BRAM-LUT tradeoff on a Polymorphic DES Design

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

[1] Instituto Superior Técnico/INESC-ID. Rua Alves Redol 9, 1000-029 Lisbon, Portugal. http://sips.inesc-id.pt/
[2] Computer Engineering Lab, TUDelft. Postbus 5031, 2600 GA Delft, The Netherlands. http://ce.et.tudelft.nl/

Abstract. A polymorphic implementation of the DES algorithm is presented. The polymorphic approach allows for a very fast integration of the DES hardware in existing software implementations, significantly reducing the time to marked and the development costs associated with hardware integration. The tradeoff between implementing the DES SBOXs in LUT or in BRAMs is the focus of the study presented in this paper. The FPGA implementation results suggest LUT reduction in the order of 100 slices (approximately 37%) for the full DES core, at the expense of 4 embedded memory blocks (BRAM). Even with this delay increase, the usage of BRAMs allows for an improvement of the Throughput per Slice ratio of 20%. The proposed computational structure has been implemented on a Xilinx VIRTEX II Pro (XC2VP30) prototyping device, requiring approximately 2% of the device resources. Experimental results, at an operating frequency of 100 MHz, suggest for the proposed polymorphic implementation a throughput of 400 Mbit/s for DES and 133 for 3DES. When compared with the software implementation of the DES algorithm, a speed up of 200 times can be archived for the kernel computation.

1 Introduction

In present days, most of the communication systems requires secure data transfer in order to maintain the privacy of the transmitted message; this message can be a simple email or a billion euro transaction between banks. In order to maintain the security of the communication channels, several encryption standards and algorithms exist, such as public key ciphers, symmetric ciphers and hash functions. For ciphering the bulk of data, symmetrical ciphering algorithms are used. Even though new emerging algorithms for symmetrical encryption have been appearing, the Data Encryption Standard (DES) [1] is still widely used, especially in banking application and monetary transactions, due to backward compatibility and legacy issues. In 1998 [2] the DES algorithm and its 54 bit key, have been deemed unsafe and replace by 3DES, which basically consists in performing the DES computation three times with three different keys, having a 112 bits equivalent key. With the increase of embedded application requiring DES (and 3DES), like RFID and bank cards, efficient hardware implementations of DES are demanded. In this paper a polymorphic implementation of the DES algorithm is proposed. This approach allows the hardware implemented DES core to be invoked in the same manner as has the equivalent software function, making its usage transparent to the software developer. This allows for a lower development cost and a much faster time to market. This paper also studies the advantages and disadvantages of using embedded memories for the implementation of the DES S-BOXs.

The FPGA implementation results suggest that a significant LUT reduction in the order of 100 slices (approximately 37%) for the full DES core, at the expense of 4 embedded memory blocks (BRAM). Even with this delay increase, the usage of BRAMs allows for an improvement of the Throughput per Slice ratio of 20%.

Experimental results for the polymorphic implementation, obtained from a prototype developed using a Xilinx VIRTEX II pro 30 prototyping FPGA, suggest:

– Speedups up to 200 times compared to the pure software implementations;
– Minimal software integration costs;
– Throughput of 400 Mbit/s for DES and 133 Mbits for 3DES, with 2% device usage.

The paper is organized as follows: Section 2 presents an overview on the DES algorithm. The implemented hardware structure is presented in section 3. Section 4 describes the proposed polymorphic DES organization and its usage in existing applications. Section 5 presents the obtained experimental results and compares them to related DES state-of-the-art. Section 6 concludes this paper with some final remarks.

2 DES Computation

Nowadays, the field of cryptography is growing up very intensively and many others algorithms are presented to meet the requirements of modern electronic systems. Since the time when the DES algorithm was introduce (in 1976), there are many devices and systems in which this algorithm is the bases into their security level. The high performance solutions are based on ASIC technologies and the reconfigurable ones are based on FPGA technologies. In both of the cases for each new solution is necessary to keep the compatibility with devices which are already available on the market. In our paper, an implementation of DES algorithm as a part of dynamic reconfigurable system based on FPGA technology is presented.

