Exploiting FPGA Block Memories for Protected Cryptographic Implementations

Shivam Bhasin , Wei He , Sylvain Guilley and Jean-Luc Danger

Abstract—**Modern Field Programmable Gate Arrays (FPGAs) are power packed with features to facilitate designers. Availability of features like huge block memory (BRAM), Digital Signal Processing (DSP) cores, embedded CPU makes the design strategy of FPGAs quite different from ASICs. FPGA are also widely used in security-critical application where protection against known attacks is of prime importance. We focus ourselves on physical attacks which target physical implementations. To design countermeasures against such attacks, the strategy for FPGA designers should also be different from that in ASIC. The available features should be exploited to design compact and strong countermeasures. In this paper, we propose methods to exploit the BRAMs in FPGAs for designing compact countermeasures. BRAM can be used to optimize intrinsic countermeasures like masking and dual-rail logic, which otherwise have significant overhead (at least 2X). The optimizations are applied on a real AES-128 coprocessor and tested for area overhead and resistance on Xilinx Virtex-5 chips. The presented masking countermeasure has an overhead of only 16% when applied on AES. Moreover Dual-rail Precharge Logic (DPL) countermeasure has been optimized to pack the whole sequential part in the BRAM, hence enhancing the security. Proper robustness evaluations are conducted to analyze the optimization for area and security.**

Keywords: FPGA, Side-Channel Analysis, Block Memories, Countermeasures.

I. Introduction

Security is now one of the major driving factors of semiconductor industry. Often there is a need to secure the whole system-on-chip (SoC), which generally is achieved by embedded cryptographic cores (crypto-cores). Depending on the application, these crypto-cores are used to encrypt/decrypt sensitive data in all parts of the system, ranging from memory content to system-bus. A major threat known as "Side-Channel Attacks" (SCA [1]) has been pointed out about 17 years ago, but curiously the design of solid and efficient protections is still an open research area. SCA generally exploits the unintentional leakages from the physical implementation of the crypto-cores. This brings into play countermeasures to protect the physical implementation of cryptography, which can be classed into intrinsic and extrinsic countermeasures. Extrinsic countermeasures are applied in parallel to crypto-cores in order to confuse the attacker. Countermeasures involving generation of noise, misalignment of activity generally fall in this category [2].

Although extrinsic countermeasures have a limited overhead, their resistance depends on the power of the attacker. Consider a noise generator which is deployed to provide $2\times$ SCA resistance than the unprotected crypto-core. The power of the countermeasure is related to the extra effort required by the attacker to acquire twice the number of traces. If the attacker needs only a couple of seconds more to acquire the extra traces, then the security enhancement is negligible. Therefore a common practice is to combine several extrinsic countermeasures with protocol level countermeasures. However provable security is not assured.

Intrinsic countermeasures are the other solution which, as the name suggests, are built into the algorithm. These countermeasures modify the implementation of the cipher in order to leak little or no sensitive information in the side channel. Also these countermeasures often come with a non-negligible overhead. Intrinsic countermeasures further fall into two wide categories, i.e., masking and hiding.

Hiding countermeasures generally comprise of dual-rail precharge logic (DPL [3]). DPL is a circuit-level countermeasure which aims at flattening or removing the data-dependent leakage from the circuit. Removal of data-dependant leakage is achieved by putting in place a generated False (F) rail that works simultaneously together with the original True (T) rail for compensating each other's activity. DPL operates in two phases: *Precharge*, i.e., where all the values are reset to a constant value, and *Evaluation*, where the cryptographic computation is performed. The two-phase operation with a dual-rail structure (theoretically) ensures constant activity and is therefore free from any exploitable data-dependent leakage.

Masking on the other hand is generally applied at the algorithmic level. The basic idea of masking is to protect all the sensitive intermediate values inside a cryptographic algorithm by applying a random mask [4]. The random mask is removed at the end, which involves complex computation on the value of mask, generally done by implementing the masked path in parallel to the actual algorithm. The linear

operations of a cryptographic algorithm can be easily tuned to masking. Masking the non-linear operations is not an easy task, as the overhead associated with it is exponential.

For a secure implementation, DPL needs balanced placement and routing of its component. Masking does not have such strict requirements at the circuit level but the non-linear operation is often hard to be realized in a secure manner. The availability of high-density block memories (BRAMs) in FPGA can help to solve both problems. BRAMs are capable of storing huge tables, which are often present in the non-linear part of protected ciphers (e.g., masked/ dual-rail Sbox). Thus intrinsic countermeasures become realizable in FPGA due to BRAMs. Several other features (discussed in Sect. II-A) are present in BRAMs which can be exploited to optimize the implementation of the cipher. BRAMs are also known to provide elevated security as compared to its logic counterpart [5], and are often recommended to implement intrinsic countermeasures. BRAM are also largely deployed in implementing hash function and other cryptographic applications.

In this paper, we concentrate on BRAMs present in FPGAs in the context of intrinsic countermeasures. In particular, we propose methods to efficiently use BRAM to implement countermeasures with reduced area overhead and higher SCA resistance. Although generic countermeasure are favourable, it as well makes sense to exploit new features to realize compact and robust countermeasure. Firstly, we propose a method to exploit the features of BRAM in-order to implement masking and DPL countermeasures with limited overhead. The proposed optimizations are applied on a real AES-128 co-processor. All the AES implementations tested implement the sboxes in BRAMs, as this configuration has been shown to offer enhanced resistance against SCA [5]. Next we analyze the security of these countermeasures in the presence of BRAM. We show that it is possible to use modern FPGA features to effectively implement intrinsic countermeasures.

