Encoding the State of Integrated Circuits: a Proactive and Reactive Protection against Hardware Trojans Horses

Xuan Thuy Ngo [1] * Sylvain Guilley [1,2] Shivam Bhasin [1]
Jean-Luc Danger [1,2] Zakaria Najm [1]

[1] Institut MINES-TELECOM, TELECOM ParisTech,
CNRS LTCI, "Laboratoire Traitement et Communication de l'Information" (UMR 5141),
Department COMELEC, 46 rue Barrault,
75 634 PARIS Cedex 13, FRANCE.

[2] Secure-IC S.A.S., 80 avenue des Buttes de Coësmes,
35 700 Rennes, FRANCE.

ABSTRACT

Hardware Trojan Horses (HTH) are a serious threat to semiconductor industry with significant economic impact. However, most of the research in HTH focuses on detection. We propose the concept of "encoded circuit", as a technique to protect HTH insertion. Encoded circuit is based on the theory of codes. It encodes the internal state with a chosen code of security parameter d, such that knowledge of less than d bits of the encoded state reveals no information about the actual state. This parameter stems from a similar notion introduced by Ishai, Sahai and Wagner at CRYPTO 2003 for the prevention of probing attacks. Usually $d < 10$ in probing attacks, whereas HTH are able to connect to more than 10 nets. In this paper, we discuss the theory behind "encoded circuits" and its practical demonstration on various HDL circuits.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: SPECIAL PURPOSE AND APPLICATION-BASED SYSTEMS—*Real-time and embedded systems*; C.3 [**Computer Systems Organization**]: SPECIAL PURPOSE AND APPLICATION-BASED SYSTEMS—*Smartcards*

Keywords

Private circuits, encoding, Hardware Trojan Horses (HTH), probing attacks, physical attacks, Quadratic residue (QR) codes, Linear Complementary Dual (LCD) codes, minimal distance.

1. INTRODUCTION

Globalisation and outsourcing have been a trend in the semiconductor industry to overcome increasing production cost. However, this cost-saving has also brought along the problem of Hardware Trojan Horses (HTH). HTH is a malicious module inserted in the original Integrated Circuit (IC) during its design or fabrication process. It can be inserted to perform various malicious task like Denial of Service [1], leakage of sensible data via circuit outputs [1], reduction of the system performance [2], etc. The problem of HTH is very serious for critical application like finance, military, health, etc. Moreover, HTH infected commercial ICs will impact the good will of the company which would lead in reduction of market share. For instance, a HTH recently reported as an alleged backdoor in a military-grade commercial FPGA [3], which allowed tampering with the security features of the FPGA. HTH can be considered more harmful than its software counterpart, because once the IC is fabricated, the HTH cannot be removed. Therefore HTH is currently a hot topic amongst researchers.

The main axis of research on HTH focuses on detection of malicious hardware. A Trojan can be introduced at any design step right from the Register Transfer Level (RTL) source code to lithographic masks fabrication. The detection of HTH is based on several techniques like hardware software co-design [4], reconfigurable logic [5], logic testing [6, 7] and side-channel analysis [8].

The second and the less popular axis of HTH research is prevention techniques. Previous works on HTH prevention are mainly based on code obfuscation techniques [6, 9]. In this paper, we propose the concept of **"encoded circuits"**, a circuit obfuscation method using encoder/decoder to prevent HTH insertions. Encoded circuits are realised by encoding all internal registers (sequential part) of the target design with a *binary code* and followed by addition (XOR) of random masks in a *supplementary code*. Once the sequential part is encoded, the combinatorial part can be easily obfuscated by exploiting the "flatten" option of the netlist synthesis tool. It merges the logic part of encoder/decoder circuit with the combinational part of the target circuit. Thus, the state is totally encoded and the structure of original combimatorial part is totally lost when synthesized together with the enocder and decoder circuit. After encoding, the complexity of the design increases which obfuscates the real functionality of the IC.

Our technique is closely based on the *private circuits* of Ishai, Sahai and Wagner presented at CRYPTO 2003 [10]. Private circuits were proposed to protect against probing attacks which ensures no

*This project has been funded by the French Government, under grant FUI #14 **HOMERE** 959 (**H**ardware tr**O**jans : **M**enaces et robust**E**sse des ci**R**cuits int**E**grés).

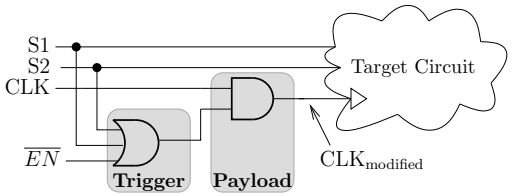


Figure 1: An example of HTH

information leakage with $\leq d$ probes, d being a security parameter. In this paper, we also use a security parameter d, which is the dual distance of code used to encode the IC. It ensures that HTH connected to less than d registers will not be effective. By choosing a bigger d, one can prevent IC from HTH insertion. In this paper, we provide the rationale behind the encoded circuits. We first describe the theoretical background of encoded circuits based on theory of codes. Thereafter, we detail the techniques to choose and generate codes for encoding a circuit. Practical application of encoded circuits is demonstrated on an 8-bit counter, an AES sbox as well as a simple microprocessor. The methodology to encode a simple microprocessor is also discussed. We show that the technique can be applied to any circuit to prevent HTH insertion.

The rest of this paper is structured as follows. Sec. 2 gives the state-of-the-art in HTH prevention and talks about the motivation of our work. Sec. 3 present the rationale behind encoded circuits followed by simple case studies in Sec. 4. Some other aspects of encoded circuits are discussed in Sec. 5. Finally we conclude in Sec. 6, and give some perspectives. For a better readability, technical data appear in appendix.

2. PRELIMINARIES AND CONCEPT

A HTH can be globally seen as a composition of two part:

- **Trigger**: which reads the target circuit state (to trigger its malicious function).
- **Payload**: which writes on the target circuit state (to realise its malicious function).

A basic example explaining the principle of HTH is shown in Fig. 1. The trigger of the HTH is a basic OR gate and the payload is an AND gate. If $S1 = 0$ and $S2 = 0$, the HTH is activated, which disables the circuit clock CLK to stop the target circuit. One can observe from Fig. 1 that to compute the good trigger value, an attacker must have knowledge of the circuit. In a complex circuit, a trigger must be connected to a number of nets (or I/O pads) in order to be controllable and efficient. If the HTH depends on very few signals, then the HTH activation rate increases, which makes it detectable. Therefore HTH act as a "probing station" which is built into the circuit. In [10], Ishai et al. have dealt with a similar problem to propose private circuits It consists in encoding and masking all gates of the sensitive circuit.