In DES, 64 bit data blocks are encrypted using a 54 bit Key (obtained from an input key with 64 bits). The intermediate ciphered values are processed as two 32-bit words (L_i and R_i), which are manipulated in 16 identical rounds as depicted in Figure 1. This manipulation consists of substitution, permutation, and bitwise XOR operation, over the 64-bit data block. The DES algorithm also has an Initial bit Permutation (IP) at the beginning of a block ciphering. To conclude the ciphering of a block, a final permutation is performed, which corresponds to the inverse of the initial permutation (IP^{-1}). The main computation is performed in 16 round designated by Feistel network, named after cryptographer Horst Feistel. In each round a different sub-key is used, generated form the main key expansion. The round computation or Feistel network is depicted in Figure 2.

The Feistel network is composed be the 3 main operation in symmetrical ciphering, namely key addition, confusion, and diffusion [3]. The first half of the round block is expanded from 32 to 48 bits and added to the 48-bits of the current sub-key. While the data expansion can be hardwired in the computation logic, the key addition requires

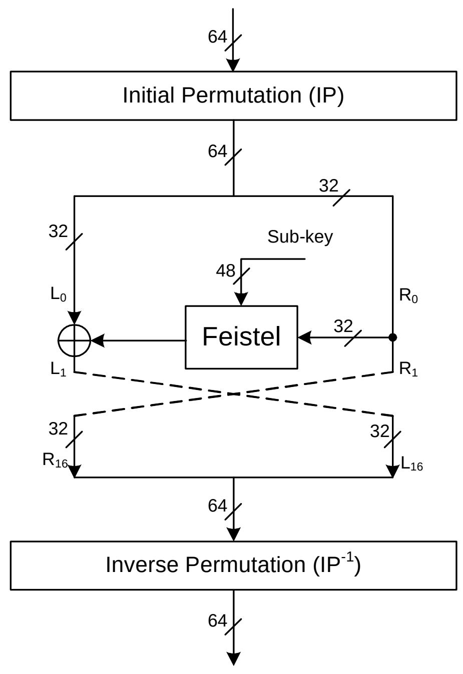

Fig. 1: DES computation.

XOR gates for the computation. The Key addition operation is followed by the confusion operation, performed by SBOXs. In this operation the value resulting from the addition is grouped in 8 blocks of 6 bits each. Each 6 bits are replaced by a different set of 8 groups of 4 bits, resulting in 32 different bits. The diffusion operation is performed by a final permutation. After the 16 rounds have been computed, a final permutation (IP^{-1}) is performed over the 64 bit data block.

The DES computational structure has the advantage that the decryption computation is identical to the encryption computation, only requiring the reversal of the key schedule.

3 Proposed DES structure

As depicted in Figures 1 and 2 the core computation of DES can be summed up to XOR operations, the SBOXs, permutations and word expansions. Since the permutations and expansions can be performed by routing, only the XORs, SBOXs, and some glue logic require computational logic. In order to create a compact DES computational core, a fully folded design has been implemented. In each clock cycle one round of the DES 16 rounds are computed, thus 16 clock cycles are required to compute a 64-bit data block. The used structure is presented in Figure 3. In this folded design some additional logic is required for multiplexing and additional round control.

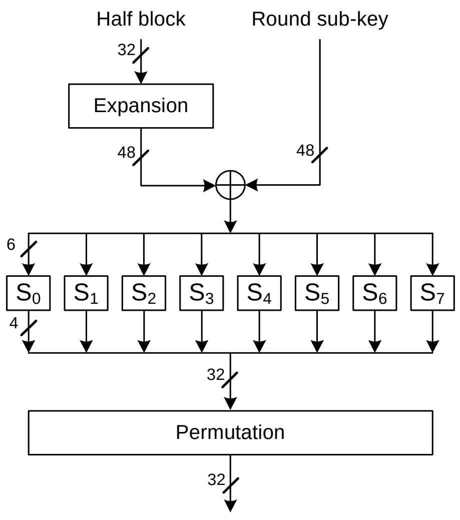

Fig. 2: DES Feistel network.

Given that, this DES core is to be used on a FPGA device, two major computational structures can chosen for the implementation of the SBOXs. The first and most commonly used is the implementation of the SBOX using the FPGA Look Up Tables (LUT). In this approach distributed memory blocks are created for each of the 32 bits of the word resulting from the SBOXs. Since most of the used Xilinx FPGAs have 4 input LUTs, the 6 bit SBOX requires at least 2 LUTs for each output bit. From this, it can be estimated that at least 64 LUT are required having a critical path of at least 2 LUTs, as depicted in Figure 4.

