td>
<td>Labb</td>
<td>Virtex 1000-4</td>
<td>8 767</td>
<td>4</td>
<td>9 279</td>
<td>1.911</td>
<td>0.218</td>
<td>0.206</td>
</tr>
<tr>
<td>McLoone and McCanny [20]</td>
<td>ML1</td>
<td>Virtex-E 3200-8</td>
<td>2 222</td>
<td>100</td>
<td>15 022</td>
<td>6.956</td>
<td>3.131</td>
<td>0.463</td>
</tr>
<tr>
<td>McLoone and McCanny [25]</td>
<td>ML2</td>
<td>Virtex-EM 812-8</td>
<td>2 000</td>
<td>244</td>
<td>3 232</td>
<td>6.010</td>
<td>0.362</td>
<td>0.528</td>
</tr>
<tr>
<td>McLoone and McCanny [21]</td>
<td>ML3</td>
<td>Virtex-II 3200-8</td>
<td>1 125</td>
<td>0</td>
<td>1 125</td>
<td>0.215</td>
<td>0.191</td>
<td>0.191</td>
</tr>
<tr>
<td>Willett and Walkerstrofer [30]</td>
<td>W1</td>
<td>Virtex E 1000-8</td>
<td>5 677</td>
<td>80</td>
<td>15 917</td>
<td>4.121</td>
<td>0.726</td>
<td>0.571</td>
</tr>
<tr>
<td>Rodriguez-H et al. [16]</td>
<td>Rod</td>
<td>Virtex E 2600</td>
<td>146</td>
<td>3</td>
<td>530</td>
<td>2.452</td>
<td>0.675</td>
<td>1.091</td>
</tr>
<tr>
<td>Rouvoy et al. [32]</td>
<td>Rouv</td>
<td>Virtex-II 40-6</td>
<td>2778</td>
<td>100</td>
<td>15 678</td>
<td>8.9</td>
<td>3.204</td>
<td>0.571</td>
</tr>
<tr>
<td>Saggese et al. [17]</td>
<td>Sagg</td>
<td>Virtex-E 1000-4</td>
<td>446</td>
<td>10</td>
<td>1 726</td>
<td>2.342</td>
<td>0.579</td>
<td>0.579</td>
</tr>
<tr>
<td>Saggese et al. [17]</td>
<td>Sagg</td>
<td>Virtex-E 1000-4</td>
<td>5 810</td>
<td>100</td>
<td>18 610</td>
<td>20.3</td>
<td>3.494</td>
<td>1.091</td>
</tr>
<tr>
<td>Saggese et al. [17]</td>
<td>Sagg</td>
<td>Virtex-E 1000-4</td>
<td>648</td>
<td>10</td>
<td>1 828</td>
<td>2.809</td>
<td>0.944</td>
<td>0.944</td>
</tr>
<tr>
<td>Saqib et al. [22]</td>
<td>Saqi</td>
<td>Virtex-EM 812-8</td>
<td>2 744</td>
<td>0</td>
<td>2 744</td>
<td>0.259</td>
<td>0.094</td>
<td>0.094</td>
</tr>
<tr>
<td>Saqib et al. [23]</td>
<td>Saqi</td>
<td>Virtex-EM 812-8</td>
<td>2 136</td>
<td>100</td>
<td>14 936</td>
<td>2.896</td>
<td>1.343</td>
<td>0.192</td>
</tr>
<tr>
<td>Siqueira et al. [18]</td>
<td>St1</td>
<td>Virtex 1000-6</td>
<td>2 257</td>
<td>0</td>
<td>2 257</td>
<td>1.563</td>
<td>0.893</td>
<td>0.693</td>
</tr>
<tr>
<td>Siqueira et al. [18]</td>
<td>St1</td>
<td>Virtex-E 3200-8</td>
<td>2 784</td>
<td>100</td>
<td>15 584</td>
<td>11.776</td>
<td>4.230</td>
<td>0.756</td>
</tr>
<tr>
<td>Siqueira et al. [18]</td>
<td>St1</td>
<td>Virtex-E 3200-8</td>
<td>542</td>
<td>10</td>
<td>1 822</td>
<td>2.675</td>
<td>0.796</td>
<td>0.796</td>
</tr>
<tr>
<td>Siqueira et al. [33]</td>
<td>St2</td>
<td>Virtex-EM 812-8</td>
<td>1 769</td>
<td>0</td>
<td>1 769</td>
<td>2.085</td>
<td>1.179</td>
<td>1.179</td>
</tr>
<tr>
<td>Siqueira et al. [33]</td>
<td>St2</td>
<td>Virtex-E 3200-8</td>
<td>15 112</td>
<td>0</td>
<td>15 112</td>
<td>18.560</td>
<td>2.228</td>
<td>1.228</td>
</tr>
<tr>
<td>Wang and Ni [40]</td>
<td>Wan</td>
<td>Virtex E 1000-8</td>
<td>1 857</td>
<td>0</td>
<td>1 857</td>
<td>1.604</td>
<td>0.864</td>
<td>0.864</td>
</tr>
<tr>
<td>Weaver and Wawrzyniec [23]</td>
<td>Weav</td>
<td>Virtex 600-8</td>
<td>2778</td>
<td>100</td>
<td>15 678</td>
<td>8.9</td>
<td>3.204</td>
<td>0.571</td>
</tr>
<tr>
<td>Zhang and Parhi [28]</td>
<td>Zha</td>
<td>Virtex 1000-6</td>
<td>11 014</td>
<td>0</td>
<td>11 014</td>
<td>16.032</td>
<td>1.456</td>
<td>1.456</td>
</tr>
<tr>
<td>Zhang and Parhi [28]</td>
<td>Zha</td>
<td>Virtex-EM 812-8</td>
<td>2 744</td>
<td>0</td>
<td>2 744</td>
<td>0.259</td>
<td>0.094</td>
<td>0.094</td>
</tr>
</tbody>
</table>

