A formal study of two physical countermeasures against side channel attacks

Sébastien Briais[1], Sylvain Guilley[2], and Jean-Luc Danger[2]

[1] Secure-IC
sebastien.briais@secure-ic.com
[2] Telecom Paristech
sylvain.guilley@telecom-paristech.fr,
jean-luc.danger@telecom-paristech.fr

Abstract. Secure electronic circuits must implement countermeasures against a wide range of attacks. Often, the protection against side channel attacks requires to be tightly integrated within the functionality to be protected. It is now part of the designer's job to implement them. But this task is known to be error-prone, and with current development processes, countermeasures are evaluated often very late (at circuit fabrication).
In order to improve the confidence of the designer in the efficiency of the countermeasure, we suggest in this article to resort to formal methods early in the design flow for two reasons. First of all, we intend to check that the process of transformation of the design from the vulnerable description to the protected one does not alter the functionality. Second, we wish to prove that the security properties (that can derive from a formal security functional specification) are indeed met after transformation. Our first contribution is to show how such a framework can be setup (in COQ) for netlist-level protections. The second contribution is to illustrate that this framework indeed allows to detect vulnerabilities in dual-rail logics.

1 Introduction

More and more electronic circuits are entrusted with security functions. In particular, they must make sure the information they process, that can be sensitive, is well kept secret. For this reason, electronic circuits must be prepared to be attacked. Thus, it is important that they are properly protected against a wide range of attacks, in particular against side-channel attacks. In practice, to thwart those attacks, extra logic is required: its role is to mask the sensitive data or balance the leakage. From a design point of view, the countermeasure is either coded manually, or implemented automatically by a tool.

In both cases, it would be relevant to ascertain that the functionality remains unchanged after the application of the countermeasure, and that the countermeasure is implemented as intended. But currently, these verifications are seldom carried out: mostly, real attacks are tried after the product is produced, without further formal investigations of the countermeasure after it is applied. The purpose of this paper is to illustrate that the application of a countermeasure can

be formally verified. All the modelisations and proofs of lemmas given in this paper have been obtained in the COQ [4] formal proof assistant.

Some efforts have already been led in order to formally study electronic circuits and their correctness [10, 5, 2]. The present article differs from these previous works in the sense that its objective is not only to study functional correctness of circuits but also to study security properties of hardware countermeasures.

The rest of the paper is structured as follows. In Sec. 2, the studied countermeasures are presented, and described informally. In Sec. 3, a framework to reason on combinational circuits is detailed. Some convenient circuits are defined in appendix (Sec. A). The application of these tools to the formal description of a single-to-dual-rail transformation is carried out in Sec. 4. It allows to show weaknesses present in WDDL but absent from BCDL. Finally, conclusions are given in Sec. 5. This last section also extends the presented methodology to other kinds of dual-rail circuits.

2 Dual-rail Precharge Logic

2.1 Overview

Dual-rail Precharge Logic (DPL) is a class of logic-level countermeasures. It aims at making the device activity constant and independent of the data being processed. In this logic style, a signal is represented by a pair of wires, hence the *dual-rail* qualifier. A cycle of computation is composed of two phases: (1) a *precharge* phase where each pair of wires is discharged by propagating the NULL value through the combinational part of circuit and (2) an *evaluation* phase where the data is processed by the combinational part of the circuit and in which exactly one wire among each pair toggles its state, depending on the logical value the dual-rail conveys. This protocol is depicted in Fig. 1.

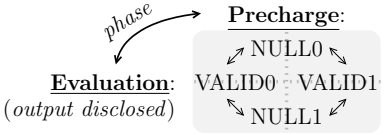


Fig. 1. Separable dual-rail encoding with precharge, yielding a constant activity

Several DPL have been invented along the years: WDDL [13], MDPL [11], DRSL [3], STTL [12], BCDL [8] and SDDL [7] to cite a few.

Each of these proposals succeeds in making the device activity constant. Nonetheless, they differ on some implementation-level aspects. We focus in the sequel on the specific characteristics of WDDL and BCDL.

2.2 Two examples

In this section, an informal description of WDDL (Wave Dynamic Differential Logic) and Balanced Cell-based Differential Logic (BCDL) is provided.

WDDL consists in a separable logic to implement the *true* and *false* halves. Glitches are partial transitions of the nets that are due to races between signals. It is known that they can be responsible for data-dependent leakage [9]. To avoid glitches, WDDL focuses on positive gates. Thus, only AND and OR primitives are used.

Now, AND and OR functions have "short-cut" evaluation. If one input is one, the AND gate has to wait for the second input before evaluating, whereas the OR gate can evaluate one at once. This effect can happen in both evaluation and precharge phases, and cause data-dependent toggling date. During one phase (evaluation or precharge), the activity is constant, but decomposes into events that are data-dependent within the phase (due to small delays between the signals).

BCDL ensures the data-independence of the gates toggling date, thanks to a synchronisation of the inputs at evaluation. The precharge has the functionality to reset all the nodes. Thus, it can be always anticipated, which is implemented by a global signal. This way, BCDL fixes the early propagation effect, and also exhibits no glitches, since by construction a BCDL gate is evaluated only once.

2.3 Vulnerabilities

In DPL styles, the registers can be balanced easily, because they merely implement the "identity function". One way to balance them is to be especially careful at the place-and-route stage. For instance, both in ASIC and FPGA technologies, it is possible to set placement constraints on the two dual registers. If placement constraints are not enough (for instance because the routing would also deserve a similar symmetry), another solution consists in applying a dedicated countermeasures built on top of DPL circuits. A strategy such as "path switching" [1] can be easily applied to the registers. It consist in saving the *true* (resp. *false*) variable in either the *true* (resp. *false*) or the *false* (resp. *true*) register half, depending on a random variable. The functionality remains unchanged, but the leakage is balanced at the first order).

Therefore, in the sequel, we focus on the leakage generated by the combinational logic. These gates are more delicate to perfectly balance, and are more susceptible to cause two major flaws previously mentioned: data dependent glitches and early propagation.

3 Combinational circuits

In this section, we define a formalism that allows to reason about combinational circuits.

3.1 Syntax

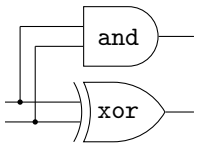
Intuitively, a combinational circuit is a directed acyclic graph whose nodes represent logical gates, and whose edges, which represent wires, are ordered. Despite the fact that this definition is perfectly rigorous from a mathematical point of view, it already involves some heavy mathematical notions which will not ease further reasoning about these objects. So, rather than using this rough definition, we define the combinational circuits as being terms of a process algebra that we define below.

Definition 1 (Combinational circuits). *Let $\mathcal{G}$ be a set of logical gates. The set of* combinational circuits over $\mathcal{G}$ *is defined inductively by (1) the empty circuit $\mathbf{0}$ is a combinational circuit, (2) any gate $g \in \mathcal{G}$ is a combinational circuit, (3) a single wire $\mathbf{I}$ is a combinational circuit, (4) a fork $\mathbf{Y}$, which duplicates a single wire in two wires, is a combinational circuit, (5) a swap $\mathbf{X}$ of two wires is a combinational circuit, and if P and Q are two combinational circuits then so are (6) their parallel composition $P \,|\, Q$ and (7) their sequential composition $P \,;\, Q$.*

Note that by definition, a combinational circuit does not contain any loop.