It is a known fact, that registers are much easier to observe in an IC layout as compared to other standard gates [11], owing to their bigger size and prominent structure. Thanks to this observation, we propose to apply the concept of "private circuits" only on internal registers. This protection is initially designed to resist the theft of

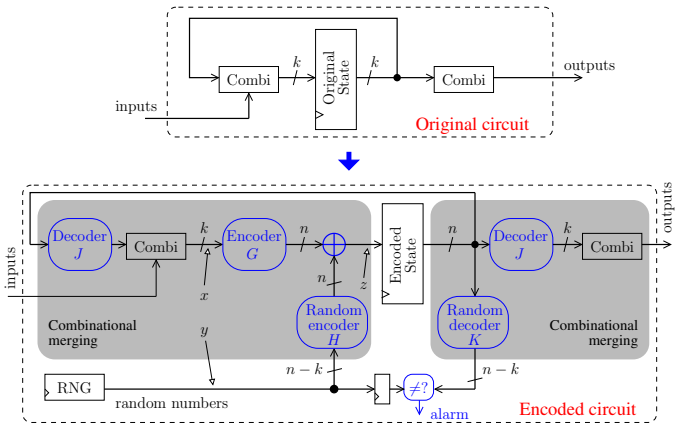


Figure 2: Architecture of "Encoded Circuit", exemplified on a canonical Moore machine.

probed signals. Our encoding method goes beyond, insofar as it shall resist against more connection than a mere probing attack. Another specificity of our protection is that it impedes both the trigger and payload parts of a supposedly inserted HTH.

The basic principle of encoded circuit is summarised in Fig. 2. It shows the transformation of a simple digital circuit to an encoded circuit in order to protect against HTH insertion. The HTH insertion scenario considered in this paper is of an untrusted founder. An untrusted founder (attacker) can derive the original design (with the list of sensitive nets) by reverse engineering the IC layout (GDSII file). Once these sensitive nets and registers are identified, insertion of HTH is straightforward. Using encoded circuit, all registers are encoded and masked. Thus even if the attacker extracts the list of sensitive registers, their dynamic values are randomly encoded which prevents the attacker from inserting HTH. Alternatively, an attacker can design a HTH exploiting sensitive nets only. However, with encoded circuits, we show that combinational nets are modified while re-synthesising and merging parts of the original circuit with the encoder/decoder system. Thus protecting the entire circuit.

3. ENCODED CIRCUITS

In this section, we detail the rationale of encoded circuits. We first describe the basic principle of encoded circuits. Next, we define security objective which determines the choice of codes for encoded circuits. This is followed by some examples and demonstration for construction of codes.

3.1 Basic Principle

The principle behind encoded circuit is very basic. Every IC is composed of two distinct kinds of logic cells: sequential (D-flip-flop or register) and combinational. In practice, it is known that registers are easily recognisable, because they are much larger than combinational gates. For instance, in the 130 nm technology of STMicroelectronics, the size of a D-flip-flop (in short: DFF) varies from 30 to 50 μm^2 while INVERTER (IVLLX05), NOR (NR2LL) and NAND (ND2LL) size are respectively 4.03, 6.05, 6.05 μm^2. A section of a layout is shown in Fig. 3 containing both combinatorial and sequential logic. The yellow blocks in Fig. 3 are DFF or sequential logic and the red blocks are combinatorial gates. We can easily notice that DFF are larger than combinatorial logic gates. DFF gates can be recognised directly in an IC. It can be seen Fig. 4

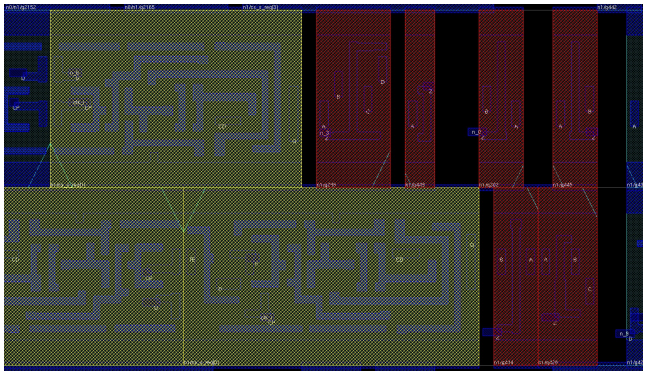


Figure 3: Floorplan of an IC with D-flip-flops (in yellow).

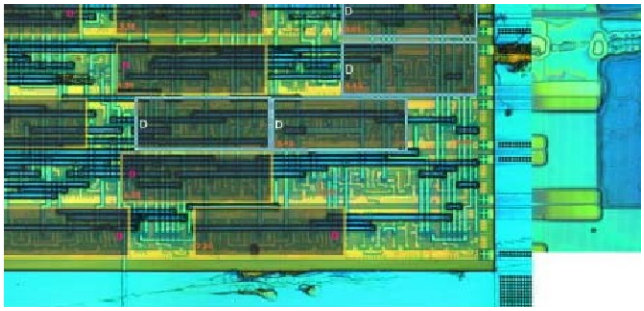


Figure 4: Layout of an ASIC with D-flip-flops [11] (in gray).

that DFF stand out clearly from combinatorial logic. Another motivation for probing the DFF output is that the signals at DFF output are synchronised. Therefore, it is easier for an attacker to insert HTH using the inputs or outputs of these registers as activation conditions. This justifies why we concentrate mainly on the protection of sequential logic. By using the encoding system, we can transform the original data of all sequential logic cells to the encoded data, hence protecting them.

One can encode a circuit, with a state x (set of all sequential resources) of k bits, we need:

- a code C of length n, which is applied on x. For the sake of simplicity, we assume that C is a linear Boolean code.
- some random numbers y of $(n-k)$ bits, which serve as a pool of entropy to mask the encoded state. The masks are also encoded, by a code D, of size $(n-k)$ and dimension n. As a result, the encoded and masked state z is the exclusive-or of one code word of C and D each.

If C and D are supplementary, the encoded x and y can be retrieved from z. We denote by G and H the generator matrices of C and D. It is thus required that the $n \times n$ square matrix $\binom{G}{H}$ be of full rank n.

The decoding logic allows to recover x from z. This operation is also a linear function, that maps elements of $\mathbb{F}_2^n$ to $\mathbb{F}_2^k$ (i.e. not injective). The decoding function generator matrix J has a simple expression if C and D are orthogonal. Indeed, in this case, H is the parity check matrix of code C, i.e., $GH^\mathsf{T} = 0$, or equivalently, $HG^\mathsf{T} = 0$, and there exists an orthogonal projection. It can be checked that $J = G^\mathsf{T}(GG^\mathsf{T})^{-1}$. If $z = xG \oplus yH$, then $zG^\mathsf{T} = xGG^\mathsf{T} \oplus yHG^\mathsf{T} = x(GG^\mathsf{T})$, which simplifies to $zG^\mathsf{T}(GG^\mathsf{T})^{-1} = x$.