The authors have selected the most relevant implementations (in their opinion) from those publications which include several different implementations. Keys are used in Figs 1 and 2.
Fig. 1  Throughput–slice chart of FPGA-based AES implementations

Fig. 2  Throughput–area chart of FPGA-based AES implementations

IEE Proc. Inf. Secur.
IDEA encrypts 64-bit plaintext blocks into 64-bit ciphertext blocks using a 128-bit input key $K$. The algorithm consists of eight identical rounds followed by an output transformation. Each round uses six 16-bit subkeys $K^{(i)}$, $1 \leq i \leq 6$, to transform a 64-bit input $X$ into an output of four 16-bit blocks, which are then input to the next round. All subkeys are derived from the 128-bit input key $K$. The subkey derivation process is different in decryption mode from the encryption mode, but otherwise encryption and decryption are performed using identical hardware.

IDEA uses only three operations on 16-bit sub-blocks $a$ and $b$: bitwise XOR, unsigned addition mod $(2^{16})$ and modulo $(2^{16} + 1)$ multiplication. All three operations are derived from different algebraic groups of $2^{16}$ elements, which is crucial to the algorithmic strength of IDEA. Of the three arithmetic operations, bitwise XOR and unsigned addition mod $(2^{16})$ are trivial to implement, whereas an implementation of modulo $(2^{16} + 1)$ multiplication that is both area efficient and fast requires careful design and bit-level optimisation.

Two early FPGA-based IDEA implementations were published by Mencer et al. [45] and Mosanya et al. [46] in 1998 and 1999, respectively. Mencer et al. studied the benefits and limitations of FPGA systems compared with processors and application-specific integrated circuits (ASICs) using IDEA encryption as a benchmark. Their IDEA implementation had a throughput of 528 Mbps, and it covered four Xilinx XC4020 FPGAs (3200 CLBs) [45]. Mosanya et al. presented a reconfigurable cryptoprocessor called CryptoBooster in [46]. They did not present any exact performance figures for their implementation but they estimated that throughputs of 200–1500 Mbps could be achieved on a state-of-the-art FPGA of that time [46].

