Abstract

Side-channel attacks are efficient attacks against cryptographic devices. They use only quantities observable from outside, such as the duration and the power consumption.

Attacks against synchronous devices using electric observations are facilitated by the fact that all transitions occur simultaneously with some global clock signal.

Asynchronous control remove this synchronization and therefore makes it more difficult for the attacker to insulate *interesting intervals*. In addition the coding of data in an asynchronous circuit is inherently more difficult to attack.

This article describes the Programmable Logic Block of an asynchronous FPGA resistant against *side-channel attacks*. Additionally it can implement different styles of asynchronous control and of data representation.

1 Introduction

Side-channel attacks (SCA) have been put forward mainly by Paul Kocher *et al.* in 1996 in [22]. This first description of a SCA explained how the mere observation of the duration of computations could allow an attacker to retrieve the secret key. The attack was then improved and extended to other cryptosystems [38, 6, 12, 49].

In 1999 Kocher et al. described what they called "DPA 1" [32]. This new attack used the power consumption instead to the duration but yielded the same result: the retrieval of the secret key. The process of this latter attack is relatively simple: a large number of cryptographic operations are monitored and the cipher text stored together with the electric consumption. Then guesses were made of some parts of the secret key, which were confirmed or or not by a statistical processing the data. Other attacks against various cryptosystems were based on this method [5, 42, 18].

Countermeasures soon appeared to protect systems based on a strong algebraic structure [2, 20, 47, 25]. At he protection of opposite symmetric cryptosystems often consisted in introducing some randomization either in the computing process or power consumption to prevent the statistical processing of the acquired data. However "counter-countermeasures" also appeared [19]. Some other protection schemes were designed [26, 10].

An interesting and apparently efficient countermeasure is the WDDL² [46] which duplicates each signal in the circuit so that whatever the value is, one of the lines will toggle. This countermeasure was enhanced by an improved routing of related signals [11], which reduces the differences between the power consumptions of a '1' and a '0'.

Asynchronous circuits, the history of which dates back to 1950, are nowadays increasingly considered as a viable alternative to classical synchronous

¹Differential Power Analysis.

²Wave Dynamic Differential Logic.

designs. Indeed they feature some very useful properties such as flexibility, robustness, high speed and low power. This article brings another good reason to consider asynchronous designs: a greater resistance against *side-channel attacks*.

Some industrial applications of asynchronous ASIC and FPGA begin to appear both in the academic world [28, 29, 44] and in the industry [1].

At the same time synchronous circuits are suffering from problems arising from the distribution of the clock signal through the IC and the excessive power consumption (and thus dissipation!).

As an asynchronous circuit has no centralized clock, the problems associated with the clock distribution, clock skew and power consumption do not exist. In addition this circuits offers advantages like:

- average-time performance,
- lower electromagnetic radiation,
- better robustness towards variations of the power voltage,
- better robustness towards fabrication process variations [31],
- better composability and modularity because of the simple handshake interfaces and the local timing [39] and
- better scrambling of the *side-channel* information [30, 16, 41].

Asynchronous circuits thus seem to be a viable alternative which would remove these limiting factors [35].

Due to these advantages, there has been a resurgence of interest in asynchronous design, especially in the reprogrammable field. There have been several recent successful design projects such as ASPRO-216 [36], AES cryptoprocessor [4], many of Philips designs targeting low power [3, 21], projects focused on designing an asynchronous FPGA from a synchronous one, like MON-TAGE [14] and PGA-STC [27] or targeting asynchronous application-specific FPGAs, locally synchronous, like GALSA [9] and STACC [33] or completely asynchronous like PAPA [43, 45], and other recent works [15, 8, 7, 17, 24]. PGA-STC was developed to implement two-phase bundled-data systems such as micro pipelines, GALSA for massively parallel computing architecture, STACC for reconfigurable computation and PAPA was mainly created and optimized for pipe-lined processes.

This article describe the design of the PLB³ of a new asynchronous FPGA with security as the main requirement, even at the expense of performance. Indeed in the particular case of cryptography performance is second to security even if it cannot be ignored. The FPGA must be able to implement various styles of asynchronous protocols and different representations of data so as to enable comparisons between these representations and protocols as for their ability to thwart the side-channel attacks.

³Programmable Logic Block.