The rest of the paper is organized as follows: Sect. II gives general background on BRAM architecture in FPGA, its application in masking and DPL countermeasures. Next in Sect. III, we propose two methodologies to exploit BRAM features in an FPGA to optimize masking and DPL countermeasures respectively. The proposed optimization are applied on an AES-128 co-processor for experimental validation. The SCA evaluation of proposed protection methodologies is discussed in Sect. IV. Finally, Sect. V draws general conclusion.

II. BRAM IN CRYPTOGRAPHIC APPLICATIONS

In this section, we first discuss the features of an FPGA BRAM. A special focus is laid on the application of these features to optimize SCA countermeasures. Thereafter a general background of the used countermeasures, i.e., Masking and DPL are provided.

A. Block RAM in Modern FPGA

Modern FPGAs possess huge blocks of memories which are synchronous in nature. For example, the latest Xilinx FPGAs have several blocks of 36Kbits true dual-port memories. The exact design of these BRAMs is not public but a few details about the general architecture of these BRAMs are documented [6]. Fig. 1 shows one port of a dual-port BRAM in Spartan-6 FPGA. It can be deduced from the figure that the BRAM contains register to synchronize input data and address before accessing the memory array. The memory array is followed by a latch and an optional output register. BRAM also contains several signals to control the use of output register or set/reset the value of the latch and output register. Altera AltSyncRam [7] also possess a similar BRAM architecture. Therefore the presented solution can also be extended to Altera FPGAs.

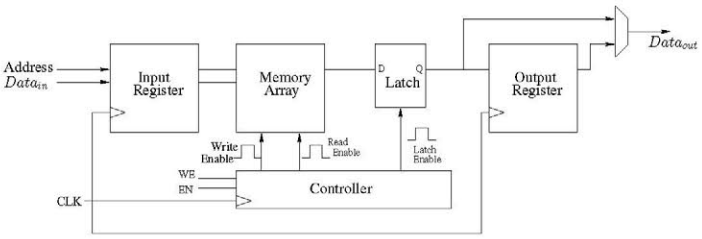


Fig. 1. Internal Architecture of Xilinx BRAM.

TABLE I
FEATURES OF A XILINX BRAM. IN THE TABLE, 1 SIGNIFIES AN IMPROVEMENT IN AREA OR PERFORMANCE AND 2 SIGNIFIES AN IMPROVEMENT IN SCA RESISTANCE.

BRAM Feature	Application to Cryptography
High Density RAM	To implement huge data[1]
Internal Register at input	To implement state register[1] Not connected to FPGA routing[2] No glitches[2]
Dual-Port Nature	Single block for multiple Sboxes[1]
Output Register	Available resource[1] To achieve better timing[2]
Reset	To enable precharge propagation in DPL[1,2]
Hard Macro in $\leq 65nm$ CMOS	Low leakage power[2] To balance placement[1,2]

As previously stated, BRAMs are recommended for crypto-applications. Tab. I summarizes the features of a BRAM and their use in relation to cryptographic applications.

Some of these options have already been used in cryptographic applications. **Internal Register at input** for state and **Dual-Port Nature** was first used by Drimer et al. in [8]. **Reset** in the BRAM was also used in Separated Dynamic Differential Logic (SDDL [5]) to enable precharge propagation. In [9], authors have shown that the internal register at input (address) of BRAM leaks very less and difficult to attack. Moreover, we assume that it is very unlikely to seperate activities of the two ports of a BRAM being a hard-macro in $\leq 65nm$ CMOS.

B. Masking and the use of BRAM

Masking relies on variable representation of sensitive data into randomized shares [10]. A d^{th}-order masking scheme splits a sensitive variable $Z \in \mathbb{F}_2^n$ into $d+1$ random shares, noted $\vec{S} = (S_i)_{i \in [\![0,d]\!]}$, in such a way that the relation $S_0 \perp \cdots \perp S_d = Z$ is satisfied for a group operation $\perp$ (*e.g.*, the XOR operation in Boolean masking). For a simple Boolean

masking scheme, order $d = 1$. When masking is implemented in hardware, generally the mask as well as the masked data are computed in parallel. Keeping this detail in mind, the leakage function for the first-order masking countermeasure in hardware can be expressed as:

$$L = HW(Z \oplus M) + HW(M) + N\,. \tag{1}$$

The share M is the random mask uniformly distributed over $\mathbb{F}_2^n$ and the share $Z \oplus M$ is the masked variable. Variables Z and M are assumed to be mutually independent. The linear parts of the cipher are easier to be masked but the computation of non-linear Sbox S in presence of masking is difficult. It involves computing $S(Z) \oplus M'$ from the variables M, $Z \oplus M$ and M' (new mask) without compromising with SCA resistance.

To deal with this problem, one of the most common solutions is *the Generalized Look-Up Table (GLUT [11]).* The main idea of GLUT is to precompute a look-up table, associated to the function $S' : (X, Y, Y') \mapsto S(X \oplus Y) \oplus Y'$. To compute the masked variable $S(Z) \oplus M'$, GLUT performs a table look-up of $GLUT[Z \oplus M, M, M']$. Thus the value $S(X \oplus Y) \oplus Y'$ has been precomputed for every possible 3-tuple of values. For first-order masking, the output mask and the input mask are equal (*i.e.*, $M = M'$). In this case, the dimension of the table is $2n$ instead of $3n$ and the look-up table becomes $GLUT[Z \oplus M, M]$, where Z, M and M' are variable of n-bits. Owing to its structure the preferred target is a BRAM. Compared to an unprotected Sbox S of size $2^n \times p$, a first-order masking GLUT requires $2^{2n} \times 2p$. Very often the hardware implementations computes the whole state in parallel, requiring multiple instances of GLUT. Therefore the basic GLUT technique can be sometimes difficult to be realized in FPGA when n is high (for example $n = 8$ in AES). The size of GLUT further explodes when the desired resistance is of order $d \geq 1$.