Representative compact and high-speed FPGA-based implementations of IDEA include a bit-serial implementation described by Leong et al. in [47], with a throughput of 500 Mbps on a Xilinx Virtex XC3000-6, and the bit-parallel implementation described by Cheung et al. in [48], which achieved throughput of 5.25 Gbps on a Virtex XC10000-6. The bit-parallel implementation also included bespoke software to customise the FPGA reprogramming bitstream for different key schedules.

In 2002, the SIG-IDEA implementation was described in [49], and, at 6.78 Gbps, its throughput represented the fastest published FPGA-based implementation of IDEA at that time. Other contributions of SIG-IDEA include implementing a fully pipelined algorithm with both inner and outer loop pipelining on a single Xilinx Virtex-E XCV1000E-6 device, the efficient usage of the diminished-one number system and an area-efficient implementation of the modulo $(2^{16})$ multiplication.

Currently, the fastest FPGA-based implementation of IDEA is probably [50], where Gonzalez et al. achieved a throughput of 8.3 Gbps on a Xilinx Virtex XC6000-6 device. The key to high throughput was replacing all the operational units involving the key with its constant-operand equivalents by partial reconfiguration, the overhead for which was 4 ms. However, only a few devices support partial reconfiguration, and the scheme requires a controlling microprocessor. Another recent IDEA implementation by Pan et al. achieved a throughput of 6 Gbps by utilising the embedded multipliers for the modulo $(2^{16} + 1)$ multiplication algorithm [51].

The FPGA-based IDEA implementations mentioned above are summarised in Table 2.

### 3 Hash algorithms

Commonly used hash algorithms, e.g., MD5 [52] and the secure hash algorithm (SHA) [53], are not as well suited to high-speed hardware implementations as most of the private-key or public-key algorithms, mainly because parallelisation cannot be used as efficiently. Hash algorithms can be implemented efficiently on software as they use common modulo $(2^{32})$ additions, which are easy and fast to perform with traditional microprocessors. However, significant accelerations from 25 to 31 times for SHA-1 and SHA-512 have been reported [54].

There are certain applications which greatly benefit from hardware acceleration. For example, if a cryptographic scheme requiring hash calculations, e.g. the digital signature algorithm [55], is implemented on an FPGA, it is well-grounded to implement a hash module on the chip too. Certain very demanding hash calculations, e.g. long chains of hash rounds, benefit from hardware acceleration [56]. Hash algorithm implementations presented in [54, 56–69] are considered here.

A throughput of several hundred Mbps can be achieved with small logic requirements using a basic iterative architecture [54, 56, 57, 63, 65, 67, 69]. However, the throughput and efficiency of an implementation can be increased considerably by partially unrolling the algorithm rounds [54, 58, 66]. This is because the structure of hash algorithms favours

Table 2: IDEA implementations on Xilinx FPGAs

<table>
<thead>
<tr>
<th>Authors</th>
<th>Device</th>
<th>Slices</th>
<th>Throughput (Gbps)</th>
<th>TPS (Mbps/slice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheung et al. [48]</td>
<td>Virtex 1000-6</td>
<td>11 602</td>
<td>5.24</td>
<td>0.452</td>
</tr>
<tr>
<td>Gonzalez et al. [50]</td>
<td>Virtex 600-6</td>
<td>6 078</td>
<td>8.3</td>
<td>1.366</td>
</tr>
<tr>
<td>Hämäläinen et al. [49]</td>
<td>Virtex-E 1000-6</td>
<td>9 855*</td>
<td>6.78</td>
<td>0.688</td>
</tr>
<tr>
<td>Leong et al. [47]</td>
<td>Virtex 300-6</td>
<td>2 801</td>
<td>0.5</td>
<td>0.179</td>
</tr>
<tr>
<td>Mencer et al. [46]</td>
<td>XC4000</td>
<td>n.a. (3 200 CLBs)</td>
<td>0.528</td>
<td>n.a.</td>
</tr>
<tr>
<td>Pan et al. [51]</td>
<td>Virtex-II 1000-6</td>
<td>4 221</td>
<td>6.0</td>
<td>1.421</td>
</tr>
</tbody>
</table>

* The value is not available in the original publication. Received from the design files.
unrolling; i.e. if $k$ rounds are unrolled, the critical path increases for fewer than $k$ times [54]. Pipelining, however, cannot be used for increasing throughput as efficiently as for block ciphers [56]. Unrolling was used in the fastest published SHA-1 implementations, whereas Lien et al. reported a throughput of 1024 Mbps on a Virtex XC1000-6 [54] and Sklavos et al. achieved a throughput of 1339 Mbps on a Virtex-II XC2V500 with their combined SHA-1 and RIPEMD design [66].