Regarding the wiring primitives, we have made the choice of minimality. Thus, we have considered a single wire $\mathbf{I}$ instead of a ribbon cable, a single swap $\mathbf{X}$ instead of a generalised permutation operator, and a simple fork $\mathbf{Y}$ instead of a generalised fork that would have replicated a single wire multiple times. This choice, which has no impact on the expressiveness of the calculus (since the generalised operators can easily be defined in terms of these simple ones), allows us to easily define transformations on circuits. Some useful circuits that rotate wires, that interleave or deinterleave wires, or that duplicate wires are thus defined in Section A.

Example 1. We assume that the set of gates contains a `xor` gate, and an `and` gate, i.e. we assume that $\{\texttt{xor}, \texttt{and}\} \subseteq \mathcal{G}$. The half-adder depicted below is represented by the term $\textsc{Half} := (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\texttt{and} \,|\, \texttt{xor})$.



As a convenient circuit, we define, for $n \in \mathbb{N}$, $\mathbf{I}^n$ which intuitively represents a straight ribbon cable composed of n wires as $\mathbf{I}^0 := \mathbf{0}$ and $\mathbf{I}^{n+1} := \mathbf{I} \,|\, \mathbf{I}^n$.

In the sequel, we let $\mathcal{G}$ be a fixed set of gates and we consider combinational circuits over $\mathcal{G}$, unless stated otherwise.

3.2 Well-formedness

By definition, a combinational circuit does not contain any loop. However, some circuits might be ill-formed. This can happen when composing sequentially two

circuits P and Q if the *number of outputs* of P is different from the *number of inputs* of Q. To exclude these ill-formed circuits, we define a simple type system on combinational circuits. For this, we assume that each gate $g \in \mathcal{G}$ has a type (m_g, n_g) where m_g is the *fan-in* (number of inputs) of the gate g and n_g is the *fan-out* (number of outputs) of the gate g. In other words, we assume that we have a typing function $\mathcal{T} : \mathcal{G} \to \mathbb{N} \times \mathbb{N}$.

Definition 2 (well-formed circuits). *Let P be a combinational circuits over $\mathcal{G}$. We say that P is* well-formed *and* has m inputs and n outputs — *written $P : m \otimes n$ — if and only if there exists a typing derivation using the rules of the type system given below. Otherwise, P is said to be* ill-formed.

$$\frac{\mathcal{T}(g) = (m, n)}{g : m \otimes n}\ g \in \mathcal{G} \qquad \frac{}{\mathbf{0} : 0 \otimes 0} \qquad \frac{}{\mathbf{I} : 1 \otimes 1} \qquad \frac{}{\mathbf{Y} : 1 \otimes 2} \qquad \frac{}{\mathbf{X} : 2 \otimes 2}$$

$$\frac{P_1 : m_1 \otimes n_1 \qquad P_2 : m_2 \otimes n_2}{P_1 \,|\, P_2 : m_1 + m_2 \otimes n_1 + n_2} \qquad \frac{P_1 : m \otimes n \qquad P_2 : n \otimes p}{P_1 \,;\, P_2 : m \otimes p}$$

We comment briefly the rule for sequential composition as it is the source of potential ill-formedness. This rule states that for $P\,;\,Q$ to be well-formed, we must have that (1) P is well-formed, (2) Q is well-formed and (3) the number of outputs of P is equal to the number of inputs of Q.

Example 2. Continuing Example 1, we assume that the and gate `and` and the xor gate `xor` have 2 inputs and 1 output, i.e. that $\mathcal{T}(\texttt{and}) = (2, 1)$ and $\mathcal{T}(\texttt{xor}) = (2, 1)$. Then HALF is well-formed and has 2 inputs and 2 outputs, i.e. HALF : $2 \otimes 2$.

Lemma 1 (uniqueness of type). *Let P be a circuit. If $P : n \otimes m$ and $P : n' \otimes m'$ then $n = n'$ and $m = m'$.*

3.3 Semantics

We interpret combinational circuits by partial functions on words over an *alphabet* Σ. Before defining the formal semantics of circuits, we recall briefly some definitions about languages to fix terminology and notations.

An *alphabet* is a finite set, whose elements are called *letters*. A word u over an alphabet Σ is a finite sequence of letters $u = u_1 \cdot \ldots \cdot u_n$ where $u_i \in \Sigma$ for any i. We note Σ^* the set of words over Σ. If $u = u_1 \cdot \ldots \cdot u_n$ is a word over Σ, we note $|u| = n$ its *length*. The set of words of length $n \in \mathbb{N}$ is written Σ^n. We note ϵ the *empty* word, i.e. the unique word of length 0. If $u = u_1 \cdot \ldots \cdot u_n \in \Sigma^*$ and $v = v_1 \cdot \ldots \cdot v_p \in \Sigma^*$, the *concatenation* $u \bullet v$ of u and v is defined by $u \bullet v = u_1 \cdot \ldots \cdot u_n \cdot v_1 \cdot \ldots \cdot v_p$ of length $|u \bullet v| = |u| + |v|$. A language L over Σ is a subset of Σ^*. If $L_1, L_2 \subseteq \Sigma^*$, we let $L_1 \bullet L_2 := \{u \bullet v \mid u \in L_1 \wedge v \in L_2\}$. If $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, we define L^n as $L^0 := \{\epsilon\}$ and $L^{n+1} := L \bullet L^n$. Finally, we define the Kleene closure of L to be $L^* := \bigcup_{i \in \mathbb{N}} L^i$.

In the following, we let Σ be an alphabet. By abuse of notations, we will identify each letter $a \in \Sigma$ with the word $a \in \Sigma^*$ of length 1.

In order to define the semantics of circuits, we assume that each gate $g \in \mathcal{G}$ is interpreted by a partial function $\mathcal{E}(g) : \Sigma^* \rightharpoonup \Sigma^*$, which is defined consistently with respect to the type of g, i.e. if $\mathcal{T}(g) = (m, n)$ then the definition domain of $\mathcal{E}(g)$ is Σ^m and its image is included in Σ^n.

Definition 3. *Let P be a combinational circuit over $\mathcal{G}$ and $x, y \in \Sigma^*$. We say that P computes y on x — written $P \Vdash x \rightsquigarrow y$ — if and only if there exists a derivation of $P \Vdash x \rightsquigarrow y$ according to the following inductive rules.*

$$\frac{x \in \Sigma^* \qquad \mathcal{E}(g)(x) = y \in \Sigma^*}{g \Vdash x \rightsquigarrow y}\ g \in \mathcal{G} \qquad \frac{}{\mathbf{0} \Vdash \epsilon \rightsquigarrow \epsilon} \qquad \frac{}{\mathbf{I} \Vdash a \rightsquigarrow a}\ a \in \Sigma$$

$$\frac{}{\mathbf{Y} \Vdash a \rightsquigarrow aa}\ a \in \Sigma \qquad \frac{}{\mathbf{X} \Vdash ab \rightsquigarrow ba}\ a, b \in \Sigma$$

$$\frac{P_1 \Vdash x_1 \rightsquigarrow y_1 \qquad P_2 \Vdash x_2 \rightsquigarrow y_2}{P_1 \,|\, P_2 \Vdash x_1 \bullet x_2 \rightsquigarrow y_1 \bullet y_2} \qquad \frac{P_1 \Vdash x \rightsquigarrow y \qquad P_2 \Vdash y \rightsquigarrow z}{P_1 \,;\, P_2 \Vdash x \rightsquigarrow z}$$

The next lemma summarises some important results about the semantics of circuits.