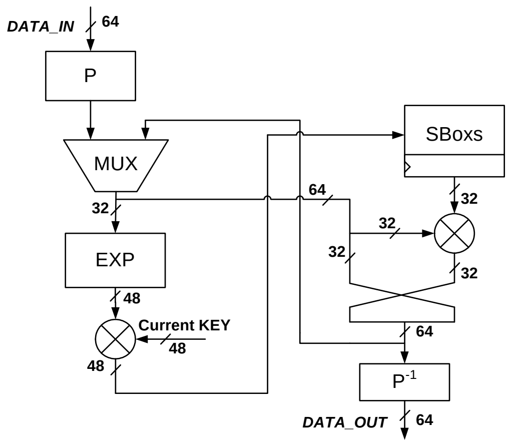

Fig. 3: DES computational structure.

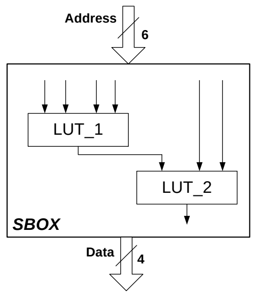

Fig. 4: LUT based SBOXs.

Taking into account that current FPGAs have embedded memory blocks (BRAMs), an alternative implementation of the SBOXs can be used. These BRAMs can be used as ROM blocks, to implement a full SBOX table. Since these BRAMs have output ports with at leat 4 bits, one BRAM can be used to replace at leat $2 \times 4 = 8$ LUTs. Moreover, modern FPGAs have embedded dual port BRAMs with more that $(2 \times 2^6 =)$ 128 words, thus, two SBOXs can be computed in each BRAM, as depicted in Figure 5. With this,

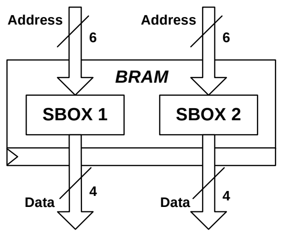

Fig. 5: BRAM based SBOXs.

only 4 BRAMs need to be used, instead of at least 64 LUTs. Due to the fact that existing BRAMs have registered output ports the round register must be located at the end of the SBOXs, limiting the options of the designer where to place the round registers.

In the DES algorithm the encryption and decryption of data differs only in the order in which the key expansion is performed. The key expansion consists of fixed permutations and rotate operations. While the permutation operations can be performed by routing, the rotation requires dedicated hardware. The rotation can be of 1 or 2 positions and, depending on the operation (encryption or decryption), to the left or to the right. The implemented structure is depicted in Figure 6.

In order to simplify the computational structure and the key expansion, only the DES algorithm is performed in hardware. To compute the 3DES algorithm, the DES hardware is called 3 times with the 3 different keys, thus performing the 3DES calculation.

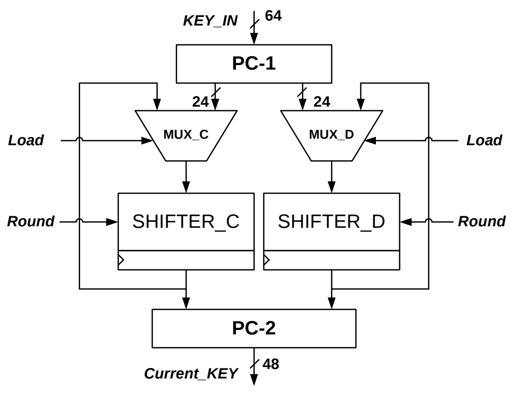

Fig. 6: DES key expansion.

4 Polymorphic Implementation

In order to efficiently use the DES core with a low development cost to the programmer, the MOLEN [4,5] computational paradigm is used. The MOLEN paradigm is based on the coprocessor architectural paradigm, allowing the usage of reconfigurable custom designed hardware units. In this computational approach, the non critical part of the software code is executed on a General Purpose Processor (GPP), while the main DES algorithm, is executed on the Custom Computing Unit (CCU). The DES core is seen by the programmer in the same manner as a software implemented function. The decision where the function is executed is made at compile time. At microarchitectural level the arbiter, depicted in Figure 7, redirects each instruction either to the GPP (a PowerPC in our case) or to the cryptographic units.