An optimized version of GLUT in FPGA logic was proposed in [12] with a net overhead of roughly $3\times$. However the implementation of GLUT in logic is sensible to higher-order attacks which exploit the leakage due the glitches. In [2], authors propose a first-order SCA resistant countermeasure using BRAM scrambling. BRAM scrambling implements a $2^n \times p$ masked Sbox with a single mask. This Sbox uses the same mask for several encryption, which limits the order of SCA resistance. In the mean time, another Sbox which is masked with a different mask is written to the other port of BRAM. Once the second Sbox is ready, it is used for encryption while the first Sbox is refreshed with a new mask. Another first-order countermeasure in the same line was proposed in [13], which proposes the reuse of Sboxes to reduce overhead. The main advantage of this masking scheme is that it does not need a parallel mask-computation path which also forms a basis for our masking scheme. Our masking scheme uses "precomputed" Sboxes with a random (secret) offset for every encryption. We show that it is possible to design a masking scheme with reduced entropy $< n$ bit, and achieve SCA resistance up to order d for a well chosen set of mask.

C. DPL and use of BRAM

The modus operandi of dual-rail circuits is to add redundant logic of opposite nature to achieve constant activity irrespective of the data processed. A DPL protocol converts every bit x to $(x_\mathrm{T}, x_\mathrm{F})$. Complementary values of x_T and x_F are desired for a proper balance and thus considered as valid values. Similar values for the pair $(x_\mathrm{T}, x_\mathrm{F})$ can be used as separators between valid values. Thus DPL operates in two phases where valid values are propagated in evaluation phase and a spacer in precharge phase. Following the conditions stated above, DPL ensures a constant activity of each compound gate pair. However, when DPL expands from a single gate to a complex circuit, different placement and routing delays introduces other imbalances.

Fig. 2 shows the Wave Dynamic Differential Logic (WDDL [3]): one of the first introduced DPL for FPGA. It can be deduced that all logic gates (except inverters) lead to an overhead of 2 while flip-flops results in an overhead of 4. WDDL also has a restriction of using only positive gates which further adds up to the overhead. In Fig. 2, the gates G and $\overline{G}$ are well balanced but if their inputs arrive at different time, an imbalance cannot be avoided. Thus proper placement and routing is required for a secure DPL design, in absence of which, DPL could fail due to early propagation effect (EPE [14]) or routing imbalance [15]. EPE arises from different evaluation time of a logic gate depending on difference in arrival of inputs. Routing imbalance is observed due to asymmetrical routing of T and F rails. Since then, several improvements to WDDL have been proposed to improve its resistance. One interesting proposal to counter the routing imbalance was called as MDPL (Masked Dual-rail Precharge Logic). MDPL randomly swaps the true and false routing network to eliminate routing imbalance and also EPE in iMDPL (improved MDPL [16]). This security improvement of iMDPL came at an area overhead even greater than WDDL.

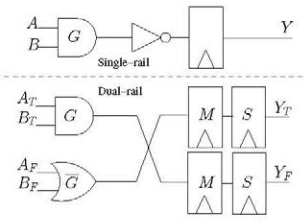


Fig. 2. WDDL building block.

Due to the discussed issues, DPL was not considered as a good countermeasure specially for FPGA application where a designer has very limited freedom over choice of gates, placement and routing. Thereafter a couple of DPL countermeasures were proposed which were able to use BRAM at a reasonable cost. One of the BRAM based DPL, is SDDL [5]. SDDL used BRAMs at an area overhead of $2\times$ compared to the unprotected design. This limited overhead comes from the reset feature present in the Xilinx BRAM which can reset the output as desired. The reset was used for precharge propagation at the output of the Sbox. Another DPL called

BCDL (Balanced Cell-based Dual-rail Logic [14]) can also use BRAM at an overhead of $4\times$ owing to a synchronization. The synchronization signal of BCDL also solves the problem of EPE. However both SDDL and BCDL do suffer from routing imbalance and therefore need back-end techniques for balancing the dual-rail. AES with T-tables reduces the fanout which in a way reduces routing imbalance and makes back-end balancing easier [17].

III. Exploitation of BRAM to optimize countermeasures

In this section, we propose two methods benefited from BRAM features for implementing secure circuits at a reasonable cost. The first method is applied to masking countermeasures by exploiting huge memory array and dual-port nature of the BRAM. The next method presents a new way (using BRAM) to organize the sequential part of crypto-algorithm in a compact and balanced manner.

A. Optimized Masking Implementation using BRAM

Several solutions are proposed to mask the non-linear operation (now called Substitution box or Sbox) of a cipher but all solutions have a significant overheads. Since we are using BRAM in our implementation, we focus on GLUT as the solution to mask the Sbox. GLUT is a precomputed table which accepts the masked Sbox input (n-bits) and the mask (n-bits) as inputs. It returns a masked Sbox output (p-bits) and the correction value (p-bits). For example, in DES a 64×4 Sbox is replaced by GLUT of size 4096×8. Similarly for AES, the size of the GLUT is 65536×16 for a 256×8 Sbox. Please note that in hardware where implementations are parallel in general, several instances of a Sbox are used and all of them must be masked. In a low-cost FPGA like Xilinx Virtex-5 LX30, a parallel DES implementation is still possible but not for AES. A single AES GLUT would occupy about 90% of the available BRAM, making a parallel AES implementation unfeasible.