Lemma 2. *Let P be a combinational circuit, $x, y, z \in \Sigma^*$ and $m, n \in \mathbb{N}$.*

- *Computation is deterministic.*
  If $P \Vdash x \rightsquigarrow y$ and $P \Vdash x \rightsquigarrow z$ then $y = z$.
- *Existence of a computation implies well-formedness.*
  If $P \Vdash x \rightsquigarrow y$ then $P : |x| \otimes |y|$.
  As a consequence, we have that if $P \Vdash x \rightsquigarrow y$ and $P : m \otimes n$ then $|x| = m$ and $|y| = n$.
- *A well-formed circuit with m inputs and n outputs computes over Σ^m.*
  If $P : m \otimes n$ and $|x| = m$ then there exists y such that $P \Vdash x \rightsquigarrow y$.
  According to the previous results, we thus have that the definition domain of a well-formed circuit with m inputs and n outputs is Σ^m and its image is included in Σ^n.

3.4 Functional equivalence, structural congruence

Functional equivalence Intuitively, functional equivalence relates any two circuits which compute the same function. It is formally defined below.

Definition 4 (functional equivalence). *Two circuits P and Q are* functionally equivalent*, written $P \simeq Q$, if and only if*

$$\forall x, y \in \Sigma^* : P \Vdash x \rightsquigarrow y \iff Q \Vdash x \rightsquigarrow y$$

An important result, that allows compositional reasoning, is that functional equivalence is a *congruence*. We formally state this result below.

Definition 5 (contexts, congruence). *A* context $C[\,]$ *is a circuit with a* hole $[\,]$ *inside. Formally, the syntax of contexts is given below.*

$$C[\,] ::= [\,] \mid C[\,] \,|\, Q \mid P \,|\, C[\,] \mid C[\,] \,;\, Q \mid P \,;\, C[\,]$$

If P is a circuit and $C[\,]$ is a context, we write $C[P]$ the circuit obtained by syntactically replacing the hole in $C[\,]$ with P.

A congruence $\mathcal{R}$ *is an equivalence relation over $\mathcal{C}_{\mathcal{G}}$ that is preserved by every context, i.e. such that whenever $P\mathcal{R}Q$ then for any context $C[\,]$, we have $C[P]\mathcal{R}C[Q]$.*

Theorem 1. *$\simeq$ is a congruence.*

Another property of $\simeq$ is that it identifies all the ill-formed circuits. Formally, if P and Q are ill-formed, then $P \simeq Q$.

Structural congruence Intuitively, structural congruence identifies circuits that only differ in some minor wiring details.

Definition 6. *The* structural congruence $\equiv$ *is the smallest congruence that satisfies the following axioms:*

1. *for any P, Q and R, $(P \,|\, Q) \,|\, R \equiv P \,|\, (Q \,|\, R)$*
2. *for any P, $P \,|\, \mathbf{0} \equiv \mathbf{0} \,|\, P \equiv P$*
3. *for any P, Q and R, $(P \,;\, Q) \,;\, R \equiv P \,;\, (Q \,;\, R)$*
4. *for any P, if $P : n \otimes m$ then $\mathbf{I}^n \,;\, P \equiv P \,;\, \mathbf{I}^m \equiv P$*
5. *for any P, Q, R and S, if $P : n \otimes m$ and $Q : m \otimes p$ then $(P \,;\, Q) \,|\, (R \,;\, S) \equiv (P \,|\, R) \,;\, (Q \,|\, S)$*
6. $\mathbf{Y} \,;\, (\mathbf{I} \,|\, \mathbf{Y}) \equiv \mathbf{Y} \,;\, (\mathbf{Y} \,|\, \mathbf{I})$
7. $\mathbf{Y} \,;\, \mathbf{X} \equiv \mathbf{Y}$
8. $\mathbf{X} \,;\, \mathbf{X} \equiv \mathbf{I} \,|\, \mathbf{I}$
9. $\mathbf{X} \,;\, (\mathbf{Y} \,|\, \mathbf{Y}) \equiv (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})$

Structural congruence preserves the well-formedness. Formally, if $P \equiv Q$ then for any m and n, we have $P : m \otimes n \iff Q : m \otimes n$.

Structural congruence is also a convenient proof method for showing functional equivalence as stated in the following theorem.

Theorem 2. *We have $\equiv \subseteq \simeq$.*

4 Formalisation of WDDL and BCDL

4.1 Dual-rail Precharge Logic

Before defining formally the WDDL and BCDL transformations, we need some more definitions.

In the following, let $\Sigma = \{0, 1\}$. The set of dual-rail words is $\left(\Sigma^2\right)^*$. Let $N := 00$, $T := 10$, $F := 01$ and $E := 11$. Let $\mathrm{NULL} := \{N\}$, $\mathrm{VALID} := \{T, F\}$ and

FAULT :=$\{E\}$. A word $u \in \Sigma^*$ is said to be *null* if and only if $u \in \text{NULL}^*$, to be *valid* if and only if $u \in \text{VALID}^*$, to be *error-free* if and only if $u \in (\text{NULL} \cup \text{VALID})^*$. If $u \in \Sigma^*$, its corresponding value in dual-rail representation is $[u] \in \text{VALID}^*$. It is defined by induction on u by $[\epsilon] := \epsilon$, $[0 \cdot u] := F \bullet [u]$ and $[1 \cdot u] := T \bullet [u]$. Clearly, we have $|[u]| = 2|u|$.

As mentioned before, DPL alternates precharge phase and computation phase. When a switch of phase occurs, input signals acquire their respective values. Thus when switching from precharge to computation, each input signal N becomes either the token T or the token F. To model this, we define on $\text{NULL} \cup \text{VALID}$ the order $\preceq$ such that for any $x \in \text{NULL} \cup \text{VALID}$ we have $x \preceq x$ and for any $x \in \text{NULL}$ and $y \in \text{VALID}$ we have $x \preceq y$. Note that by construction, the elements of VALID are maximal. Due to routing differences, input signals are likely to acquire their respective logic value at different times. To model this evolution when switching from precharge to computation, we extend the definition of $\preceq$ to error-free words as follows: $u \preceq v$ if and only if there exists $n \in \mathbb{N}$ such that $u = t_1 \cdots t_n$, $v = t'_1 \cdots t'_n$ with $\forall i : t_i, t'_i \in \text{NULL} \cup \text{VALID}$ and for any $1 \leq i \leq n$, $t_i \preceq t'_i$. For example, we have $NN \preceq TN \preceq TF$. By construction, since elements of VALID are maximal, we have that if $x \in \text{VALID}^*$ and $x \preceq y$ then $y = x$.

We define the equivalence relation $\sim$ that equates dual-rail words of same length where each corresponding signals has the same nature: null, valid or faulty. Formally, we let $\sim$ be defined on Σ^2 by $x \sim y$ if and only if both x and y are null (i.e. $x, y \in \text{NULL}$) or both x and y are valid (i.e. $x, y \in \text{VALID}$) or both x and y are faulty (i.e. $x, y \in \text{FAULT}$). In other words, $\sim$ is the equivalence relation on Σ^2 such that its equivalence classes are NULL, VALID and FAULT. We extend this definition to dual-rail words as follows: $u \sim v$ if and only if there exists $n \in \mathbb{N}$ such that $u = t_1 \cdots t_n$, $v = t'_1 \cdots t'_n$ with $\forall i : t_i, t'_i \in \Sigma^2$ and for any $1 \leq i \leq n$, $t_i \sim t'_i$.

To define semantics of circuits, we define the following Boolean operators:

- $\neg$ is the unary operator on Σ such that $\neg x = 1$ if and only if $x = 0$.
- $\wedge$ is the binary operator on $\Sigma \times \Sigma$ such that $x \wedge y = 1$ if and only if $x = y = 1$.
- $\vee$ is the binary operator on $\Sigma \times \Sigma$ such that $x \vee y = 0$ if and only if $x = y = 0$.