Next, we can also check the random numbers which belong to the code D. If the state z is corrupted by some means, that would also impact x and y. Therefore, it is relevant to recover y from z, which can be done by a linear function of generator matrix K. If $D = C^\perp$, there is also an orthogonal projection: $K = H^\mathsf{T}(HH^\mathsf{T})^{-1}$.

The encoding / decoding functions are summarised in Fig. 5. Unless otherwise mentioned, we will assume in the sequel that C and D are orthogonal. We denote supplementary dual codes, such as C, "LCD" (for "Linear Complementary Dual"). This term has been coined by Massey in [12]. Notice that D is LCD if and only if C is also LCD. A good overview of the structure of encoded circuits is illustrated in Fig. 2.

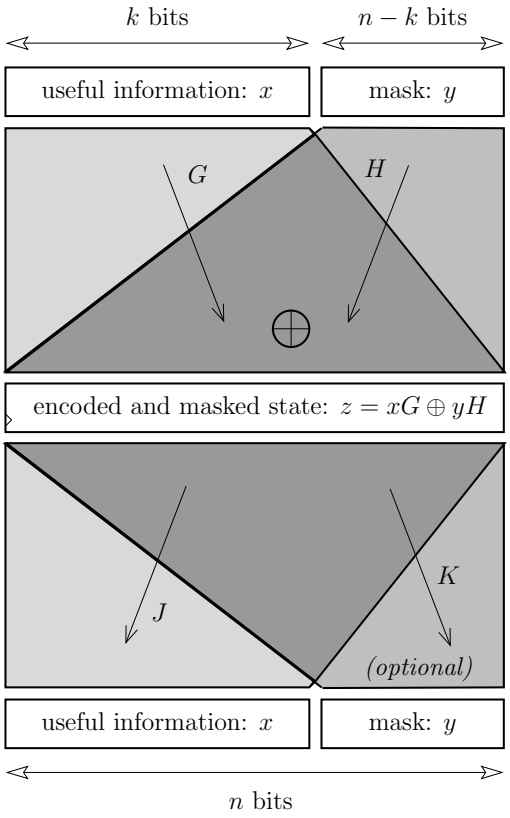


Figure 5: Principle of "circuit encoding"

3.2 Security objective

We intend to apply the notion of private circuits, discussed by Ishai, Sahai and Wagner at CRYPTO 2003 (see [10]). They introduce a security metric d. Private circuits are designed to protect against probing attacks; such that probing $< d$ probes reveals no information about the real data manipulated by the circuit. The complexity of this construction is quadratic in d. For a circuit with k gates, the size of the private circuit is of the order $O(k \cdot (d+1)^2)$.

In our construction, we focus on the encoding of the k registers. A register is easy to protect as its transformation function is "iden-

tity", which is linear. This is why linear codes are suitable in our case, which will further result in lower complexity of the protected circuit. Specifically, we also want to ensure that the knowledge of $d-1$ (or less) bits of the encoded & masked state z does not disclose any information on x. This is feasible, as stated in the following proposition.

Proposition 3.1. *The encoding of x as $z = xG \oplus yH$, where y is a uniformly distributed mask in $\mathbb{F}_2^{n-k}$ does not reveal any information on x provided up to $d-1$ bits of z are known, if and only if C is of minimal distance d.*

Proof. The mask y is applied additively on the encoded state xG as yH. The property that is required is that any tuple of size strictly less than d be balanced. As y is assumed uniformly distributed in $\mathbb{F}_2^{n-k}$, the distribution of any such tuple is unchanged (hence uniform) if and only if the code D is of dual distance d (or more). Now, the dual distance of a code is the minimal distance of its dual [13]. Therefore, as C and D are dual, this is equivalent to require that C has minimal distance d. □

The encoded circuit can also detect a fault injection attack (FIA). The fault can be introduced by a external setup or internal HTH. The detection of faults and protection against FIA is based on the following proposition.

Proposition 3.2. *Let us consider the encoding of x as $z = xG \oplus yH$, where y is a uniformly distributed mask in $\mathbb{F}_2^{n-k}$. Any fault on z of Hamming weight strictly smaller than d can be detected.*

Proof. The state z is modified into $z \oplus \varepsilon$, for some random $\varepsilon \in \mathbb{F}_2^n$. By supplementarity of C and D, there exists a unique ordered pair $(e, f) \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$ such that $\varepsilon = eG \oplus fH$. A detection strategy consists in checking whether or not the mask has been altered, i.e., $zK \stackrel{?}{=} y$. This verification does not jeopardise the security model of Prop. 3.1 since x is not uncovered, only y. By linearity of the fault injection, the equality $(z \oplus \varepsilon)K = y$ happens if and only if $\varepsilon K = 0 \iff f = 0$, i.e., $\varepsilon \in C$. As $\varepsilon = 0$ is pointless (since without observable effect), harmful (since undetected) faults only happen if and only if $\varepsilon \in C \setminus \{0\}$. In particular, a necessary condition for the fault to be undetected is that the Hamming weight of ε be greater than or equal to the minimal distance d of code C. □

Now, given that the minimal distance d of C is a security parameter, it is set as high as possible. Therefore, choosing a C LCD of greatest possible minimal distance simultaneously improves the resistance against HTH insertion and FIA. One specificity of our protection is that we expect to have the security parameter d quite large, for instance $d \approx 32$. Indeed, our protection aims at preventing the attacker from inserting small HTH, that are stealthy, hence difficult to detect if inserted. However, a HTH that would connect to 32 DFFs (or more) would be easy to detect by various means, as discussed in the introduction, such as side-channel, visual inspection [14], etc.

We notice that a value of d around 32 is larger that the values d used in [10], where we recall that the objective is to refrain an attacker from probing the circuit. Usually, probing is delicate, and starting from a few probe tips (especially very close one from each other), it begins to be difficult to contact the wires without making short circuits. Therefore, our focus on registers is also motivated by the potential increase of security parameter d.

3.3 Encoding system examples

In this subsection, we show a small example of LCD code with $k = 8$. We consider a code C of parameters $[16, 8, 5]$, where 5 is the minimal dual-distance which is the security parameter d. It has shorter length than the shortest QR LCD code (a $[17, 9, 5]$ code) with a dimension greater or equal to $k = 8$. The matrices G, H, and J are given in Appendix A.