Section 2 describes the representation of data and the different asynchronous protocols used in the FPGA. We also discuss their suitablity for trusted computing. Section 3 shows the construction of the PLB to implement the 4-phase protocol using both binary and ternary representations of data. Section 4 shows the necessary additions to the PLB to accommodate the 2-phase protocols. Section 5 shows how the FPGA is programmed. Finally section 6 concludes the article.

2 Asynchronous Representation of Signals

As opposed to synchronous data, whose validity is guaranteed by the timing of some global "clock" signal, the asynchronous computations are synchronized by the availability of data and, when necessary, by a Request/Acknowledge handshake signalling.

A formal description of delay insensitive representation of data can be found in [48]. In the **Q**uasi-**D**elay **I**nsensitive (QDI) protocols the request is carried by the data itself. This allows to obtain a reliable design, independent of the routing.

The data are transmitted together with the availability information and thus a logic signal or, shorter a "signal", must be represented by more than a single electrical signal or, shorter, a "wire"⁴. In this article, a wire is able to take one of two values, which we denote **0** and **1** regardless of their actual electric implementation.

In order to avoid glitches, a sufficient condition is that given a signal S represented by n wires, the transmission of a new value of S must consist in **exactly one** of the n wires changing its electrical state. This means that the number of wires is greater than or equal to the number of the states of S. As silicon and routing is a precious resource, the number of wires representing a given signal will thus be equal to the number of possible values of this signal.

The most frequently used kind of signal is the binary signal, which carries a {'1','0'} information. Such a signal is encoded with 2 wires. This representation is called "Dual-Rail" or "1-out-of-2". However ternary signals, which carry a {'0','1','2'} information, can also be thought of. Such a signal is represented by 3 wires and one speaks of "1-out-of-3" representation. This representation is more compact than the 1-out-of-2 as for arithmetic: 6 wires in 1-out-of-2 represent 3 1-out-of-2 signals which can take 8 valid values, compared to two 1-out-of-3 signals, which can take 9 valid values. However due to the greater complexity of gates in 1-out-of-3 representation, the binary signals are most of the time preferred.

An asynchronous design may need additional signals, which are specialized to synchronisation. These signal carry no *data* information and can thus be coded on a single wire. They will be referred to as *Acknowledge* signals. The

 $^{^4}$ If one could work with non-standard electrical levels, a $\{-5\ V, 0\ V, +5\ V\}$ representation on a single wire per signal would be acceptable in some cases but we shall restrict ourselves in the following pages to standard CMOS levels: V_{dd} and V_{ss} .

inputs of the gates which receive such a signal will be denoted S_{in} and those driving these signals will be called S_{out} .

2.1 Asynchronous Protocols

There are two main families of QDI asynchronous communication protocols, which differ by the nature of the signalling information: the 2-phase protocols and the 4-phase protocols.

2.1.1 4-Phase Protocol

Under a 4-phase protocol, valid values of a signal are separated by a special value, denoted Ω . The transmission of a value x from an emitter to a receiver proceeds as follows:

	Emitter		Receiver
1	sends x	\longrightarrow	
2		\leftarrow	acknowledges x
3	sends Ω	\longrightarrow	
4		\leftarrow	acknowledges Ω

For instance, if a signal S is represented by n wires $(S_0, S_1, ..., S_{n-1})$, the Ω value will be implemented as the n-tuple (0, 0, ..., 0) while the value i will be represented by $(0, ..., 0, S_i = 1, 0, ..., 0)$.

This particular kind of 4-phase protocol is named "WCHB" in [37, Sec. 2.3.1] and as DPL⁶ among the secure computing community [34].

2.1.2 2-Phase Protocols

Under a 2-phase protocol, no special value is used to separate valid ones. The transmission of a value x from an emitter E to a receiver R proceeds as follows:

	Emitter		Receiver
1	sends x	\longrightarrow	
2		\leftarrow	acknowledges x

In this article we will describe the implementations of two 2-phase protocols:

2-phase-edge protocol:

a signal S, which can take n values is represented by n wires and the arrival of a new value i is signalled by wire i toggling $0 \to 1$ or $1 \to 0$. Note that the instantaneous values of the wires is not significant under this protocol: only the toggles are significant.

⁵Weak Condition Half Buffer.

⁶ Dual-Rail Precharge Logic

2-phase-ledr protocol:

a signal S is represented by two wires: S_d and S_r . The arrival of a new value x, is signalled by one of O_d and O_r toggling $0 \to 1$ or $1 \to 0$ and the value is given by O_d .