4.2 Wave Dynamic Differential Logic (WDDL)

WDDL transformation process is not defined for circuits built with arbitrary logical gates. In the following, we thus assume that the circuits to be secured with WDDL are built over the set $\mathcal{G} = \{\texttt{and}, \texttt{not}\}$. The transformation produces a circuit of $\mathcal{C}_{\mathcal{G}'}$ where $\mathcal{G}' = \{\texttt{and}_{\text{WDDL}}\}$.

We assume the following types: $\mathcal{T}_{\mathcal{G}}(\texttt{and}) = (2, 1)$, $\mathcal{T}_{\mathcal{G}}(\texttt{not}) = (1, 1)$ and $\mathcal{T}_{\mathcal{G}'}(\texttt{and}_{\text{WDDL}}) = (4, 2)$.

We assume that the interpretation functions of these gates are defined by:

- $\mathcal{E}_{\mathcal{G}}(\texttt{and})(a \cdot b) := a \wedge b$ for $a, b \in \Sigma$.
- $\mathcal{E}_{\mathcal{G}}(\texttt{not})(a) := \neg a$ for $a \in \Sigma$.
- $\mathcal{E}_{\mathcal{G}'}(\texttt{and}_{\text{WDDL}})(a_t \cdot a_f \cdot b_t \cdot b_f) := (a_t \wedge b_t) \cdot (a_f \vee b_f)$ for $a_t, a_f, b_t, b_f \in \Sigma$.

We are now ready to define the WDDL securisation process. This process is illustrated on Figure 2.

Definition 7. *We define by induction on* $C \in \mathcal{C}_{\mathcal{G}}$ *the WDDL-secured circuit* $\mathrm{WDDL}(C) \in \mathcal{C}_{\mathcal{G}'}$ *by*

$$
\begin{aligned}
\mathrm{WDDL}(\mathbf{0}) &:= \mathbf{0} \\
\mathrm{WDDL}(\mathbf{I}) &:= \mathbf{I} \,|\, \mathbf{I} \\
\mathrm{WDDL}(\mathbf{X}) &:= (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \\
\mathrm{WDDL}(\mathbf{Y}) &:= (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \\
\mathrm{WDDL}(\mathit{and}) &:= \mathit{and}_{\mathrm{WDDL}} \\
\mathrm{WDDL}(\mathit{not}) &:= \mathbf{X} \\
\mathrm{WDDL}(C_1 \,|\, C_2) &:= \mathrm{WDDL}(C_1) \,|\, \mathrm{WDDL}(C_2) \\
\mathrm{WDDL}(C_1 \,;\, C_2) &:= \mathrm{WDDL}(C_1) \,;\, \mathrm{WDDL}(C_2)
\end{aligned}
$$

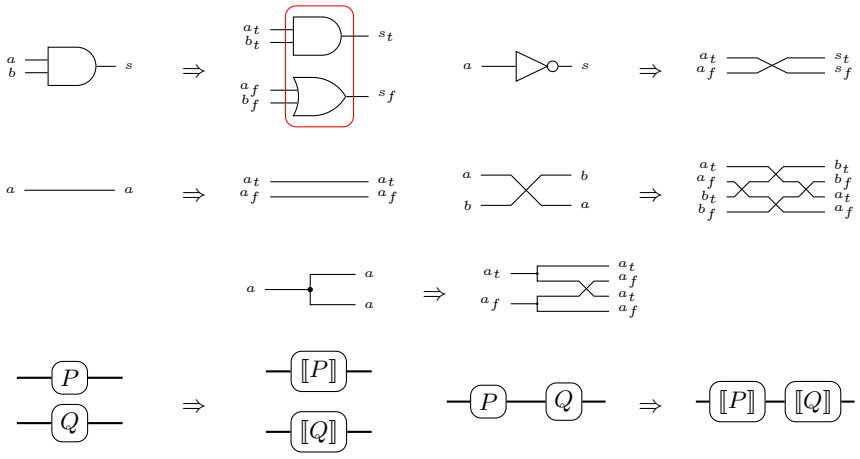


Fig. 2. WDDL securisation process

The WDDL securisation process produces a well-formed circuit if and only if the input circuit is well-formed, as stated below.

Lemma 3. *Let* $C \in \mathcal{C}_{\mathcal{G}}$*. Then*

- *if* $C : n \otimes m$ *then* $\mathrm{WDDL}(C) : 2n \otimes 2m$*.*
- *if* $\mathrm{WDDL}(C) : n' \otimes m'$ *then there exists* n *and* m *such that* $C : n \otimes m$ *and* $n' = 2n$ *and* $m' = 2m$*.*

The following lemma states that a WDDL circuit fulfils the DPL invariants: it propagates the NULL state and the VALID state.

Lemma 4. *Let $C \in \mathcal{C}_{\mathcal{G}}$ and assume that* $\mathrm{WDDL}(C) \Vdash x \rightsquigarrow y$. *Then*

- *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
- *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.

We prove with the following lemma that the WDDL securisation process is sound. In other words, a WDDL secured circuit computes at least as the original circuit.

Lemma 5. *Let $C \in \mathcal{C}_{\mathcal{G}}$. If $C \Vdash x \rightsquigarrow y$ then* $\mathrm{WDDL}(C) \Vdash [x] \rightsquigarrow [y]$.

The next lemma states the converse result: the WDDL securisation process is complete. In other words, a WDDL secured circuit computes no more than the original circuit on valid inputs.

Lemma 6. *Let $C \in \mathcal{C}_{\mathcal{G}}$. If* $\mathrm{WDDL}(C) \Vdash x' \rightsquigarrow y'$ *and* $x' \in \mathrm{VALID}^*$ *then there exists $x, y \in \Sigma^*$ such that $x' = [x]$, $y' = [y]$ and $C \Vdash x \rightsquigarrow y$.*

4.3 Balanced Cell-based Differential Logic (BCDL)

Contrary to WDDL, BCDL transformation process can be defined for circuits build on top of arbitrary logical gates. We thus let $\mathcal{G}$ be the set of basic gates of the circuits to be protected. We assume to have a typing function $\mathcal{T}_{\mathcal{G}} : \mathcal{G} \to \mathbb{N} \times \mathbb{N}$ and an evaluation function $\mathcal{E}_{\mathcal{G}} : \mathcal{G} \to (\Sigma^* \rightharpoonup \Sigma^*)$. The transformation process produces a circuit of $\mathcal{C}_{\mathcal{G}'}$ where $\mathcal{G}' := \{g_{\mathrm{BCDL}} \mid g \in \mathcal{G}\} \cup \{\mathrm{U}_n \mid n \in \mathbb{N}\} \cup \{\mathrm{ANDN}\}$.

We assume the following types for the gates of $\mathcal{G}'$. If $g \in \mathcal{G}$ and $\mathcal{T}_{\mathcal{G}}(g) = (n, m)$ then $\mathcal{T}_{\mathcal{G}'}(g_{\mathrm{BCDL}}) = (n+1, 2m)$. The gate g_{BCDL} corresponding to g has an extra input that indicates whether evaluation is enabled or not and produces a dual-rail result, thus the $2m$ outputs. We also assume $\mathcal{T}_{\mathcal{G}'}(\mathrm{ANDN}) = (2, 1)$ and for $n \in \mathbb{N}$, $\mathcal{T}_{\mathcal{G}'}(\mathrm{U}_n) = (2n, 1)$.