Such a family of codes prevent insertion of HTH that would connect to $d-1$ or 4 registers. For instance, it would deny the insertion of HTH such as our initial example (Fig. 1). But we concur that $d = 5$ is obviously a very small security parameter: most HTH would connect to more DFFs, in order make the triggering condition more furtive. Thus, we would prefer $d \approx 32$, for instance.

There are two classical solutions to increase the code minimal distance:

1. At fixed k, increase n. However, this is not really scalable for small k, such as $k = 8$. Indeed, by the Singleton bound, a linear code of parameters $[n, k, d]$ satisfies $n \geq k + d - 1$.

2. At fixed d, increase k (which will cause n to grow too). It is possible to increase k by considering not only small groups of registers, e.g. bytes, but by merging registers. For instance, in a CPU, several registers can be masked altogether. This allows to grow k. In the sequel, we will assume that k is actually the number of all the registers of the circuit. So, if $k \gg d$, the Singleton bound is more favourable.

In the next subsection, we exhibit constructs that allow reaching large values of d. (Notice however that our solution is far from the Singleton bound: n is closer to $2k$ than to $k + d - 1$; but there is venue for finding LCD codes with larger rates k/n.)

3.4 Real constructions of LCD codes of large dimension

An important selection criterion is the existence of a bound on the minimal distance, that otherwise cannot be computed by testing all the possible Hamming distances between pairs of different code words since our codes can have lengths of the order of one to several hundreds [15].

For large dimensions k, there is no precomputed table (for instance, the tables of Grassl [16] are provided for $k \leq 256$ for binary codes). Fortunately, cyclic codes have a minoration on their minimal distance, via the BCH bound. The definition of a cyclic code is recalled below:

Definition 3.1 (Cyclic code). *A linear code C of length n over a finite field $\mathbb{F}_q$ is cyclic if it is stable by any circular rotation.*

In the sequel, we consider binary codes ($q = 2$). The condition for being LCD is rather simple and not difficult to achieve. Moreover, a potentially stronger lower bound on the minimum distance exists

for the sub-class of quadratic-residue (QR) codes[1], which can also be LCD.

We give here, for large k, our results based on QR codes, that is one option that works, but certainly not the best. But as we wanted to exhibit a proof-of-concept (PoC), this is good enough to serve our purpose. A QR code has for length a prime number and has a minimal distance d at least $\sqrt{n}$. A binary QR code has length n congruent with ± 1 modulo 8 and is LCD if the length is congruent with 1 modulo 8 [13].

We implemented the constructions of codes in SAGE. We proceed in two steps: First algorithm (see listing. 1) finds the minimal length n suitable for a QR LCD code. A second algorithm (see listing. 2) computes the four generating matrices G, H, J and K.

```
import math, itertools

def is_prime(n):
  """ Checks whether n is prime or not """
  if n % 2 == 0 and n > 2:
    return False
  return all(n % i for i in \
    range(3, int(math.sqrt(n)) + 1, 2))

def get_length(k):
  for dim in itertools.count( k, 1 ):
    for pm1 in [-1, +1]:
      n = 2*dim + pm1
      if is_prime(n) and (n%8) == 1:
        print "Parameters␣of␣the␣"+\
          "shortest␣QR␣code␣for␣dimension␣"+\
          ">=␣{k}:␣[{n},{dim},>={d}]".format(
            n=n,
            k=k,
            dim=dim,
            d=int(math.ceil(math.sqrt(n))) )
        return n
```

Listing 1: Search for the parameters of a QR LCD code

```
def gen_matrices(n):
  C = QuadraticResidueCode(n, GF(2))
  D = C.dual_code()

  G = C.gen_mat()
  H = D.gen_mat()
  J = G.transpose() * (G * G.transpose())^-1
  K = H.transpose() * (H * H.transpose())^-1

  # Testing supplementarity and duality
  # of codes C & D (valid by design)
  if rank(block_matrix([[G],[H]]))!=G.ncols()\
    or G * H.transpose()!=0:
    raise("Logic␣error:␣The␣code␣is␣not␣LCD")

  return [ G, H, J, K ]
```

Listing 2: Generation of matrices G, H, J, K

[1]Not to be confused with the two-dimensional bar code, also nicknamed "QR code" (abbreviated from Quick Response Code).

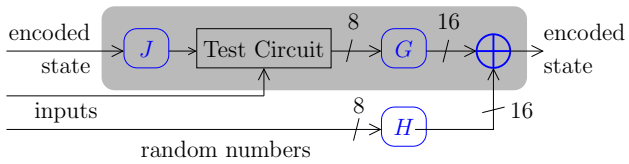


Figure 6: Case study of an 8-bit counter and an 8×8 Sbox

It can be seen that the code obtained with listing 2 has dimension greater than k. But in practice, this dimension is only slightly larger than k (a few more bits, if any). Those can, for instance, be fed as constant 0 along with the k bits that come from the state register. Or, still better, they can be fed by random numbers, which contributes to the overall noise of the countermeasure.

4. CASE STUDIES

The rationale behind encoded circuits was discussed in the previous sections. In this section, we show practical application of encoded circuits on some basic designs. As previously shown, the sequential part of a circuit can be well encoded using a LCD code. Nevertheless, it is still possible to insert a HTH. An attacker that can isolate all blocks of an encoded IC (i.e., combinational part, encoder data G, encoder noise H and decoder data J), can bypass the prevention by inserting a HTH which probes directly at the inputs of encoder block (or at the output of decoder block, etc.). This is all the more possible as the IC is synthesised (i.e., generated) hierarchically. Of course, if the combinational logic is linear (of $k \times k$ matrix L), then the encoded logic $z \mapsto zJLG$ completely dissolves L into a new linear transformation of $n \times n$ matrix $L' = JLG$. We demonstrate in this section that using a *flattening* option for the netlist synthesis, we manage to merge these blocks together (combinational part, encoder for data, encoder for noise and decoder for data). Therefore it becomes a challenge for an attacker to reverse the real functionality of the IC. In the rest, we firstly show that the netlist synthesis tool merges and optimises these blocks on a simple counter circuit and a highly non-linear circuit. The non-linear circuit is the Substitution Box of the Advanced Encryption Standard (AES).

4.1 Case Study I: 8-bit counter

For the first case, the goal is to demonstrate that the combinational logic part of a counter circuit will merge with logic parts of the encoder/decoder circuit. This will result in protection of the combinational logic. To demonstrate the principle, we choose a simple 8-bit counter as test circuit written in VHDL.