In a software function, the parameter passing is done through the stack. In the Molen processor, when a hardware function is invoked the parameters are passed through a dedicated register bank, designated by eXchange REGisters (XREG).

Given that the dedicated computational units are also connected to the main data memory, only initialization parameters are passed to the DES computational unit via the XREG. These parameter are the private key, memory pointers to the data to be ciphered, and the operation modes, e.g. encrypt or decrypt. The data to be processed is directly retrieved and send to the main data memory, via a shared memory mechanism.

In order to illustrate the data flow, the encryption operation for a 64 bit data block is described. When the DES cipher function is called, a few software instructions are executed, namely instructions that move the function parameters from the GPP internal registers to the XREG, followed by an execute instruction. When an execute instruction is detected by the arbiter, the later starts addressing the microcode memory, giving control of the data memory to the DES core, and signals it to start the computation via the $start$ signal depicted in Figure 7.

Once the DES core receives the $start$ signal, it starts retrieving the values from the XREG. The first value read is the operation mode, which indicates which operation will be performed. Continuously, the start and end memory addresses for the data to

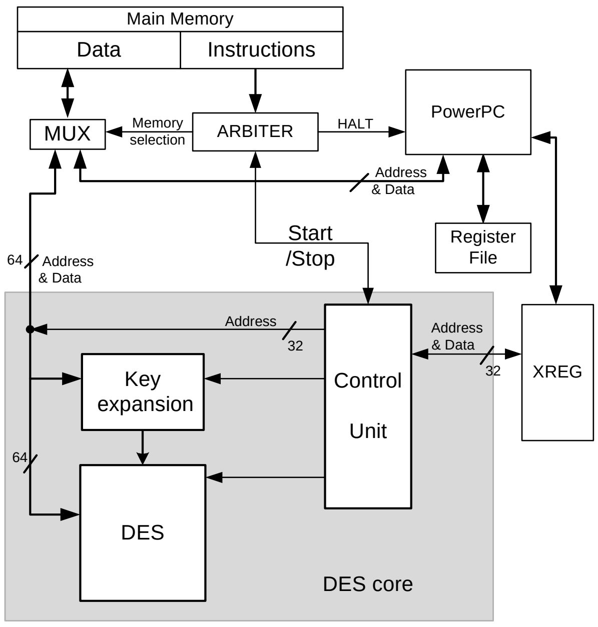

Fig. 7: Polymorphic processor internal structure.

cipher are retrieved from the XREG. While the first data block is read from the memory, the key is read from the XREG and stored in the DES internal registers. After this initialization phase, the DES core enters a loop where, while the data is being ciphered, the next 64-bit data block is read from the memory. In the end of each loop, the ciphered data is written back into the data memory. When the current memory address coincides with the data end address, the computation loop is broken and the *stop* signal is sent to the arbiter. Upon receiving this stop signal, the arbiter returns the memory control to the GPP.

To indicate which function performs the DES encryption computed in hardware, a pragma annotation is used in the C code, as depicted in Figure 8.

```
#pragma DES
DES (key, &data[0], &data[end], mode){
   \* implemented in HW *\
}
```

Fig. 8: Usage of the pragma notation.

This pragma annotation is recognized by the compiler which automatically generates the required instructions sequence [4]. This pragma addition and recompilation are the only operation required to use the hardware implemented DES, instead of the software implemented version. With this mechanism, any application using DES or 3DES can be accelerated by the DES core, with a reduced time market and a very low development cost.

5 Performance Analysis

Table 1: Stand-alone DES performances

	Our-BRAM	Our-LUT	Wee [6]	Rouv [7]	Our-BRAM	Our-LUT	CAST [8]	Our-BRAM	Our-LUT
Device	V1000E	V1000E	V2-4	V2-5	V2-5	V2-5	V2P2-7	V2P30-7	V2P30-7
Freq. (MHz)	81	138	179	274	152	202	261	218	287
Slices	174	277	382	**189**	**175**	**278**	**255**	**175**	**278**
BRAMs	4	0	0	0	4	0	0	4	0
Thrput (Mb/s)	354	551	716	974	609	808	1044	872	1148
Latency	16	16	16	18	16	16	16	16	16
TP/S	2.03	1.99	1.87	5.15	3.48	2.91	**4.09**	**4.98**	**4.13**

To evaluate the advantages and disadvantages of using BRAMs on DES computational structures and the polymorphic DES implementation, a Xilinx VIRTEX II Pro 30 prototyping FPGA device has been used. The FPGAs embedded PowerPC is used as the core GPP [4].The PowerPC is running at 300 MHz, with a main data memory running at 100 MHz. The DES co-processor runs at the same frequency as the data memory, 100 MHz.