Regarding the interpretation function, we assume that:

- $\mathcal{E}_{\mathcal{G}'}(\mathrm{ANDN})(a \cdot b) := (\neg a) \wedge b$ for $a, b \in \Sigma$.
- for $n \in \mathbb{N}$, and $x \in \Sigma^{2n}$, $\mathcal{E}_{\mathcal{G}'}(\mathrm{U}_n)(x) := 1$ if $x \in \mathrm{VALID}^n$ and $\mathcal{E}_{\mathcal{G}'}(\mathrm{U}_n)(x) := 0$ otherwise.
- for $g \in \mathcal{G}$, if $\mathcal{T}_{\mathcal{G}}(g) = (n, m)$ then for $x \in \Sigma^n$, $\mathcal{E}_{\mathcal{G}'}(g_{\mathrm{BCDL}})(0 \cdot x) := 0^{2m}$ and $\mathcal{E}_{\mathcal{G}'}(g_{\mathrm{BCDL}})(1 \cdot x) := [\mathcal{E}(g)(x)]$.

Before defining the BCDL securisation process on whole circuits, we focus on how a simple gate $g \in \mathcal{G}$ is secured. The idea of BCDL is that evaluation is enabled only once every dual-rail input signal becomes valid and when global precharge signal is low. Figure 3 shows how to achieve this. An unanimity gate U_n (circled on the figure) verifies that every dual-rail signal is valid and transmits the result to a gate ANDN which ands this signal with the negation of the precharge signal. The result is then transmitted to the dual-rail gate g_{BCDL} corresponding to g, which uses this signal to enable evaluation. Formally, for $g \in \mathcal{G}$, we define $C_g := (\mathbf{I} \,|\, (\mathrm{unint}_n \,;\, (\mathrm{dup}_n \,|\, \mathbf{I}^n) \,;\, (\mathbf{I}^n \,|\, (\mathrm{int}_n \,;\, \mathrm{U}_n)) \,;\, \mathrm{ror}_{n+1})) \,;\, (\mathrm{ANDN} \,|\, \mathbf{I}^n) \,;\, g_{\mathrm{BCDL}}$.

The BCDL securisation process is defined thereafter. One difficulty that arises when defining BCDL transformation process is the fact that a circuit only made

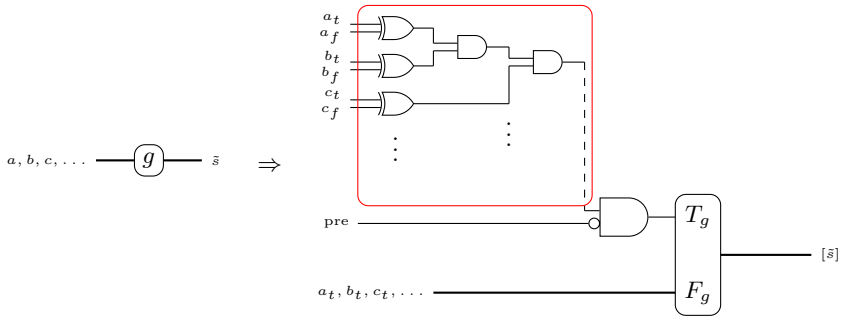


Fig. 3. Securing an arbitrary gate with BCDL

of wires (with no gates) does not need a global precharge signal. For this reason, the BCDL transformation function returns a triple (b, n, C) where $b \in \{\mathrm{tt}, \mathrm{ff}\}$ is a boolean, $n \in \mathbb{N}$ is an integer and C a BCDL-secured circuit. The boolean b indicates whether C has a global precharge signal. The integer n corresponds to the number of inputs of the unsecured circuit (while we do not assume it is well-formed).

Definition 8. *We define by induction on $C \in \mathcal{C}_{\mathcal{G}}$ the BCDL-secured circuit $\mathrm{BCDL}(C) \in \mathcal{C}_{\mathcal{G}'}$ by*

$$
\begin{aligned}
\mathrm{BCDL}(\mathbf{0}) &:= (\mathrm{ff}, 0, \mathbf{0}) \\
\mathrm{BCDL}(\mathbf{I}) &:= (\mathrm{ff}, 1, \mathbf{I} \,|\, \mathbf{I}) \\
\mathrm{BCDL}(\mathbf{X}) &:= (\mathrm{ff}, 2, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})) \\
\mathrm{BCDL}(\mathbf{Y}) &:= (\mathrm{ff}, 1, (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})) \\
\mathrm{BCDL}(g \in \mathcal{G}) &:= (\mathrm{tt}, n, C_g) && \text{if } \mathcal{T}_{\mathcal{G}}(g) = (n, m) \\
\mathrm{BCDL}(C_1 \,;\, C_2) &:= (\mathrm{tt}, n_1, (\mathbf{I} \,|\, C_1') \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{ff}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (\mathrm{tt}, n_1, (\mathbf{Y} \,|\, \mathbf{I}^{2n_1}) \,;\, (\mathbf{I} \,|\, C_1') \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{tt}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (b, n_1, C_1' \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (b, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{ff}, n_2, C_2') \\
\mathrm{BCDL}(C_1 \,|\, C_2) &:= (\mathrm{tt}, n_1 + n_2, (\mathrm{rol}_{2n_1+1} \,|\, \mathbf{I}^{2n_2}) \,;\, (C_1' \,|\, C_2')) \\
& && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{ff}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&\quad (\mathrm{tt}, n_1 + n_2, (\mathbf{Y} \,|\, \mathbf{I}^{2n_1+2n_2}) \,;\, (\mathbf{I} \,|\, \mathrm{rol}_{2n_1+1} \,|\, \mathbf{I}^{2n_2}) \,;\, (C_1' \,|\, C_2')) \\
& && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{tt}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (b, n_1 + n_2, C_1' \,|\, C_2') \\
& && \text{if } \mathrm{BCDL}(C_1) = (b, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{ff}, n_2, C_2')
\end{aligned}
$$

In the sequel, let $\delta_{\mathrm{ff}} := 0$ and $\delta_{\mathrm{tt}} := 1$. The following lemma states that BCDL securisation process produces a well-formed circuit if and only if the input circuit is well-formed.

Lemma 7. *Let $C \in \mathcal{C}_{\mathcal{G}}$ and b, n, C' such that $\mathrm{BCDL}(C) = (b, n, C')$. Then*

- *if $C : n' \otimes m$ then $n = n'$ and $C' : 2n + \delta_b \otimes 2m$.*
- *if $C' : n' \otimes m'$ then there exists m such that $C : n \otimes m$, $n' = 2n + \delta_b$ and $m' = 2m$.*

The next lemma states a result similar to that of Lemma 4: a BCDL circuit propagates the NULL state when the precharge signal is high and propagates the VALID state when the precharge signal is low.

Lemma 8. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *and* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$ *and* $\mathrm{BCDL}(C) \Vdash 1 \cdot x \rightsquigarrow y$ *then*
  - *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
  - *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.
- *if* $b = \mathrm{ff}$ *and* $\mathrm{BCDL}(C) \Vdash x \rightsquigarrow y$ *then*
  - *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
  - *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.

Observe in the previous lemma that precharge signal is the *first* input, when the BCDL secured circuit routes such a signal (i.e. when $b = \mathrm{tt}$).

We can refine this result on circuits C_g where $g \in \mathcal{G}$ as shown in the next lemma. Hence, when precharge signal is high, C_g produces null. It also produces null when input is not valid, whatever the state of the precharge signal is.

Lemma 9. *Let* $g \in \mathcal{G}$*. Then*

- *if* $C_g \Vdash 1 \cdot x \rightsquigarrow y$ *then* $y \in \mathrm{NULL}^*$.
- *if* $C_g \Vdash p \cdot x \rightsquigarrow y$ *and* $x \notin \mathrm{VALID}^*$ *then* $y \in \mathrm{NULL}^*$.