The encoded circuit is composed of 5 different parts:

1. **Test** circuit: 8-bit counter.
2. **Encoder data** circuit: matrix G of size 8×16 (Eq. (1)) is used to encode data.
3. **Encoder noise** circuit: matrix H of size 8×16 (Eq. (2)) is used to mask the encoded data.
4. **Decoder data** circuit: matrix $J = G^T(GG^T)^{-1}$ of size 16×8 (Eq. (3)) is used to decode data.
5. **XOR gates**: n two-input XORs gates used to add noise to encoded data.

Table 1: Synthesis results for the encoded 8-bit counter

Hierarchical synthesis

Gate	Instances	Area	Library
AO2LL	5	50.430	CORE9GPLL
AO7LL	8	64.550	CORE9GPLL
EN3LL	2	56.482	CORE9GPLL
EO3LL	17	480.094	CORE9GPLL
HA1LL	6	108.929	CORE9GPLL
IVLLX05	21	84.722	CORE9GPLL
MUX21NLL	39	472.025	CORE9GPLL
ND2LL	8	48.413	CORE9GPLL
NR2ALL	6	48.413	CORE9GPLL
NR2LL	2	12.103	CORE9GPLL
total	114	1426.160	

Type	Instances	Area	Area %
inverter	21	84.722	5.9
logic	93	1341.438	94.1
total	114	1426.160	100.0

Flatten synthesis

Gate	Instances	Area	Library
AO1LL	3	30.258	CORE9GPLL
AO2LL	9	90.774	CORE9GPLL
AO3LL	5	50.430	CORE9GPLL
AO7LL	12	96.826	CORE9GPLL
EN3LL	3	84.722	CORE9GPLL
EO3LL	14	395.371	CORE9GPLL
IVLLX05	22	88.757	CORE9GPLL
MUX21NLL	37	447.818	CORE9GPLL
ND2LL	18	108.929	CORE9GPLL
NR2LL	3	18.155	CORE9GPLL
total	126	1412.040	

Type	Instances	Area	Area %
inverter	22	88.757	6.3
logic	104	1323.283	93.7
total	126	1412.040	100.0

The choice of G, H and J are presented in App. A. In order to evaluate the proof-of-concept of coding circuit, we compare the encoded circuit with **Hierarchical** synthesis against **Flattened** synthesis. The technology used for the case study is STMicroelectronics CMOS 130 nm process. The synthesis is done using Cadence Encounter RTL Compiler. The synthesis result are presented in Tab. 1.

From the synthesis result, we can make the following observations:

- There are some standard cells which have disappeared when we flatten the design, for example arithmetic gates (HALL, that are Half-Adders).
- The type and the number of others standard cells as AND, NAND, OR, MUX and INVERTER are also different between the two synthesis (114 for hierarchical synthesis and 126 for flatten synthesis).
- The area taken by hierarchical result is bigger than flatten result (1412 μm^2 *vs* 1426 μm^2).

Therefore we can conclude that Encounter Compiler merged the logic part of encoded circuit using the "flatten" option, making it difficult for the attacker to reverse-engineer the design from netlist and insert a meaningful HTH.

4.2 Case Study II: Substitution box

Next we repeat the same experiment with a highly non-linear circuit. The non-linear circuit used is the 8×8 Substitution Box (Sbox) of the AES.

The results are presented in Tab. 3 (App. B). Like before, the flattened netlist has different cell types. However, the area taken by hierarchical synthesis result is smaller than flatten synthesis result. This is because, in this case, the synthesizer has used bigger cells in the flattened netlist. The problem can be easily solved by constraining the synthesizers. Hence our method is applicable also on non-linear circuits.

4.3 Case Study III: Nanoprocessor

After applying the encoding method on two basic circuits, we also applied the technique on a simple microprocessor. We choose nanoprocessor [17], which is a 8-bit processor without pipeline and requires 3 clock cycle to execute every instruction. It has 16 basic instructions, and operates using an external 256 bytes memory.

The unprotected nanoprocessor gives the following after synthesis:

- 37 sequential cells (flip-flops),
- 208 combinational cells.

Thus we have $k \geq 37$ for the nanoprocessor netlist. After searching various LCD codes suitable to our case, we found a Quadratic Residue (QR) code **[73,37,13]** i.e. $k = 37$, $n = 73$ [18]. Since $k = 37$ equals number of flip-flops in nanoprocessor, this is the smallest code which can be applied. The minimal distance or the security parameter d of this code is 13, i.e. an attacker should connect to at least 13 DFF to implement an effective HTH. To achieve a larger minimal distance or security, the dimension must be increased. We found another QR code which is a LCD of dimensions

[89,45,17] and a shortened code derive from it i.e. **[86,42,17]** [12]. Both these codes will result in a better protection at the cost of chip area. The process to apply the three codes is exactly the same.

4.3.1 *Encoding Process*

The encoding is done as following:

- We start with the flat netlist of nanoprocessor and extract the 37 flip-flops with their initial reset value. We make sure that these flip-flops have no logic on the clock and reset path.
- For the remaining combinational part of the circuit, we add a `from_seq` input and a `to_seq` output bus (of bitwidth k).
- Next for the given k, we find a suitable code to find n (see function `get_length` in Listing 1), and dump the behavioural code for those four matrix operations G, H, J *and* K (see function `get_matrices` in Listing 2).
- Next we connect the extracted combinatorial part of nanoprocessor with the HDL of matrices G, H, J, K and RNG as shown in Fig. 2 and keep the hierarchy.
- Thereafter k flip-flops in the uncoded state are replaced by n flip-flops in the encoded state, keeping the equivalent state at reset.
- At this stage, we combine the modified combinatorial and sequential part of the circuit using a wrapper circuit.
- Finally, we synthesise, place and route the encoded circuit.

This encoded method can be easily automated and integrated into a standard EDA flow. The result of synthesis for the encoded nanoprocessor is presented in Tab. 2. This table shows the total sequential gates, combinational gates, the area and security parameter d. Results are shown for the three QR codes. We can notice that the number of sequential gates increased from 37 to 73/86/89. It is logical because we encoded k flip-flops into n. The combinational logic part also increased due to the integration of G, H, K *and* J matrices. Pre-synthesis and post-synthesis simulations are performed to ensure that the encoded processor works correctly.

5. DISCUSSION

The encoder and the decoder blocks are product of matrices. As an example the computation of encoder matrix product $c = xG$ is detailed in Alg. 1.

Algorithm 1 Vector–matrix product.