In Table 1, the two implemented DES computational structures, with and without BRAMs, are compared. In this table related DES stand-alone art is also presented. Note that these figures are for the DES kernel computation only.

From the implementation results of Our DES core with and without BRAMs on the VIRTEX-2 and VIRTEX-2 Pro FPGA technologies it can be concluded that a significant reduction on the required slices (37%), at the expense of 4 BRAMs, can be achieved. However, as a consequence, the critical path increases about 32%. This delay increase is due to the fact that a BRAM has a critical path equivalent to about 3 Look Up Tables (LUT), and the critical path of a LUT implemented SBOX is of 2 LUTs. Nonetheless, an improvement of 20% to the Throughput per Slice (TP/S) efficiency metric can be achieved. In these technologies and for the BRAM based structures, the slice occupation (2%) is the same as the BRAM usage (2%), thus an adequate utilization of the available resources in the device is achieved. In older technologies, where BRAMs are not so fast, like the VIRTEX-E, the penalty on the delay is higher. In this case, practically no improvement to the TP/S is achieved (only 2%).

When compared with related art, that use the unmodified DES algorithm structure, the proposed core has an equivalent Throughput per Slice as the commercial core from CAST, when compared with the proposed LUT based DES structure. The TP/S metric improves to 22% when compared with the BRAM based DES structure. When compared with [6] a TP/S metric improvement of 86% and 57% is achieved for the proposed structure with and without BRAMs, respectively.

In [7], the authors propose a modification to the DES computational algorithm, which allows for the efficient use of a pipeline computation, resulting in a very efficient computational structure. This improvement comes at the expense of a higher latency and a potentially lower resistance to side-channel attacks, since the same key is added at two locations, instead of one [9,10]. This algorithmic alteration also makes the usage of side-channel defences more difficult [11, 12]. Nevertheless, when no side-channel concerns exist, this structure is quite advantageous.

Taking into account that, the computational block used to perform the SBOXs operation is exactly the same in both papers; the same tradeoff between LUTs and BRAMs can still be applied to the design proposed in [7]. As a result, the 64 slices [7] required for the SBOXs can be replaced by 4 BRAMs, further improving the Throughput per Slice efficiency metric, as suggested by the results in Table 1. In the proposed usage of the DES core, as a polymorphic processor, the operating frequency is constituted by the memory not by the core itself. This means that the higher latency and pipeline depth makes the proposed structure [7] less advantageous.

For the experimental results a VIRTEX-2 Pro FPGA on a Xilinx University Program (XUPV2P) board. The comparative results for a pure software implementation and for the polymorphic usage are presented in Table 2. This table also presents the

Table 2: DES polymorphic performances

Bits	Hardware ThrPut	Software ThrPut	Kernel SpeedUp
64	89 Mbit/s	0.92 Mbit/s	97
128	145 Mbit/s	1.25 Mbit/s	116
4k	381 Mbit/s	1.92 Mbit/s	198
64k	399 Mbit/s	1.95 Mbit/s	205

speedup achieved for the kernel computation of the DES algorithm. In these results, a difference in the ciphering throughput can be seen, for different block sizes. This is due to the initialization cost of the of DES CCU, which includes the loading of the key and the transfer of the data addresses from the XREG to the DES core. This initialization overhead becomes less significant as the amount of data to be ciphered increases, becoming negligible for data blocks above 4 kbits. A speedup of 200x can be attained, achieving a ciphering throughput of 399 Mbit/s, working at the memory frequency of 100 Mbit/s.

Table 3 presents the figures for the proposed polymorphic DES core and for related art, using DES hardware acceleration. It can be seen that the proposed DES processor is able to outperform the related art in terms of throughput by 30% with less than 40% FPGA usage. This results in a Throughput per Slice improvement of 117%. Another advantage of this polymorphic computational approach is the capability to easily integrate existing software application in this embedded system, since existing applications just have to be recompiled, in order to used the dedicated DES hardware, as depicted in Figure 8.