The next lemma states that a BCDL secured circuit computes at least as the original circuit. This is a result analogous to Lemma 5.

Lemma 10. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *and* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$ *and* $C \Vdash x \rightsquigarrow y$ *then* $C' \Vdash 0 \cdot [x] \rightsquigarrow [y]$.
- *if* $b = \mathrm{ff}$ *and* $C \Vdash x \rightsquigarrow y$ *then* $C' \Vdash [x] \rightsquigarrow [y]$.

The converse result is true, as it was the case for WDDL (see Lemma 6). However, note that in the case of BCDL, it is only sufficient to assume that the output is valid (and not the input) as long as the original circuit does not contain a gate g with no outputs.

Lemma 11. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *such that it does not contain a gate* g *with no outputs and let* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$*,* $C' \Vdash 0 \cdot x' \rightsquigarrow y'$ *and* $y' \in \mathrm{VALID}^*$ *then there exists* x, y *such that* $C \Vdash x \rightsquigarrow y$ *and* $x' = [x]$ *and* $y' = [y]$.
- *if* $b = \mathrm{ff}$*,* $C' \Vdash x' \rightsquigarrow y'$ *and* $y' \in \mathrm{VALID}^*$ *then there exists* x, y *such that* $C \Vdash x \rightsquigarrow y$ *and* $x' = [x]$ *and* $y' = [y]$.

4.4 Security properties

In this section, we will illustrate how our formalism allows us to tackle some security properties such as glitches or early-evaluation.

Glitches An electronic glitch is an undesired transition that occurs before the signal settles to its intended value. In DPL, the precharge and the evaluation phase alternate. Thus, when switching from precharge to evaluation, input signals progressively change their value from NULL to VALID. And conversely, when switching from evaluation to precharge, input signals progressively change their value from VALID to NULL. This change of state is precisely modeled by the partial order $\preceq$ we introduced previously (see Section 4.1). Indeed, when switching from precharge to evaluation, input signals modeled by a word of $(\Sigma^2)^*$ take different values $x_1, \ldots, x_n$ where $x_1 \in \text{NULL}^*$, $x_n \in \text{VALID}^*$ and for all $1 \leq i < n$, $x_i \preceq x_{i+1}$. For instance, Figure 4 illustrates the transition of an input signals composed of two dual-rails $(a_t, a_f), (b_t, b_f)$ from the precharge phase to the evaluation phase taking values $NN \preceq TN \preceq TF$.

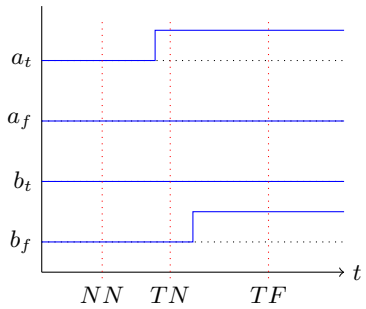


Fig. 4. $\preceq$ models transition of signals from null to valid

The next two lemmas state that WDDL as well as BCDL-secured circuits preserves the partial order $\preceq$.

Lemma 12. *Let $C \in \mathcal{C}_\mathcal{G}$.*
If $\text{WDDL}(C) \Vdash x \rightsquigarrow y$, $\text{WDDL}(C) \Vdash x' \rightsquigarrow y'$ *and* $x \preceq x'$, *then* $y \preceq y'$.

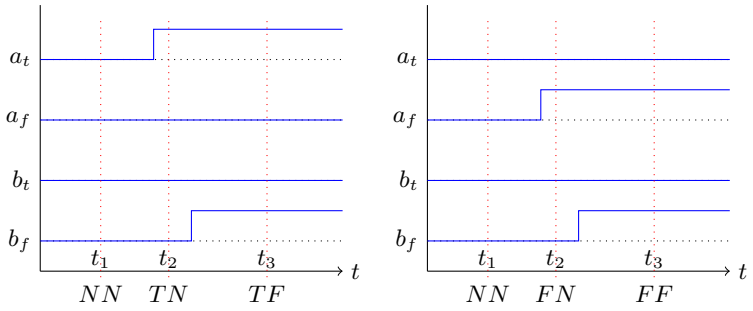
Lemma 13. *Let $C \in \mathcal{C}_\mathcal{G}$ and b, n, C' such that* $\text{BCDL}(C) = (b, n, C')$. *Then*

- *if* $b = \text{tt}$ *then for any* $p \in \Sigma$, *if* $C' \Vdash p \cdot x \rightsquigarrow p \cdot y$, $C' \Vdash p \cdot x' \rightsquigarrow p \cdot y'$ *and* $x \preceq x'$, *then* $y \preceq y'$.
- *if* $b = \text{ff}$ *then if* $C' \Vdash x \rightsquigarrow y$, $C' \Vdash x' \rightsquigarrow y'$ *and* $x \preceq x'$, *then* $y \preceq y'$.

The fact that WDDL and BCDL circuits preserve the partial order $\preceq$ intuitively means that when input signals acquire progressively their values then output signals also acquire progressively their values. By construction of $\preceq$, this means that once a dual-rail output signal has taken its value (in VALID), it won't change its value until the next switch of phase. This precisely means that no glitches are possible.

Early-evaluation In order to address the problem of early-evaluation, we compare the behaviour of circuits on equivalent inputs. Indeed, intuitively, a circuit does not suffer from the early-evaluation problem if it produces the same amount of work on equivalent inputs.

We have defined previously an equivalence relation $\sim$ that equate words which have the same amount of information, i.e. in which corresponding dual-rail signals have the same nature. For instance, on Figure 5, the pair of dual-rail signals (a_t, a_f),(b_t, b_f) in scenario #1 and scenario #2 conveys the same amount of information at time t_1 since $NN \sim NN$, at time t_2 since $TN \sim FN$ and at time t_3 since $TF \sim FF$.



(a) Scenario #1 (b) Scenario #2

Fig. 5. $\sim$ equates states where the nature of dual-rail signals is the same

The next lemma states that BCDL-secured circuit preserves the equivalence relation $\sim$.

Lemma 14. *Let $C \in \mathcal{C}_\mathcal{G}$ and b, n, C' such that* $\mathrm{BCDL}(C) = (b, n, C')$. *Then*

- *if $b = \mathrm{tt}$ then for any $p \in \Sigma$, if $C' \Vdash p \cdot x \rightsquigarrow p \cdot y$, $C' \Vdash p \cdot x' \rightsquigarrow p \cdot y'$ and $x \sim x'$, then $y \sim y'$.*
- *if $b = \mathrm{ff}$ then if $C' \Vdash x \rightsquigarrow y$, $C' \Vdash x' \rightsquigarrow y'$ and $x \sim x'$, then $y \sim y'$.*

On the contrary, a similar result for WDDL circuits does not hold. Indeed, consider the WDDL circuit $\mathtt{and}_{\mathrm{WDDL}}$. We have that $\mathtt{and}_{\mathrm{WDDL}} \Vdash FN \rightsquigarrow F$, $\mathtt{and}_{\mathrm{WDDL}} \Vdash TN \rightsquigarrow N$ and $FN \sim TN$. But $F \nsim N$. In other words, WDDL does not preserve $\sim$ as stated below.