It is possible to design a masking implementation which reduces the overhead of GLUT still keeping it resistant to the certain higher order of side-channel attacks. Masking schemes can reduce the GLUT overhead by reusing the mask and thus reducing the overhead from $2^{2n} \times 2p$ to $2^{n+k} \times p$ where $k < n$ is the entropy of the mask. In other words, instead of using 2^n different values to mask the data, only 2^k values are used. For a proper hardware optimization, the number of Sboxes in a cipher N should be a multiple of k. Such an implementation generally protects against first-order attack, however by application of coding theory, the right set of mask can be chosen to resist zero-offset higher-orders (univariate attacks targeting a single Sbox). In the following, we consider univariate attacks which combine different leakages.

For simplicity, we restrict ourselves to ciphers (e.g., AES, PRESENT) where all the N Sboxes are the same and of bijective construction, i.e., of the format $2^n \times n$. Ciphers not abiding by these conditions are still possible to protect by this scheme with an extra overhead. The details of this masking scheme are as follows. Firstly, a set of 2^k n-bit mask M is chosen. Now both the input and output of each Sbox S are masked as: $S(x \oplus m_i) \oplus m_{i+1}$ where m_i and m_{i+1} are consecutive elements of the set M. Actually i and $i+1$ are to be understood as $(i \mod 2^k)$ and $((i+1) \mod 2^k)$ (omitted for simplicity of representation). The masked Sbox is now denoted as S_m and is of the same size as unmasked S. If 2^k is equal to N then all the Sboxes are unique. At each round of the algorithm, the Sboxes S_m are reused by circular rotation of one position. Let us consider a masked state $x' = x \oplus m_i$ is computed by S_{m_i} which is masked with m_i in the current round r. In the next round, x' is processed by $S_{m_{i+1}}$. Precisely the computation done by $S_{m_{i+1}}$ will be $S(x' \oplus m_i) \oplus m_{i+1}$ which is simplified to $S(x) \oplus m_{i+1}$. Similarly in the next round, mask m_{i+1} is removed at the input of Sbox $S_{m_{i+1}}$ and m_{i+2} is applied at the output. If the Sboxes are not bijective, an expansion function should be put in place to make the output of Sbox coincide with size of the mask.

The set of mask M can be public however the M should be shifted by a random offset before each encryption. M is chosen such that the jth order moment of the conditional leakage $L^j|Z = z$ given a guess on the sensitive variable Z are all the same for $j = 1, 2, \cdots, d$. Thus only an attack of order $(d+1)$ can succeed. Under this constraint, the masks set M must be an orthogonal array of strength d [18]. The linear operations are masked by a simple XOR operation with precomputed constants applied at the end of each round. The $N \times n$ bit constants are chosen as a function of initial offset and can be stored in BRAM as well. It is not always possible to find a solution for M which resists at order d. Another feature of FPGA which comes handy in such cases is dynamic reconfiguration. If it is not possible to find a solution for M at order d, designers can opt for several sets of M with order $< d$ and update them regularly. Since the mask dependent part is inside the memory, modern FPGA kits have specific tools which can reconfigure the FPGA to just change the BRAM content. Alternately, concurrent read and write technique used in [2] can be used by doubling the memory overhead.

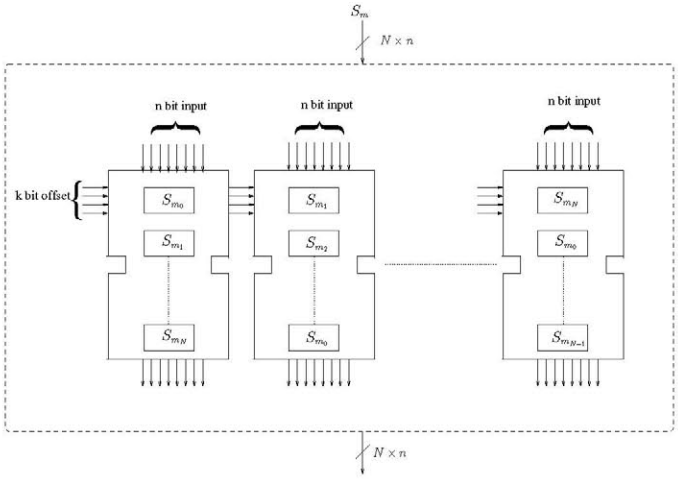


Fig. 3. Optimized implementation of proposed masking scheme without barrel shifters

The required rotation in the presented masking scheme can be done using barrel shifter. Since the barrel shifters are composed of series of multiplexers which are a major source of glitches in FPGA, they can cause unintentional leakage. Barrel shifters are also resource consuming and affecting the performance of the whole system. For example, a 128-bit barrel shifter would alone acquire around 100 slices in Xilinx Virtex-5 LX-30 FPGA. BRAM can be very efficiently used in this application to get rid of barrel shifters and thus glitches. The scheme to organize the Sboxes and implement them in BRAM is shown in Fig. 3. These Sboxes can be further compressed by using dual-port memory. All the masked Sboxes S_m are placed in each BRAM. From one BRAM to another, S_m are laid with an offset of 1. Thus the BRAM has an input address of $n+k$ bits where n is the input of the S_m and k selects the correct masked Sbox from S_{m_0} to S_{m_N}. k forms the most significant bits of $n+k$. Thus the memory cost is multiplied by 2^k but can be small in terms of number of blocks. Since all the BRAM contain same Sboxes in different order, the dual-port feature can be used to access the same data with the corrected offset.