Require: $x \in \mathbb{F}_2^k$ and $G \in (\mathbb{F}_2^n)^k$, whose rows are denoted by $G[i] \in \mathbb{F}_2^n$ (for $1 \leq i \leq k$)
Ensure: $c = xG \in \mathbb{F}_2^n$
1: $c \leftarrow 0$ (a vector of n bits [i.e., a *line*, as opposed to a *column*])
2: **for** $i \in [\![1, k]\!]$ **do**
3: $\quad c \leftarrow c \oplus x[i] \wedge G[i]$ ▷ $\begin{cases} 0 \wedge G[i] = 0 \in \mathbb{F}_2^n \text{ and} \\ 1 \wedge G[i] = G[i] \end{cases}$
4: **end for**
5: **return** c

For the nanoprocessor, we use different codes with $n = 73/86/89$ and $k = 37/42/45$ for a sequential information on 37 bits. In the latter two cases, n is much greater than k. If we choose a coding system where n is closer to k (i.e. $n = 73$), we can reduce significantly the overhead, but this is at the expense of robustness as security parameter d is much lower according to the Singleton bound ($n - k \geq d - 1$). This brings us to a area-security trade-off which is a common issue in security engineering. Using another encoding system where G, H and J matrices are sparse, we could reduce significantly the overhead of this method. Some studies should be conducted to reduce the encoding complexity. For instance we can consider sparse matrices as for low-density parity check (LDPC) codes.

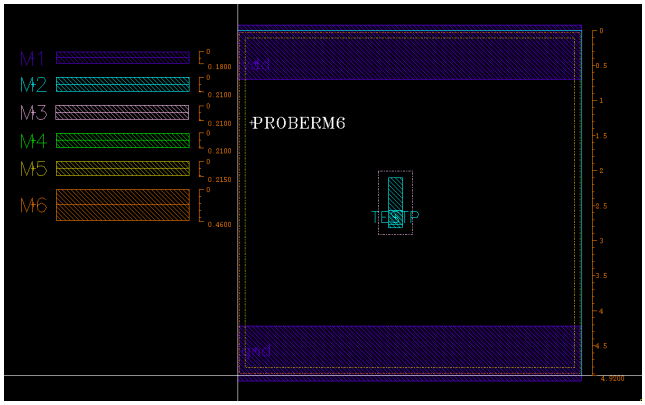


Figure 7: Comparison between minimum sizes of the metal 1–6 wires and of the `PROBERM6` cell from the `CORX9GPLL` standard cell library of STMicroelectronics `HCMOS9GP` technology.

5.1 Protection Against Other Physical Attacks

Encoded circuits are mainly proposed to counter the problem of HTH insertion. Moreover, these circuits can find some other interesting applications in the field of physical security. As encoded circuits are based on private circuits, they directly address the threat of **probing attack**. Probing attacks use tiny probes to monitor the inputs/outputs of internal blocks to directly recover sensible data. By encoding all internal sequential logic part and by masking encoded data with random numbers, this prevention method can also protect IC against probing attack with less than $d - 1$ probes.

Some professional equipment (such as those proposed by GigaTest Labs, Lake Shore Cryotronics, Inc., Micromanipulator, KeithLink, J microTechnology, Inc., etc.) allow the adversary to place dozen of probes on a chip or a wafer. However, probing attacks are not easy to mount. The circuits are designed to be probed this way, using for instance "prober" pads (see Fig. 7 on the right). In order to attack our protection, an attacker must probe inner metal lines, which are much smaller than "prober" pads. For instance, the width of a metal 6 (uppermost layer) wire in a 0.13 μm technology (`HCMOS9GP` from STMicroelectronics) is about one-tenth of a "prober" pad, as sketched in Fig. 7. Also "prober" pads are not usually found in a critical circuits. Second, the probes are placed far from each other while the lines to probe might be located in great proximity of each other, thereby making any probing attempt very chancy.

Encoded circuits can also be potentially used to protect against **Side Channel Attack** (SCA). SCA is a passive but powerful attack which observes unintentional physical leakage to extract the secret key of cryptographic block. By using a LCD code with dual distance d, encoded circuit can be protected against monovariate

Table 2: Synthesis results of Nanoprocessor encoded circuit method, and security parameter.

IC (Code)	Sequential gates k	Combinational	Area (μm^2)	Security parameter d
Original (-)	37	199	1181	1
Encoded ([73,37,13])	73	1001	6926	13
Encoded ([86,42,17])	86	1410	9717	17
Encoded ([89,45,17])	89	1754	11296	17

SCA of orders 1, 2, to $d-1$. And the monovariate SCA of order d is the lowest degree attack to be practical [19].

As stated earlier, encoded circuits can also be used to detect **Fault Injection Attacks** (FIA). This can be done, thanks to the random numbers injected to mask the encoded circuit. Precisely, when we decode the encoded data, we can verify the random numbers used to mask circuit using the matrix K. A simple comparator, as shown in Fig. 2, is inserted to check whether the output random number zK are same as y i.e. one used to mask. If they differ, an "alarm" signal will be set to indicate that there has been a fault in the encoded registers. When this "alarm" is high, the circuit can launch a recovery mechanism, for instance to stop its operations, hence preventing an attacker from effectively exploiting faulted outputs. Practical robustness of encoded circuits against SCA, FIA and probing will be rigoursly dealt in future works.

5.2 Difference from Private Circuits

The proposed encoded circuits method prevents HTH insertion at two different levels. First, like private circuits [10], it prevents the HTH from retrieving any sensitive data by eavesdropping $< d$ flip-flops. This protection impedes the insertion of HTH *trigger part*. Moreover, for the HTH *payload part*, the proposed countermeasure also brings another aspect of active HTH detection. If some how the HTH is able to write a malicious value into the state, the encoded state can be checked for errors introduced by the HTH. For a HTH to be functional, its payload must also be encoded with the same code as the original circuit (i.e. C and D matrix).

5.3 Comparison with Previous Works

Preventing HTH insertion by encoding internal variables of a circuit have been partially dealt in few previous works. Chakraborty et al. [9] initially presented a prevention method which obfuscates only the state machine of the IC. It is inspired by obfuscation methods [20, 21] initially intended to protect against IC counterfeiting. It partition the states into: an original state space and an isolation state space. The original state space can only be reached using a specific input pattern (ex. secret key). If a wrong input pattern is presented at the input, the IC locks itself in a non-reversible isolation state space. Presented technique protects only the control part, while the data-sensitive part remains attackable. Instead, encoded circuits protect both parts (control and data). Moreover in [9], when the IC is well configured to reach the original state, it operates normally and cannot resist others physical attacks. Using encoded circuits, we can theoretically not only protect against HTH insertion attack but also against others physical attacks because of the use of random numbers.