Because many of the commonly used hash algorithms share resources and have a similar kind of structure, many implementations combining several hash algorithms have been proposed. Dominikus presented a general hash processor architecture which can be used for MD5, SHA-1, SHA-256 and RIPEMD calculations in [59]. A general processor architecture naturally achieves slower performance than algorithm-specific implementations, such as [54, 56–58, 67], but algorithm flexibility may be an essential feature in certain applications. A sufficient balance between speed and algorithm flexibility may be achieved by implementing an algorithm-specific design of two or more commonly used algorithms. Implementations combining certain algorithms have been published; e.g. MD5 and SHA-1 were combined in [61, 62, 68], different SHA algorithms were combined in [63] and MD5 was combined with RIPEMD in [64]. Algorithm support of published FPGA-based implementations is presented in Table 3.

The performance and area requirements of FPGA-based open-literature hash algorithm implementations are presented in Table 4. Hash algorithm implementations achieving throughputs of several hundred Mbps require only a minimal amount of logic resources. Hash algorithms with different cryptographic strengths, e.g. SHA-1 and SHA-512, have almost similar throughputs. Hash algorithms require considerably more area than implementations of block ciphers, e.g. the AES, of the same speed [56]. However, very high-speed implementations of hash algorithms require considerably more area than implementations of block ciphers, e.g. the AES, of the same speed [56]. This can be verified, for example, by comparing the SIG-MD5 implementation of four parallel MD5 blocks [56] with the AES implementation by Standaert et al. [33]. Both have a similar level of throughput (2395 and 2085 Mbps), but the MD5 implementation consumes a lot more area (5732 slices) than the AES implementation (1769 slices).

MD5, RIPEMD and SHA-1 were recently compromised so that finding collisions is possible with much less effort than exhaustive searching [70, 71]; therefore these algorithms can no longer be considered secure. Although finding collisions is not a problem in every application using hash algorithms, e.g. HMAC (the keyed-hash message authentication code), these algorithms will certainly be replaced with stronger ones in the future. Hardware implementation of hash algorithms will probably be studied actively when MD5 and SHA-1 are replaced with new algorithms.

### 4 Conclusions and future work

We have shown that FPGAs can be used very efficiently for high-speed implementations of cryptographic algorithms. The field has been studied extensively for the past few years and very efficient implementations have been presented regardless of the implemented algorithm. Similar design methodologies apply to all algorithms studied in this survey. The key to a high-speed implementation is to identify the critical operation, e.g. SubBytes in the AES, and implement it efficiently. In general, operations, or at least the critical operation,

---

**Table 3: Algorithm support of published FPGA-based implementations of hash algorithms**

<table>
<thead>
<tr>
<th>Authors</th>
<th>MD5</th>
<th>SHA-1</th>
<th>SHA-256</th>
<th>SHA-384</th>
<th>SHA-512</th>
<th>RIPEMD</th>
<th>HAS-160</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deepakumara et al. [57]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diez et al. [58]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diez et al. [58]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominikus [59]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grembowski et al. [60]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grembowski et al. [60]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Järvinen et al. [56]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Järvinen et al. [61]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kang et al. [62]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>McLoone and McCanny [63]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ng et al. [64]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selinsis et al. [65]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sklavos et al. [66]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ting et al. [67]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al. [68]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zibin and Ning [69]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*IEE Proc. Inf. Secur.*
should be implemented on as low a level as possible in order to guarantee maximum performance with minimum resources.

Although all the designs considered here were implemented on FPGAs, only a small number of them specifically target FPGAs as they are merely general hardware implementations which could be implemented on ASICs as well. Exploiting the special properties of FPGAs has not yet been thoroughly studied, with the exception of embedded memory usage. However, partial reconfigurability was used in an IDEA implementation [50] as discussed in Section 2.2. Certain implementations which have been optimised especially for the slice structure have been published, e.g. [18].