Table 3: DES processors

	Chodo [13]	Our-LUT	Our-BRAM
Device	V1000	V1000E	V2P30-7
Freq. (MHz)	57	100	100
FPGA usage	5%	3%	2%
DES (Mbit/s)	306	399	399
3DES (Mbit/s)	102	133	133

6 Conclusions

In this paper, a hybrid hardware/software implementation of the DES algorithm was presented, using a polymorphic computational paradigm. The tradeoffs of using BRAMs to implement the DES SBOXs are also studied in this paper. Implementation results suggest that the Throughput per Slice metric can be improved by 20% with the use of BRAMs. The use of the BRAM implies a decrease on the maximum frequency, compensated by a significant reduction on amount of required slices. Implementation results suggest that for the complete DES core, the employed polymorphic paradigm and the tightly coupled organization between the General Purpose Processor (GPP) and the dedicated DES core, allow for a short development cycle and substantial performance improvement. Given that the DES core can directly access the main data memory and the usage of the exchange register to transfer the initialization parameters, the hardware implemented DES algorithm can be invoked in the same manner as the software implemented function. The parameter passing via the exchange register is performed by the compiler, thus making the usage of the DES core transparent for the programmer. Experimental results of the proposed processor on a VIRTEX II Pro FPGA, indicate that for data blocks of larger that 4 kbits a speedup of 200x for the DES algorithm can be attained, achieving a throughput of 400 Mbit/s for DES and 133 Mbit/s for 3DES. This performance improvement is achieved with a significantly low cost in terms of reconfigurable area, approximately 2% of the used device (328 slices and 4 BRAMS), and with a minimal development cost, since the integration of the dedicated hardware is performed by the compiler. In conclusion, with this polymorphic implementation of the DES algorithm, existing software application that demand high ciphering rates can be embedded with DES hardware implementations with a low development cost and without large reconfigurable resources.

Evaluation prototype

An evaluation prototype for the XUP prototyping board of the hybrid DES processor is available for download at http://ce.et.tudelft.nl/MOLEN/applications/DES

Acknowledgments

This work has been partially supported by the Portuguese FCT–Fundacão para a Ciência e Tecnologia, the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533).

References

1. NIST, "Data encryption standard (DES), FIPS 46-2 ed," tech. rep., National Institute of Standards and Technology, December 1993.
2. NIST, "Data encryption standard (DES), FIPS 46-3 ed," tech. rep., National Institute of Standards and Technology, 1998.
3. C. E. Shannon, "Communication theory of secrecy systems," *Bell Systen Technicl Journal*, vol. 28, pp. 656–715, Oct. 1949.
4. S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The Molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.
5. 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*, pp. 275–285, August 2001.
6. C. M. Wee, P. R. Sutton, and N. W. Bergmann, "An FPGA network architecture for accelerating 3DES CBC," in *International Conference on Field Programmable Logic and Applications*, pp. 654–657, Aug. 2005.
7. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Design strategies and modified descriptions to optimize cipher FPGA implementations: fast and compact results for DES and triple-DES," in *FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, (New York, NY, USA), pp. 247–247, ACM Press, 2003.
8. CAST, "DES Cryptoprocessor Core – XILINX FPGA Results." http://www.cast-inc.com/, 2007.
9. P. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks." http://www.cryptography.com/dpa/technical, 1998.
10. M.-L. Akkar and L. Goubin, "A generic protection against high-order differential power analysis.," in *FSE* (T. Johansson, ed.), vol. 2887 of *Lecture Notes in Computer Science*, pp. 192–205, Springer, 2003.
11. L. Goubin and J. Patarin, "DES and differential power analysis (the "duplication" method)," in *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, (London, UK), pp. 158–172, Springer-Verlag, 1999.
12. M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," in *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, (London, UK), pp. 309–318, Springer-Verlag, 2001.
13. Chodowiec, Gaj, Bellows, and Schott, "Experimental testing of the gigabit IPSec-compliant implementations of rijndael and triple DES using SLAAC-1V FPGA accelerator board," in *ISW: International Workshop on Information Security, LNCS*, pp. 220–234, 2001.