Another prevention method, ODETTE [6], is more intended to raise the HTH activity for a better detectability than a proactive prevention. Furthermore, each bit of the state is masked with one bit of secret. With our method, we provide a more flexible solution, where the number of "mask" bits can be chosen, thus allowing the designer to adjust the security level. In [22], authors propose the method named "EPIC" which encodes the combinational logic part whereas in our encoded circuit method, the sequential logic part is encoded. EPIC is based on "security by obscurity" hence probing can be done after configuration to recover the key. This EPIC method is static therefore an attacker can create a HTH which learns the key and subsequently gets activated, hence bypassing the EPIC method. Whereas, our "encoded circuit method" is dynamic because the circuit is encoded with a random mask hence avoiding key learning attacks.

6. CONCLUSION AND PERSPECTIVES

HTH insertion has become a serious issue with the globalization of semiconductor industry. In this paper, we propose "encoded circuits" to prevent the HTHs insertion. It is based on a quantifiable security metric d, similar to the one introduced for probing by Ishai, Sahai and Wagner at CRYPTO 2003, but adapted to HTH insertion prevention. Here d defines the minimum number of connections required to insert an effective HTH.

We studied the theory of codes and its rationale in "encoded circuits". Practical application from simple circuits like a counter to a full microprocessor are demonstrated. In this proof of concept, the area overhead for encoding a microprocessor was $> 6\times$. This obfuscation method can also potentially prevent against side-channel attacks, fault injection attacks and probing attacks.

As a future work, better codes are to be found for increasing the security (minimal distance d) or decreasing the complexity. This implies two research directions (that can be combined). First of all, shorter codes (with smaller length n) could be imagined. For instance, such codes could result from avoiding the condition that C and D are dual. But in this case, the security property (see Proposition 3.1) would be different, and should be rederived. Also, the computation of the matrices J and K would be different. Second, long codes could be kept, but be chosen with sparse matrices. One possibility are low density parity check (LDPC) codes.

7. REFERENCES

[1] Yier Jin, Nathan Kupp, and Yiorgos Makris. Experiences in hardware trojan design and implementation. In *Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, HST '09, pages 50–57, Washington, DC, USA, 2009. IEEE Computer Soc.

[2] Francis G. Wolff, Christos A. Papachristou, Swarup Bhunia, and Rajat Subhra Chakraborty. Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. In *DATE* [23], pages 1362–1365.

[3] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *CHES*, September 9-12 2012. Leuven, Belgium.

[4] Gedare Bloom, Bhagirath Narahari, and Rahul Simha. OS Support for Detecting Trojan Circuit Attacks. In Mohammad

Tehranipoor and Jim Plusquellic, editors, *HOST*, pages 100–103. IEEE Computer Society, 2009.

[5] Miron Abramovici and Paul Bradley. Integrated circuit security: new threats and solutions. In Frederick T. Sheldon, Greg Peterson, Axel W. Krings, Robert K. Abercrombie, and Ali Mili, editors, *CSIIRW*, page 55. ACM, 2009.

[6] M. Banga and M. S. Hsiao. ODETTE : A Non-Scan Design-for-Test Methodology for Trojan Detection in ICs. In *International Workshop on Hardware-Oriented Security and Trust (HOST), IEEE*, pages 18–23, 2011.

[7] Susmit Jha and Sumit Kumar Jha. Randomization Based Probabilistic Approach to Detect Trojan Circuits. In *Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, HASE '08, pages 117–124, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan Detection using IC Fingerprinting. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 296–310, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Rajat Subhra Chakraborty and Swarup Bhunia. Security against hardware trojan through a novel application of design obfuscation. In *ICCAD, IEEE*, pages 113–116, 2009.

[10] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, volume 2729 of *LNCS*, pages 463–481. Springer, August 17–21 2003. Santa Barbara, California, USA.

[11] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann. Cryptanalysis of the DECT Standard Cipher. In *FSE*, volume 6147 of *LNCS*, pages 1–18. Springer, February 7-10 2010.

[12] James L. Massey. Linear codes with complementary duals. *Discrete Mathematics*, 106-107:337–342, 1992.

[13] F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier, Amsterdam, North Holland, 1977. ISBN: 978-0-444-85193-2.

[14] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Thuy Ngo, and Laurent Sauvage. Hardware Trojan Horses in Cryptographic IP Cores. In *FDTC*, pages 15–29, August 20 2013. Santa Barbara, CA, USA.

[15] Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

[16] Markus Grassl. Bounds on the minimum distance of linear codes and quantum codes. Online available at http://www.codetables.de/, 2007.

[17] M.J. Wirthlin, B.L. Hutchings, and K.L. Gilson. The Nano Processor: a low resource reconfigurable processor. In *FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on*, pages 23–30, Apr 1994.

[18] Claude Carlet and Sylvain Guilley. Complementary Dual Codes for Counter-measures to Side-Channel Attacks. In Springer, editor, *ICMCTA, 4th International Castle Meeting on Coding Theory and Applications*, CIM-MS, September 15-18 2014. Palmela, Portugal. URL: http://icmcta.web.ua.pt. (article #9). ISBN 978-3-319-17295-8. http://www.springer.com/978-3-319-17295-8.

[19] Shivam Bhasin, Wei He, Sylvain Guilley, and Jean-Luc Danger. Exploiting FPGA block memories for protected cryptographic implementations. In *ReCoSoC*, pages 1–8. IEEE, 2013.

[20] Yousra M. Alkabani and Farinaz Koushanfar. Active hardware metering for intellectual property protection and security. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 20:1–20:16, Berkeley, CA, USA, 2007.

[21] Rajat Subhra Chakraborty and Swarup Bhunia. Hardware protection and authentication through netlist level obfuscation. In *IEEE/ACM Int'l Conf. on Computer-Aided Design*, ICCAD '08, pages 674–677, Piscataway, NJ, USA.

[22] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *DATE* [23], pages 1069–1074.

[23] *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*. IEEE, 2008.