1) Application to AES-128: Now we apply the presented scheme to secure a parallel AES-128 co-processor which computes one round per clock cycle. For AES, $n = 8$ and $N = 16$. We found that it is possible to select a mask M for AES which resist up to order $d = 3$. M is the cosets of the linear code $[8, 4, 4]$ and thus $k = 4$ of 16 mask. We found the set

M= [245, 226, 222, 201, 187, 172, 144, 135, 120, 111, 83, 68, 54, 33, 29, 10],

should be order 3 resistant. To optimize the scheme we use the input register of BRAM as state register. An unmasked AES Sbox is 2Kb which makes the composite Sbox (S_{m_0} to S_{m_N}) of size $16 \times 2 = 32Kb$. This composite Sbox which easily fits in a Xilinx BRAM of 36Kb, now has 12 bits of address, i.e., 8 bits corresponding to masked byte concatenated with 4 bits of offset. Moreover the dual-port feature of the BRAM can reuse the same memory space with two different ports. Thus $N = 16$ Sboxes need only $N/2 = 8$ BRAM. The overhead of presented masking scheme as compared to unprotected reference AES is shown in Tab. II. The precomputed round unmasking constants are implemented in BRAM which consumes 8 extra blocks. The net overhead in terms of slices is only 16% with minor loss of frequency. Since higher-order attacks of order 4 and greater are difficult to realize in practise [19], an order 3 masking with mere overhead of 16% is a very practical solution.

TABLE II
AREA AND FREQUENCY OVERHEAD OF MASKED AES AFTER OPTIMIZATION ON VIRTEX-5.

Architecture	Unprotected	Masked	Overhead
Slices	733	856	1.16×
Registers	0	0	0×
BRAM	8	16	2×
Max. Frequency [MHz]	144.3	141.1	1.02×

B. Optimized DPL Implementation using BRAM

DPL involves duplication of each component of the circuit to ensure a balanced activity. Duplication of standard logic is simple which leads to an overhead of little over twice in terms of resources used. However a simple duplication of memory leads into exponential increase in overhead. A memory of size $2^n \times p$, will have an overhead of 2^{n+1} up on duplication. This overhead can be reduced to just $2\times$ by using BRAM properties. The BRAM overhead is not the only problem. For a DPL circuit to have a constant activity in every cycle, a precharge spacer should flow through the whole circuit.

We propose a method to further optimize FPGA implementations of DPL both in terms of area and security. This optimization exploits the following features of BRAM: input register, output register with reset, dual-port nature and hard macro. A DPL flip-flop is made of 4 flip-flops (Fig. 2), where each flip-flops pair (master-slave) is located in the true and false rails. The input register can be used for the master flip-flop and the output register serves as the slave. The use of output register also introduces a latency of one clock cycle. The extra cycle latency is not a problem in DPL because it aids the two-phase DPL protocol. Moreover, the dual-port feature allows to implement the true and the false rails of the flip-flop. The optimization scheme is depicted in Fig. 4.

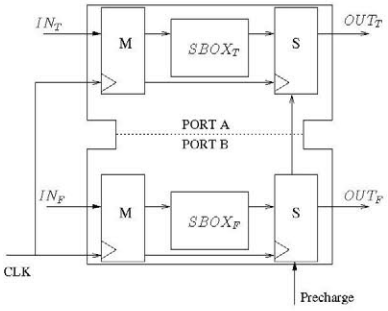


Fig. 4. Proposed Scheme to implement a DPL SBOX and Flip-Flops in a BRAM.

A very common issue in DPL design is the propagation of precharge or the spacer. Since the BRAM will be preceded by some combinational circuit, the spacer is easily propagated to the input register of the BRAM. To precharge the output register, the reset (also known as SSR [5]) feature provides just the right solution. Only the combinational gates are implemented in FPGA slices. The proposed architecture brings a three-fold advantage for implementing DPL design into FPGA. Firstly, the logic is not used to implement two-stages (otherwise leading to $4\times$) of the state registers thus significantly reducing the overhead. Secondly, the regular structure of BRAM ensures proper and balanced placement of the main leakage source of the design, i.e., state register. Finally, it is known that leakage from a BRAM itself is less than flip-flops in FPGA slices, thus enhanced SCA resistance [5]. The balanced placement of XOR gates can be ensured by using $LUT6\_2$ from Xilinx to place the whole dual-rail cells (G and $\overline{G}$ in Fig. 2). Balancing routing in FPGAs is challenging, because FPGA architecture and CAD tools are not designed for these weird DPL structures.

But some amount of balancing can be achieved by proper placement. Balancing routing is another area of research, and repair techniques like proposed in [15] can repair routing with extra effort, but it falls out of the scope of this paper.

1) Application to AES-128: To test our proposed optimization on a real DPL circuit, we had two choices: SDDL and BCDL. To our knowledge, SDDL and BCDL, are the only DPL subsets proposed which are capable of using BRAM at reasonable cost. Since SDDL suffers from security issue like EPE, we choose to apply our optimization on BCDL. The target algorithm is AES-128 using T-tables because T-tables merge SubBytes and MixColumns function in a precomputed table, thus reducing the routing fanout. Precisely, we use the implementation of AES-128 protected by BCDL as described in [17]. Applying our technique to further optimize BCDL, BRAMs and flip-flops are merged into a single entity.

TABLE III
AREA AND FREQUENCY OVERHEAD OF BCDL FOR AES MODULE EXCLUDING KEY EXPANSION AFTER OPTIMIZATION ON VIRTEX-5.