Lemma 15. *There exists $C \in \mathcal{C}_\mathcal{G}$ and x, x', y, y' such that* $\mathrm{WDDL}(C) \Vdash x \rightsquigarrow y$, $\mathrm{WDDL}(C) \Vdash x' \rightsquigarrow y'$, *$x \sim x'$ and $y \nsim y'$.*

The problem of early-evaluation can also be seen in another manner. A DPL circuit does not suffer from the early-evaluation problem if it produces valid outputs only on valid inputs. It is insightful to compare Lemma 6 and Lemma 11

with this idea in mind. Indeed, in the case of BCDL, it is true that a BCDL-secured circuit produces valid outputs only on valid inputs (provided it does not contain gates with no outputs). On the contrary, this result does not hold for WDDL-secured circuit. Indeed, consider the circuit $\texttt{and}_{\text{WDDL}}$. Then we have $\texttt{and}_{\text{WDDL}} \Vdash FN \rightsquigarrow F$ and $F \in \text{VALID}^*$. But $FN \notin \text{VALID}^*$.

Measuring activity of circuits Hitherto, we have shown how problems such as glitches and early-evaluation effects can be detected by just looking at the functionality of a circuit. However, this approach is not sufficient because the principle of physical attacks is to look at the details of implementation and not only to abstract the functionality of circuits.

For instance, the circuit of Figure 6 is a WDDL "and" gate, but the result is conditioned by the fact that the input signals are valid. From the functionality point of view, this circuit does not suffer from early-evaluation effect w.r.t. the definition we suggested before: it preserves $\sim$ and it produces valid outputs only on valid inputs. But this circuit is problematic because it still suffers from early-evaluation. Indeed, the "and" and "or" gate computes whatever the nature of the input signals and then the result is accepted or rejected depending on the nature of the input signals. Hence, we can detect different speed of response of the circuit depending on the input data.

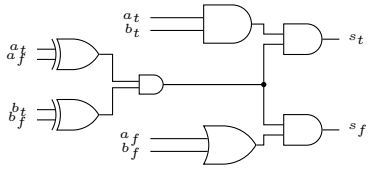


Fig. 6. A WDDL "and" gate with a synchronisation stage

In order to be able to compare different implementations of the same boolean function, we propose to measure activity of circuits. We assume to have a function $\mu^{\mathcal{G}} : \mathcal{G} \to \Sigma^* \to \mathbb{N}$ which measures the activity of each gate $g \in \mathcal{G}$, $\mu^{\mathcal{G}}(g, x)$ being the activity of g on input $x \in \Sigma^*$.

The following inductive rules define the predicate $\mu(C, x, n)$, which relates the activity n produced by a circuit C on input word x.

$$\frac{x \in \Sigma^* \qquad \mathcal{E}(g)(x) \in \Sigma^*}{\mu(g, x, \mu^{\mathcal{G}}(g, x))}\ g \in \mathcal{G} \qquad \frac{}{\mu(\mathbf{0}, \epsilon, 0)} \qquad \frac{}{\mu(\mathbf{I}, a, 0)}\ a \in \Sigma$$

$$\frac{}{\mu(\mathbf{Y}, a, 0)}\ a \in \Sigma \qquad \frac{}{\mu(\mathbf{X}, ab, 0)}\ a, b \in \Sigma \qquad \frac{\mu(P_1, x_1, n_1) \qquad \mu(P_2, x_2, n_2)}{\mu(P_1 \,|\, P_2, x_1 \bullet x_2, n_1 + n_2)}$$

$$\frac{\mu(P_1, x, n) \qquad P_1 \Vdash x \rightsquigarrow y \qquad \mu(P_2, y, m)}{\mu(P_1 \,;\, P_2, x, n + m)}$$

The next lemma gives basic properties of the activity predicate.

Lemma 16. *1. If $\mu(P, x, n)$ and $\mu(P, x, n')$ then $n = n'$.*
2. If $\mu(P, x, n)$ then there exists y such that $P \Vdash x \rightsquigarrow y$.
3. If $P \Vdash x \rightsquigarrow y$ then there exists n such that $\mu(P, x, n)$.

We now study the activity of BCDL-secured circuit. We assume that the activity of the basic gates is such that:

- for any $n \in \mathbb{N}$, for any $x, y \in \Sigma^*$, if $x \sim y$ then $\mu^{\mathcal{G}'}(\mathrm{U}_n, x) = \mu^{\mathcal{G}'}(\mathrm{U}_n, y)$.
- for any $g \in \mathcal{G}$, for any $x, y \in \Sigma^*$, and $s \in \Sigma$, if $|x| = |y|$ then we have $\mu^{\mathcal{G}'}(g_{\mathrm{BCDL}}, s \cdot x) = \mu^{\mathcal{G}'}(g_{\mathrm{BCDL}}, s \cdot y)$.

Then a BCDL-secured circuit has a constant activity on equivalent inputs, as stated in the next lemma.

Lemma 17. *Let $C \in \mathcal{C}_{\mathcal{G}}$ and b, n, C' such that $\mathrm{BCDL}(C) = (b, n, C')$. Then*

- *if $b = \mathrm{tt}$, then for any $p \in \Sigma$, for any $x, x' \in \Sigma^*$ and for any $k, k' \in \mathbb{N}$, if $\mu(C', p \cdot x, k)$, $\mu(C', p \cdot x', k')$ and $x \sim x'$ then $k = k'$.*
- *if $b = \mathrm{ff}$, then for any $x, x' \in \Sigma^*$ and for any $k, k' \in \mathbb{N}$, if $\mu(C', x, k)$, $\mu(C', x', k')$ and $x \sim x'$ then $k = k'$.*

Note that a similar result is not true for the circuit of Figure 6 if we take for $\mu^{\mathcal{G}}(g, x)$ the hamming weight of $\mathcal{E}(g)(x)$. For $(a_t, a_f), (b_t, b_f) = FN$, we would have an activity of 2 (one "xor" gate and one "or" gate react) whereas for $(a_t, a_f), (b_t, b_f) = TN$ we would have an activity of 1 (only one "xor" gate reacts).

5 Conclusions and Perspectives

This article has shown that the transformation from an unprotected to a side channel attack resistant netlist can be captured formally. The scheme described in the article allows to work on any kind of combinational circuit, unprotected or protected. To the authors' best knowledge, it is the first time a circuit-level countermeasure is processed formally. Furthermore, our scheme can be augmented with the verification of some properties. In particular, we formally describe the presence of glitches and of early propagations, properties that were previously only discussed informally or on examples in the embedded secure literature. These properties are tested on two dual-rail logic styles, namely WDDL and BCDL. It is shown that none have glitches, but that WDDL is flawed with early propagation.

As a perspective, we intend to apply these results on other dual-rail styles. For instance, SDDL [7] could be shown to be victim of both glitches and early propagation. Also, some other subtle bugs could be discovered if the inner structure of the logic gates was included in the modeling. Typically, DRSL [3] features a data-dependent glitch happening only internally inside a gate [6]. This flaw cannot be found in the current state of the scheme. In a nutshell, we believe the scope of formal methods can be broadened to detect early some future security troubles related to implementation-level attacks.