Many cryptographic algorithms use similar kinds of operations, so it may be possible to combine several algorithms into a single design efficiently by exploiting these similarities. Such combinations of cryptographic algorithms have not been studied extensively, except in the case of hash algorithms.

There exist dedicated embedded blocks for certain commonly used operations in modern FPGAs, e.g. the Altera Stratix-II architecture includes dedicated blocks for digital signal processing (DSP) [73]. Dedicated blocks for cryptography do not yet exist on any device, but if such blocks were implemented they could speed up the performance of cryptographic algorithms substantially. The question of how these blocks should be arranged and which operations should be implemented is an open research problem.

Based on our observations, certain possible research topics in the future include

- architectures for constrained environments
- an increase of generality, general cryptographic architectures implementing several cryptographic algorithms in a single design (cf. MD5/SHA-1 implementations)
- dedicated blocks for cryptographic operations into FPGAs (cf. DSP blocks)
- efficient utilisation of the special abilities of FPGAs (e.g. partial reconfiguration)
- efficient implementations of strong hash algorithms.

It was concluded that regardless of the algorithm, very efficient implementations have been published in terms of both speed and logic requirements. Although cryptographic algorithm implementation has been widely studied, certain open problems remain.

Table 4: Performance and area requirements of published FPGA-based implementations of hash algorithms