Architecture	Unprotected	BCDL	Overhead
Slices	176	1128	6.4×
Registers	0	0	0×
BRAM	8	16	2×
Max. Frequency [MHz]	258	283	0.911×

The overhead of protecting AES with BCDL after applying our optimization is given in Tab. III. Both the unprotected AES and its BCDL version implement state registers in BRAM. The number of slices is increased by roughly 6.4× as XOR in BCDL is costly as also pointed in the original paper [17]. It is limited to 1-LUT per bit of XOR. The BRAMs are simply doubled, while the performance is improved due to the usage of output register of the BRAM. As BRAMs are a hard-macro, balanced placement of the sequential part of AES (128*4 bits registers) is ensured without placement constraints. DPL balancing has been checked by the post P&R i.e. absolutely close to on-device conditions. Besides, BCDL is free of glitch by design. We do a proof of concept study to quantify the gain of balanced placement and keep routing untouched for the two designs to have a fair evaluation.

IV. SECURITY ANALYSIS

The previous sections dealt with the implemetation aspect of the proposed optimizations to masking and DPL. Now we analyze the implemented countermeasures from a security aspect with respect to SCA.

A. Attack Metrics and Experimental Platform

Let us denote a random variable L representing the side-channel leakage (*e.g.*, power consumed) while computing $Z = f(X.K)$. K is the n-bit secret key and X is a variable quantity known to the attacker. Time is another parameter not shown. A standard SCA tries to find correct key $k^\star$ for which Z and L have maximum dependency. Since L is noisy, thus several measurements of Z are required to estimate L. For hardware implementation, the leakage L depends on the *Hamming distance (HD) model*. It expresses at first-order the power consumption of CMOS gates in electronic devices as it corresponds to signal transitions. The leakage can be expressed as:

$$L = HD(Z, R) + N = HW(Z \oplus R) + N\,,$$

where N is the noise, and R is the reference state. $HW(X)$ is the *Hamming Weight function* which returns the number of bits set to 1 in binary representation of X.

For SCA analysis, we use Correlation Power Analysis (CPA [1]) as a distinguisher. CPA is a computation of the *Pearson Correlation Coefficient* ρ between the side-channel leakage T and the expectation of the leakage model L knowing Z, noted $\mathbb{E}(L|Z)$, which can be estimated as:

$$\rho(T, \mathbb{E}(L|Z)) = \frac{\Sigma_{i=0}^{n}(t_i - \mu_T)\cdot(\mathbb{E}(L|Z = z_i) - \mu_{\mathbb{E}(L|Z)})}{\sigma_T \cdot \sigma_{\mathbb{E}(L|Z)}}\,,$$

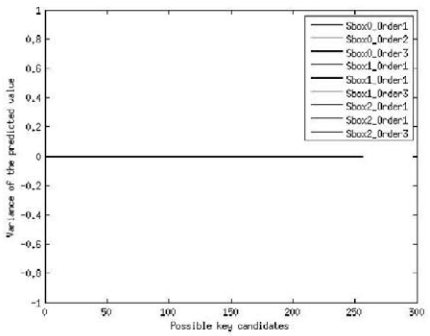
where σ and μ denote the standard deviation and the mean respectively, and n is the traces count. To analyze the efficiency of SCA, two metric are used. The first metric is Minimum Traces to Disclosure (MTD), i.e., the minimum number of measurements needed to perform a successful attack. The other metric used is called guessing entropy which generally is useful when an attack is not successful. Guessing entropy gives the average number of key hypothesis to test to reveal the correct key.

We test our designs on Xilinx Virtex-5 FPGA soldered on a SASEBO-GII platform. For SCA, traces are acquired on a 54855 Infiniium Agilent oscilloscope with a bandwidth of 6 GHz and a maximal sampling rate of 20 GSample/s, using an antenna of the HZ–15 kit from Rohde & Schwarz. Since the analysis results can widely vary from one measurement setup to another, we always use a reference implementation to give readers an idea of the security gain achieved.

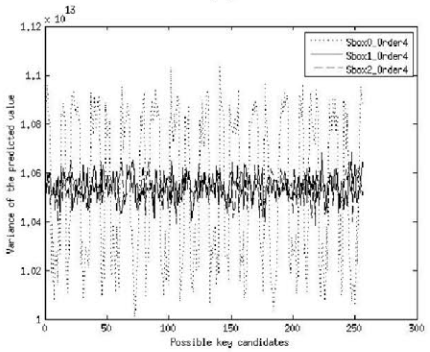
B. Security Analysis of Masked AES

To develop the leakage-function of masking we refer back to Eq. (1). For a first-order mask, the prediction function $z \mapsto \mathbb{E}(HW(Z \oplus M) + HW(M)|Z = z)$ reduces to a constant and makes simple SCA attacks impossible. To exploit the leakage from masked implementation zero-offset SCA [20] are often used. These attacks are based on the principle that higher-order moments are related to the key. For a masking of order d, the $d+1$ order moment can be key-dependant. Theoretically, $\rho(T^{d+1}, \mathbb{E}(L|Z))$ should result in a successful attack, where T are the centered side-channel traces. However, as the order d increases, CPA becomes less practical because the noise in the traces is amplified. Moradi in [19] suggests that attacks of order 5 and greater can be considered far from practise. We acquired 150,000 side channel traces (averaged 16 times) for the masked implementation explained in Sect. III-A1.