References

1. G. Fraidy Bouesse, Gilles Sicard, and Marc Renaudin. Path Swapping Method to Improve DPA Resistance of Quasi Delay Insensitive Asynchronous Circuits. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 384–398. Springer, 2006.
2. Thomas Braibant. Coquet: A coq library for verifying hardware. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2011.
3. Zhimin Chen and Yujie Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *CHES*, volume 4249 of *LNCS*, pages 242–254. Springer, October 10-13 2006. Yokohama, Japan, `http://dx.doi.org/10.1007/11894063_20`.
4. The Coq Development Team. *The Coq Proof Assistant Reference Manual Version 7.2*. INRIA-Rocquencourt, December 2001. http://coq.inria.fr/doc-eng.html.
5. Solange Coupet-Grimal and Line Jakubiec. Certifying circuits in type theory. *Formal Asp. Comput.*, 16(4):352–373, 2004.
6. Jean-Luc Danger, Sylvain Guilley, Shivam Bhasin, and Maxime Nassar. Overview of Dual Rail with Precharge Logic Styles to Thwart Implementation-Level Attacks on Hardware Cryptoprocessors, — *New Attacks and Improved Counter-Measures* —. In *SCS*, IEEE, pages 1–8, November 6–8 2009. Jerba, Tunisia. DOI: 10.1109/ICSCS.2009.5412599.
7. Wei He, Eduardo de la Torre, and Teresa Riesgo. An Interleaved EPE-Immune PA-DPL Structure for Resisting Concentrated EM Side Channel Attacks on FPGA Implementation. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012.
8. Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley. BCDL: A high performance balanced DPL with global precharge and without early-evaluation. In *DATE'10*, pages 849–854. IEEE Computer Society, March 8-12 2010. Dresden, Germany.
9. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008. Seoul, Korea.
10. C. Paulin-Mohring. Circuits as streams in Coq : Verification of a sequential multiplier. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs, TYPES'95*, volume 1158 of *Lecture Notes in Computer Science*, 1996.
11. Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *CHES*, volume 3659 of *LNCS* , pages 172–186, 2005. `http://dx.doi.org/10.1007/11545262_13`.
12. Rafael Soares, Ney Calazans, Victor Lomné, Philippe Maurine, Lionel Torres, and Michel Robert. Evaluating the robustness of secure triple track logic through prototyping. In *SBCCI'08: Proceedings of the 21st symposium on Integrated circuits and system design*, pages 193–198, Gramado, Brazil, 2008. ACM.
13. K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Proceedings of DATE'2004*, pages 246–251, Paris, France., February 2004.

A Wiring

In this section, we define several circuits that manipulate wires and are convenient when defining more complex transformations.

A.1 Rotations

The *right rotation* $\overrightarrow{u}$ of a word $u \in \Sigma^*$ is ϵ if $u = \epsilon$ and $a \cdot v$ if there exists $a \in \Sigma$ and $v \in \Sigma^*$ such that $u = v \bullet a$. Similarly, the *left rotation* $\overleftarrow{u}$ of a word $u \in \Sigma^*$ is ϵ if $u = \epsilon$ and $v \bullet a$ if there exists $a \in \Sigma$ and $v \in \Sigma^*$ such that $u = a \cdot v$.

Clearly, we have for any word $u \in \Sigma^*$ that $\overleftarrow{\overrightarrow{u}} = \overrightarrow{\overleftarrow{u}} = u$, i.e. left and right rotation on words are inverse operations. Moreover, rotations preserve length, i.e. for any $u \in \Sigma^*$, $|\overleftarrow{u}| = |\overrightarrow{u}| = |u|$.

We define by induction on $n \in \mathbb{N}$ the circuit ror_n by (1) $\mathrm{ror}_0 := \mathbf{0}$, (2) $\mathrm{ror}_1 := \mathbf{I}$, and (3) $\mathrm{ror}_{n+2} := (\mathbf{I}^n \,|\, \mathbf{X})\,;\,(\mathrm{ror}_{n+1} \,|\, \mathbf{I})$ for $n \in \mathbb{N}$.

The next lemma states that the circuit ror_n implements right rotation of words of length n.

Lemma 18. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{ror}_n : n \otimes n$, *and*
- *for all $x, y \in \Sigma^*$, $\mathrm{ror}_n \Vdash x \rightsquigarrow y$ if and only if $|x| = n$ and $y = \overrightarrow{x}$.*

Similarly, we define by induction on $n \in \mathbb{N}$ the circuit rol_n by (1) $\mathrm{rol}_0 := \mathbf{0}$, (2) $\mathrm{rol}_1 := \mathbf{I}$, and (3) $\mathrm{rol}_{n+2} := (\mathrm{rol}_{n+1} \,|\, \mathbf{I})\,;\,(\mathbf{I}^n \,|\, \mathbf{X})$ for $n \in \mathbb{N}$.

The next lemma states that the circuit rol_n implements left rotation of words of length n.

Lemma 19. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{rol}_n\,;\,\mathrm{ror}_n \equiv \mathrm{ror}_n\,;\,\mathrm{rol}_n \equiv \mathbf{I}^n$,
- $\mathrm{rol}_n : n \otimes n$, *and*
- *for all $x, y \in \Sigma^*$, $\mathrm{rol}_n \Vdash x \rightsquigarrow y$ if and only if $|x| = n$ and $y = \overleftarrow{x}$.*

A.2 Interleaving

The *interleaving* $u \,|||\, v$ of two words $u, v \in \Sigma^*$ of the same length is defined by induction on u and v by (1) $\epsilon \,|||\, \epsilon := \epsilon$, and (2) $(a \cdot u) \,|||\, (b \cdot v) := ab(u \,|||\, v)$.

We define by induction on $n \in \mathbb{N}$ the circuit int_n by (1) $\mathrm{int}_0 := \mathbf{0}$ and (2) $\mathrm{int}_{n+1} := (\mathbf{I} \,|\, \mathrm{ror}_{n+1} \,|\, \mathbf{I}^n)\,;\,(\mathbf{I} \,|\, \mathbf{I} \,|\, \mathrm{int}_n)$.

The next lemma states that the circuit int_n interleaves two ribbons of n wires.

Lemma 20. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{int}_n : 2n \otimes 2n$, *and*

– *for all* $x, y \in \Sigma^*$, $\mathrm{int}_n \Vdash x \rightsquigarrow y$ *if and only if there exists* $u, v \in \Sigma^*$ *such that* $x = u \bullet v$ *and* $|u| = |v| = n$ *and* $y = u \mid\mid\mid v$.

We define by induction on $n \in \mathbb{N}$ the circuit unint_n by (1) $\mathrm{unint}_0 := \mathbf{0}$ and (2) $\mathrm{unint}_{n+1} := (\mathbf{I} \,|\, \mathbf{I} \,|\, \mathrm{unint}_n)\,;\, (\mathbf{I} \,|\, \mathrm{rol}_{n+1} \,|\, \mathbf{I}^n)$.

The next lemma states that the circuit unint_n deinterleaves a ribbon of $2n$ wires.

Lemma 21. *Let* $n \in \mathbb{N}$. *Then*

– $\mathrm{int}_n\,;\, \mathrm{unint}_n \equiv \mathrm{unint}_n\,;\, \mathrm{int}_n \equiv \mathbf{I}^{2n}$,
– $\mathrm{unint}_n : 2n \otimes 2n$, *and*
– *for all* $x, y \in \Sigma^*$, $\mathrm{unint}_n \Vdash x \rightsquigarrow y$ *if and only if there exists* $u, v \in \Sigma^*$ *such that* $y = u \bullet v$ *and* $|u| = |v| = n$ *and* $x = u \mid\mid\mid v$.

A.3 Duplication

We define by induction on $n \in NN$ the circuit dup_n by (1) $\mathrm{dup}_0 := \mathbf{0}$, and (2) $\mathrm{dup}_{n+1} := (\mathbf{Y} \,|\, \mathrm{dup}_n)\,;\, (\mathbf{I} \,|\, \mathrm{rol}_{n+1} \,|\, \mathbf{I}^n)$.

The next lemma states that the circuit dup_n duplicates a ribbon of n wires.

Lemma 22. *Let* $n \in \mathbb{N}$. *Then*

– $\mathrm{dup}_n : n \otimes 2n$, *and*
– *for all* $x, y \in \Sigma^*$, $\mathrm{dup}_n \Vdash x \rightsquigarrow y$ *if and only if* $|x| = n$ *and* $y = x \bullet x$.