<table>
<thead>
<tr>
<th>Authors</th>
<th>Device</th>
<th>Algorithm</th>
<th>Slices</th>
<th>BlockRAMs</th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deepakumara et al. [57]</td>
<td>Virtex 1000-6</td>
<td>MD5</td>
<td>880</td>
<td>2</td>
<td>185</td>
</tr>
<tr>
<td>Deepakumara et al. [57]</td>
<td>Virtex 1000-6</td>
<td>MD5</td>
<td>4763</td>
<td>0</td>
<td>394</td>
</tr>
<tr>
<td>Diez et al. [58]</td>
<td>Virtex-II 3000</td>
<td>MD5</td>
<td>1389</td>
<td>0</td>
<td>467.3</td>
</tr>
<tr>
<td>Diez et al. [58]</td>
<td>Virtex-II 3000</td>
<td>SHA-1</td>
<td>1550</td>
<td>0</td>
<td>899.8</td>
</tr>
<tr>
<td>Dominikus [59]</td>
<td>Virtex-E 300</td>
<td>MD5</td>
<td>1004</td>
<td>0</td>
<td>146</td>
</tr>
<tr>
<td>Dominikus [59]</td>
<td>Virtex-E 300</td>
<td>RIPEMD</td>
<td>1004</td>
<td>0</td>
<td>89</td>
</tr>
<tr>
<td>Dominikus [59]</td>
<td>Virtex-E 300</td>
<td>SHA-1</td>
<td>1004</td>
<td>0</td>
<td>119</td>
</tr>
<tr>
<td>Dominikus [59]</td>
<td>Virtex-E 300</td>
<td>SHA-256</td>
<td>1004</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>Grembowski et al. [60]</td>
<td>Virtex 1000-6</td>
<td>SHA-1</td>
<td>1475</td>
<td>0</td>
<td>462</td>
</tr>
<tr>
<td>Grembowski et al. [60]</td>
<td>Virtex 1000-6</td>
<td>SHA-512</td>
<td>2826</td>
<td>2</td>
<td>616</td>
</tr>
<tr>
<td>Jarvinen et al. [56]</td>
<td>Virtex-II 4000-6</td>
<td>MD5</td>
<td>1325</td>
<td>0</td>
<td>607</td>
</tr>
<tr>
<td>Jarvinen et al. [56]</td>
<td>Virtex-II 4000-6</td>
<td>MD5</td>
<td>5732</td>
<td>0</td>
<td>2395</td>
</tr>
<tr>
<td>Jarvinen et al. [56]</td>
<td>Virtex-II 4000-6</td>
<td>MD5</td>
<td>11498</td>
<td>10</td>
<td>5857</td>
</tr>
<tr>
<td>Jarvinen et al. [61]</td>
<td>Virtex-II 2000-6</td>
<td>MD5</td>
<td>1882</td>
<td>0</td>
<td>602</td>
</tr>
<tr>
<td>Jarvinen et al. [61]</td>
<td>Virtex-II 2000-6</td>
<td>SHA-1</td>
<td>1882</td>
<td>0</td>
<td>485</td>
</tr>
<tr>
<td>Kang et al. [62]</td>
<td>Apex 20K 1000-3</td>
<td>MD5</td>
<td>10573</td>
<td>0</td>
<td>142</td>
</tr>
<tr>
<td>Kang et al. [62]</td>
<td>Apex 20K 1000-3</td>
<td>SHA-1</td>
<td>10573</td>
<td>0</td>
<td>114</td>
</tr>
<tr>
<td>Kang et al. [62]</td>
<td>Apex 20K 1000-3</td>
<td>HAS-160</td>
<td>10573</td>
<td>0</td>
<td>160</td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>Virtex 1000-6</td>
<td>SHA-1</td>
<td>480</td>
<td>0</td>
<td>544</td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>Virtex 1000-6</td>
<td>SHA-1</td>
<td>1480</td>
<td>0</td>
<td>1024</td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>Virtex 1000-6</td>
<td>SHA-512</td>
<td>2384</td>
<td>0</td>
<td>717</td>
</tr>
<tr>
<td>Lien et al. [54]</td>
<td>Virtex 1000-6</td>
<td>SHA-512</td>
<td>3521</td>
<td>0</td>
<td>929</td>
</tr>
<tr>
<td>McLoone and McCanny [63]</td>
<td>Virtex-E 600-8</td>
<td>SHA-384/512</td>
<td>2914</td>
<td>2</td>
<td>479</td>
</tr>
<tr>
<td>Ng et al. [64]</td>
<td>Flex 50-1</td>
<td>MD5</td>
<td>1964</td>
<td>0</td>
<td>206</td>
</tr>
<tr>
<td>Ng et al. [64]</td>
<td>Flex 50-1</td>
<td>RIPEMD</td>
<td>1964</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>Selmis et al. [65]</td>
<td>Virtex 150</td>
<td>SHA-1</td>
<td>518</td>
<td>0</td>
<td>518</td>
</tr>
<tr>
<td>Sklavos et al. [66]</td>
<td>Virtex-II 500</td>
<td>SHA-1</td>
<td>2245</td>
<td>0</td>
<td>1339</td>
</tr>
<tr>
<td>Sklavos et al. [66]</td>
<td>Virtex-II 500</td>
<td>RIPEMD</td>
<td>2245</td>
<td>0</td>
<td>1656</td>
</tr>
<tr>
<td>Ting et al. [67]</td>
<td>Virtex-E 300-8</td>
<td>SHA-256</td>
<td>1261</td>
<td>0</td>
<td>693</td>
</tr>
<tr>
<td>Wang et al. [68]</td>
<td>Apex 20K 1000-3</td>
<td>MD5</td>
<td>3040</td>
<td>1</td>
<td>178.6</td>
</tr>
<tr>
<td>Wang et al. [68]</td>
<td>Apex 20K 1000-3</td>
<td>SHA-1</td>
<td>3040</td>
<td>1</td>
<td>143.3</td>
</tr>
<tr>
<td>Zibin and Ning [69]</td>
<td>Acex 100-1</td>
<td>SHA-1</td>
<td>1622</td>
<td>0</td>
<td>266.99</td>
</tr>
</tbody>
</table>

Notice that 1 slice ≈ 2 logic elements (LEs), 1 BlockRAM = 4096 bits [36, 37, 26] and 1 embedded system block (ESB) = 2048 bits [72].

a Calculated from the percentages presented in the paper.
5 Acknowledgments
This paper was written as part of the GO-SEC project at Helsinki University of Technology. GO-SEC is financed by the National Technology Agency of Finland and several Finnish telecommunications companies. Finally, we would like to thank the anonymous reviewers for their helpful comments and proposals for improvement.

6 References