The presented masking is a special case where the number of mask is 16 and the mask set is public. The secret is the 4-bit offset which is not known to the attacker. In this case, the leakage function can be written as $HW(Z \oplus M)^d$, d being the power at which the attacker raises the centered traces. The prediction function is $z \mapsto \mathbb{E}(HW(z \oplus M)^d)$, *i.e.*, the leakage to the power d averaged over the whole set M for all



(a)



(b)

Fig. 5. (b) Variance of the predicted leakage for first three Sboxes at (a) order 1, 2 and 3; (b) Order 4 leakage L

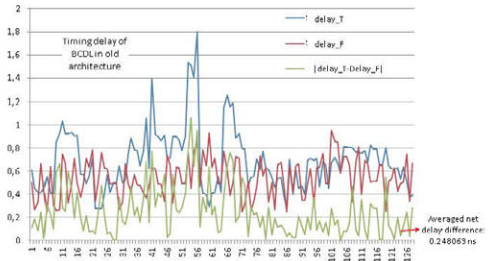
offsets. We tested the values of possible predicted leakage (i.e. $\mathbb{E}((z \oplus M)^d)$) for 256 possible subkeys of two chosen Sbox for order 1-4. As expected, the predictions came out to be constant for order 1, 2 and 3 which renders the attack impractical. Since the actual prediction is constant, its Hamming weight will also be constant. Only at order 4, the predictions vary from each other as shown in Fig. 5 (b) which points towards possibility of an attack. In Fig. 5 (b), it is normal to have a higher variance of prediction in Sbox 0 as it covers only 163 of the 256 value due to absence of ShiftRows. Sbox 1 and 2 cover all 256 values and thus show lower variance. We tried a 4-th order attack on the set of acquired traces which failed probably due to limited number of traces or high noise at order 4. Thus the masking scheme is shown to be compact and secure at least up to order 3.

C. Security Analysis of DPL AES

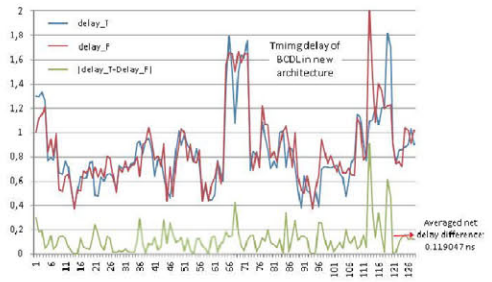
Theoretically, a DPL design should be leakage-free. DPL is a special case where any two evaluations are separated by a precharge phase. In a well-balanced DPL circuit, along with every Z, a $\overline{Z}$ is also computed, which can be modelled as:

$$L = HW(Z) + HW(\overline{Z}) + N\,.$$

Ideally, HW(Z)+HW($\overline{Z}$) is a constant which reduces the leakage L to just noise. Now if we also consider placement and routing imbalance, Z and $\overline{Z}$ do not occur simultaneously. Thus for short periods of time, L depends either on HW(Z) or HW($\overline{Z}$), which reduces the model to $HW(Z)$.



(a)



(b)

Fig. 6. Dual-rail timing bias in (a) BCDL_OLD, (b) BCDL_NEW

Now we try to quantify the security improvement brought due to balanced placement of BCDL by the proposed optimization. As previously stated, net delay bias has significant impacts on the balance between the dual rails of DPL logic. We achieved better routing balance between the nets for the security sensitive nets in the optimized BCDL (now referred as BCDL_NEW) as compared to original BCDL (now referred as BCDL_OLD) version. We would like to remind the readers that the BCDL_OLD implements the state register in FPGA slices. Fig. 6 depicts net delays and the differences between each of the 128 pairs of input nets to the flip-flop (same as BRAM input in BCDL_NEW). We choose these nets because they accumulate the maximum delay and therefore are most sensible to bias. Values for T and F rails are outlined by different colours. BCDL_NEW has a smaller delay difference as compare to BCDL_OLD. Averaged delay bias from the old BCDL to the new one is reduced roughly from **0.25ns to 0.12ns**, roughly a reduction factor of **1.48**.

Further we analyzed the two architectures using CPA over 100,000 traces which were averaged 16 times. The result is shown in Fig. 7. The Sboxes with MTD more than 100,000 traces indicate insufficient traces for a successful attack. So we plot the guessing entropy of the correct key in Fig. 7(b). It can be simply deduced from the plot that with the optimization, the resistance has been improved. We cannot directly connect the timing result with the CPA result because of lack of precise information on the physical properties of the device. However, both the timing and CPA results favour the improved BCDL (BCDL_NEW).

V. Conclusions and Perspectives

In this paper, we investigated the power of BRAMs available in FPGAs to implement intrinsic countermeasures. BRAMs possess many features which can aid the designers of cryptographic circuits. These features like presence of registers at

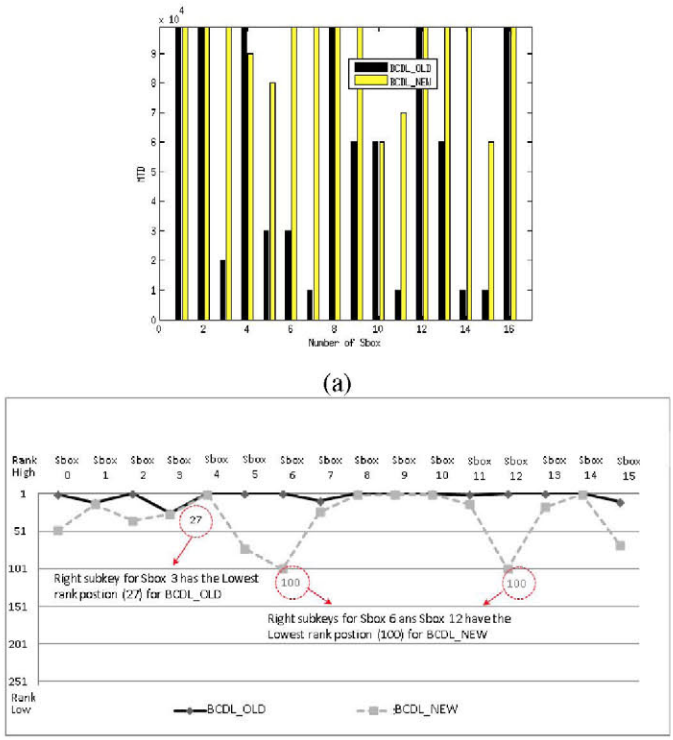


Fig. 7. (a) MTD and (b) Guessing entropy for the two BCDL circuits after CPA on 100k traces