APPENDIX

A. EXAMPLE OF CODE GENERATOR MATRICES

The generator matrices G and H for codes C and D are given in Eq. (1) and (2). The matrices J and K to recover x and y from $z = xG \oplus yH$ are in Eq. (3) and (4) respectively.

$$G = \left(\begin{array}{cccccccc|cccccccc} 0&1&0&0&1&0&0&1 & 1&0&1&0&1&0&1&1 \\ 1&0&1&0&0&1&0&1 & 0&1&1&1&0&1&1&1 \\ 1&1&0&1&0&0&0&0 & 1&0&1&1&0&1&1&1 \\ 1&1&1&0&1&0&1&0 & 0&1&0&1&0&1&1&1 \\ 1&1&0&0&0&0&0&1 & 0&0&0&0&0&1&1&0 \\ 1&0&0&0&0&1&0&1 & 1&0&1&1&1&0&1&1 \\ 1&0&0&1&1&0&1&0 & 1&0&1&1&0&0&1&0 \\ 1&0&0&0&1&1&0&0 & 0&1&0&1&1&1&1&1 \end{array}\right) \quad (1)$$

$$H = \left(\begin{array}{cccccccc|cccccccc} 1&0&1&0&0&1&0&1 & 1&0&0&0&0&0&0&0 \\ 0&1&1&1&0&1&1&1 & 0&1&0&0&0&0&0&0 \\ 0&0&0&1&1&1&1&0 & 0&0&1&0&0&0&0&0 \\ 1&0&0&0&1&1&1&1 & 0&0&0&1&0&0&0&0 \\ 1&1&1&0&0&0&1&0 & 0&0&0&0&1&0&0&0 \\ 1&1&1&1&0&0&0&1 & 0&0&0&0&0&1&0&0 \\ 1&1&0&1&1&1&0&1 & 0&0&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&1 & 0&0&0&0&0&0&0&1 \end{array}\right) \quad (2)$$

$$J = \left(\begin{array}{cccccccc} 1&0&0&1&0&0&1&0 \\ 0&0&1&1&0&1&1&0 \\ 1&1&0&0&0&0&1&1 \\ 0&1&0&0&0&1&0&1 \\ 1&1&0&0&1&1&1&1 \\ 1&0&0&0&0&0&1&1 \\ 1&0&0&1&0&0&0&1 \\ 0&0&1&0&0&0&1&1 \\ \hline 1&1&1&1&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \\ 1&0&0&1&1&0&0&0 \\ 0&1&1&0&1&1&0&0 \\ 1&1&1&1&0&1&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&1&1&1&0 \\ 0&1&0&0&1&0&1&1 \end{array}\right) \quad (3)$$

$$K = \left(\begin{array}{cccccccc} 0&1&0&0&1&1&1&0 \\ 1&1&1&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1 \\ 1&0&0&1&0&0&1&1 \\ 1&0&0&0&1&1&0&0 \\ 0&1&0&0&0&1&1&0 \\ 1&0&1&0&0&0&1&1 \\ 0&1&0&1&1&0&1&0 \\ \hline 1&1&1&0&0&0&1&1 \\ 1&0&1&1&0&1&0&0 \\ 1&1&0&1&1&0&1&0 \\ 0&1&1&0&1&1&0&1 \\ 0&0&1&1&1&1&0&1 \\ 0&1&0&1&1&0&1&1 \\ 1&0&1&0&0&1&1&0 \\ 1&0&0&1&1&1&0&1 \end{array}\right) \quad (4)$$

B. SYNTHESIS RESULTS FOR CASE STUDY II

Tab. 3 compares the result of hierarchical and flatten syntheses when protecting an AES Sbox using the proposed encoder/decoder scheme.

Table 3: Synthesis results for encoded Sbox

Hierarchical synthesis

Gate	Instances	Area	Library
AO10LL	5	70.602	CORE9GPLL
AO10NLL	7	98.843	CORE9GPLL
AO11LL	9	127.084	CORE9GPLL
AO14LL	1	14.120	CORE9GPLL
AO17LL	9	108.929	CORE9GPLL
AO1CLL	2	28.241	CORE9GPLL
AO1LL	8	80.688	CORE9GPLL
AO1NLL	2	28.241	CORE9GPLL
AO20ALL	2	24.206	CORE9GPLL
AO21LL	2	20.172	CORE9GPLL
AO23LL	4	56.482	CORE9GPLL
AO23NLL	2	32.275	CORE9GPLL
AO2ALL	2	24.206	CORE9GPLL
AO2LL	31	312.666	CORE9GPLL
AO35LL	1	12.103	CORE9GPLL
AO3CLL	1	14.120	CORE9GPLL
AO3LL	4	40.344	CORE9GPLL
AO3NLL	1	14.120	CORE9GPLL
AO4LL	10	121.032	CORE9GPLL
AO52LL	15	181.548	CORE9GPLL
AO52NLL	1	14.120	CORE9GPLL
AO6LL	5	50.430	CORE9GPLL
AO6NLL	9	90.774	CORE9GPLL
AO7LL	38	306.614	CORE9GPLL
AO7NLL	4	40.344	CORE9GPLL
AO8LL	1	10.086	CORE9GPLL
AO9LL	1	12.103	CORE9GPLL
EN3LL	2	56.482	CORE9GPLL
EO3LL	17	480.094	CORE9GPLL
IVLLX05	54	217.858	CORE9GPLL
MUX21NLL	36	435.715	CORE9GPLL
ND2ALL	8	80.688	CORE9GPLL
ND2LL	60	363.096	CORE9GPLL
ND3ABLL	2	20.172	CORE9GPLL
ND3LL	3	24.206	CORE9GPLL
ND4ABLL	1	14.120	CORE9GPLL
ND4LL	16	161.376	CORE9GPLL
NR2ALL	13	104.894	CORE9GPLL
NR2LL	47	284.425	CORE9GPLL
NR3ABLL	3	30.258	CORE9GPLL
NR3LL	6	48.413	CORE9GPLL
NR4ABCLL	5	60.516	CORE9GPLL
NR4ABLL	6	72.619	CORE9GPLL
NR4ALL	5	60.516	CORE9GPLL
NR4LL	24	242.064	CORE9GPLL
total	485	4692.007	

Type	Instances	Area	Area %
inverter	54	217.858	4.6
logic	431	4474.150	95.4
total	485	4692.007	100.0

Flatten synthesis

Gate	Instances	Area	Library
AN2LL	1	10.086	CORE9GPLL
AO2LL	5	50.430	CORE9GPLL
AO7LL	9	72.619	CORE9GPLL
EO3LL	15	423.612	CORE9GPLL
IVLLX05	29	116.998	CORE9GPLL
MUX21NLL	56	677.779	CORE9GPLL
ND2LL	48	290.477	CORE9GPLL
ND3ABLL	22	221.892	CORE9GPLL
ND3LL	2	16.138	CORE9GPLL
ND4ABCLL	1	14.120	CORE9GPLL
ND4LL	19	191.634	CORE9GPLL
NR2ALL	9	72.619	CORE9GPLL
NR2LL	271	1639.984	CORE9GPLL
NR3ALL	3	36.310	CORE9GPLL
NR3LL	17	137.170	CORE9GPLL
NR4ABCLL	14	169.445	CORE9GPLL
NR4ABLL	8	96.826	CORE9GPLL
NR4ALL	69	835.121	CORE9GPLL
NR4LL	48	484.128	CORE9GPLL
total	646	5557.386	

Type	Instances	Area	Area %
inverter	29	116.998	2.1
logic	617	5440.388	97.9
total	646	5557.386	100.0