input and output, ability to reset the output register, dual-port nature can be very well exploited. Also the regular structure of BRAM (hard-macro) saves the designers from applying specific placement constraints. We exploit these features to propose compact and secure implementation of existing countermeasures (masking and DPL). The optimizations have been applied on AES co-processor and tested on Xilinx Virtex-5 FPGA. Their security analyses reveal positive results. The masking countermeasure had an overhead of only 16% and was shown to be secure for the chosen model, thanks to the removal of barrel shifters. In the DPL countermeasure, the whole sequential part of AES that is also the main source of leakage was packed inside the BRAM with balanced placement by design. To our knowledge, the implementations proposed are the most compact of the state-of-the-art. Thus security is a another parameters which motivates integration of ample BRAM resourses into FPGA chips.

Finally we would like to conclude that the security of a countermeasures depends specific leakage model of the device. Therefore it should be interesting to research formal methods to characterize leakage models for the given device.

ACKNOWLEDGMENTS

This research is partly supported by Strategic International Cooperative Program (Joint Research Type), Japan Science and Technology Agency (JST), and the French Agence Nationale pour la Recherche (ANR), via grant for project SPACES (Security evaluation of Physically Attacked Cryptoprocessors in Embedded Systems).

REFERENCES

[1] É. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES*, ser. LNCS, vol. 3156. Springer, August 11–13 2004, pp. 16–29, Cambridge, MA, USA.

[2] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," in *CHES*, ser. LNCS, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 33–48.

[3] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *DATE'04*. IEEE Computer Society, February 2004, pp. 246–251, Paris, France. DOI: 10.1109/DATE.2004.1268856.

[4] L. Goubin and J. Patarin, "DES and Differential Power Analysis. The "Duplication" Method," in *CHES*, ser. LNCS. Springer, Aug 1999, pp. 158–172, Worcester, MA, USA.

[5] R. Velegalati and J.-P. Kaps, "Techniques to enable the use of block RAMs on FPGAs with dynamic and differential logic," in *International Conference on Electronics, Circuits, and Systems, ICECS 2010*. IEEE, Dec 2010, pp. 1251–1254.

[6] Xilinx, "Spartan-6 FPGA Block RAM Resources User Guide — UG383 (v1.5)," http://www.xilinx.com/support/documentation/user_guides/ug383.pdf.

[7] Altera, "Stratix-II Device Handbook — Volume 1," http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf.

[8] S. Drimer, T. Güneysu, and C. Paar, "DSPs, BRAMs and a Pinch of Logic: New Recipes for the AES on FPGAs," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 14-15 Apr 2008, pp. 99–108, stanford, Palo Alto, CA.

[9] S. Bhasin, S. Guilley, A. Heuser, and J.-L. Danger, "From cryptography to hardware: analyzing and protecting embedded Xilinx BRAM for cryptographic applications," *Journal of Cryptographic Engineering*, vol. 3, no. 3, 2013. [Online]. Available: http://dx.doi.org/10.1007/s13389-013-0048-4

[10] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," in *CRYPTO*, ser. LNCS, vol. 1666. Springer, August 15-19 1999, Santa Barbara, CA, USA. ISBN: 3-540-66347-9.

[11] E. Prouff and M. Rivain, "A Generic Method for Secure SBox Implementation," in *WISA*, ser. Lecture Notes in Computer Science, S. Kim, M. Yung, and H.-W. Lee, Eds., vol. 4867. Springer, 2007, pp. 227–244.

[12] F. Regazzoni, Y. Wang, and F.-X. Standaert, "FPGA Implementations of the AES Masked Against Power Analysis Attacks," in *COSADE*, February 2011, pp. 56–66, Darmstadt, Germany.

[13] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger, "RSM: a Small and Fast Countermeasure for AES, Secure against First- and Second-order Zero-Offset SCAs," in *DATE*, March 12-16 2012, pp. 1173–1178, Dresden, Germany. (TRACK A: "Application Design", TOPIC A5: "Secure Systems").

[14] M. Nassar, S. Bhasin, J.-L. Danger, G. Duc, and S. Guilley, "BCDL: A high performance balanced DPL with global precharge and without early-evaluation," in *DATE'10*. IEEE Computer Society, March 8-12 2010, pp. 849–854, Dresden, Germany.

[15] W. He, A. Otero, E. de la Torre, and T. Riesgo, "Automatic generation of identical routing pairs for fpga implemented dpl logic," in *ReConFig*. IEEE, 2012, pp. 1–6.

[16] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard, "Evaluation of the Masked Logic Style MDPL on a Prototype Chip," in *CHES*, ser. LNCS, vol. 4727. Springer, Sept 2007, pp. 81–94, Vienna, Austria.

[17] S. Bhasin, S. Guilley, Y. Souissi, T. Graba, and J.-L. Danger, "Efficient Dual-Rail Implementations in FPGA using Block RAMs," in *ReConFig*. IEEE Computer Society, November 30 – December 2 2011, pp. 261–267, Cancún, Quintana Roo, México. DOI: 10.1109/ReConFig.2011.32.

[18] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal Arrays, Theory and Applications*, ser. Springer series in statistics. New York: Springer, 1999, ISBN 978-0-387-98766-8.

[19] A. Moradi, "Statistical tools flavor side-channel collision attacks," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, 2012, pp. 428–445.

[20] J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," in *CHES*, ser. LNCS, vol. 3156. Springer, 2004, pp. 1–15, Cambridge, MA, USA.