Note that the requirement that any change of the value of the signal be implemented by the toggling of exactly 1 wire limits the 2-phase-ledr protocol to binary signals.

Remark 1 The 4-phase protocol can be considered as a 2-phase protocol in which all "valid" values are followed by a Ω dummy value and in which the gates return to the Ω value as soon as all inputs have received the Ω value. The 2-phase protocols are thus inherently twice faster as the 4-phase ones. This is especially important in a FPGA, in which the routing delays are often the limiting factor of the speed of the system. However, even if twice faster, they lead to much more complex gates than the 4-phase ones.

2.2 Initialization of the System

At the initial time of the system's operation, all gates must be reset to a known, deterministic value. (This is also true for synchronous systems even if some flip-flops sometimes need no initialization.)

The requirement of a known, deterministic value, implies no specific value to the wires. However the simplest initialization, which we shall use in this article, consists in initializing all gates so that all wires be set to 0.

The consequence of this initialization is that the parity of the Hamming weight of any signal is 0 just after reset, which implies that its parity is even.

The relevant property just after RESET is thus that:

- under a 4-phase protocol an Ω value is thus output by all gates and
- under a 2-phase protocol the parity of the Hamming weight of the outputs of any gate is 0.

2.3 Request Signalling

The **Request** event is coded into the data of the QDI protocol itself; a request corresponds to a change of one of the wires encoding the signal. A gate will be ready to perform its computation when each of its input have received a request and when all gates using its output have acknowledged the last value sent.

If performance were the major requirement this would not be true: for instance, a AND gate could perfectly output a '0' as soon as one of its inputs has received a '0'. But such an early evaluation would occur only when some input(s) receive a '0' and never when all receive '1'. This difference in timing could potentially leak some information about the computations being performed to a malevolent observer. Thus such "early evaluation" will never be allowed in a secured circuit and computations will always be performed upon the rendez-vous of all data and Acknowledge inputs.

As the arrival of a new value is always signalled by a single wire changing value, the parity of the Hamming weight of any signal changes each time a new value is transmitted.

Under a two-phase protocol, a gate will be ready to compute its output when all its inputs show a parity opposed to the current output parity.

Under a 4-phase protocol a gate is ready to compute as soon as each input has left then Ω state. As Ω is coded as $(0,0,\ldots,0)$ is has an even parity while any valid value, signalled by a single wire at 1, has an even parity. The behaviour of the gates under a 4-phase protocol is thus coherent with the one of the gates under 2-phase protocols. This will be useful for the design of the FPGA.

2.4 Acknowledge Signalling

The Acknowledge signal consists of a single wire, carrying a $\{\Omega, \mathbf{ack}\}$ under a 4-phase protocol or an $\{\mathbf{odd}, \mathbf{even}\}$ "phase" information, under a 2-phase protocol.

Given the "parity" property of the signals, the *Acknowledge* signal is computed as the XOR of all wires carrying the output signal. An OR gate would be enough under a *4-phase* protocol. However it is easy to show the OR and the XOR functions are identical on the allowed domain of values of the wires under a *4-phase* protocol.

This signal is sent by a given gate to those which drive its inputs. When the output of a gate S is sent to more than one gate, D_1 , D_2 ,..., a rendez-vous is computed to combine the synchronization signals coming from the D_i into a single signal, fed to S.

2.5 C-Element

The C-element is the gate which implements the *rendez-vous* of signals. It has an arbitrary number p of input wires, denoted I_1, I_2, \ldots, I_p , and a single output Z, whose equations are:

$$Z = \begin{cases} 1 & \text{if } I_1 = I_2 = \dots = I_p = 1\\ 0 & \text{if } I_1 = I_2 = \dots = I_p = 0\\ Z & \text{otherwise.} \end{cases}$$
 (1)

Eq. 2 shows an equivalent form of Eq. 1.

$$Z = (Z \wedge (I_1 \vee \dots \vee I_p)) \vee (I_1 \wedge \dots \wedge I_p). \tag{2}$$

Where \wedge and \vee are respectively the AND and OR operators.

Fig. 1 depicts the implementation of a C-element derived form Eq. 2, using a multiplexer (MUX), which we use in out FPGA.

In an FPGA the C-element can be implemented in many ways. A p-input C-element can be implemented in p+1-input LUT, provided the output of the LUT can be fed back to one of the inputs.

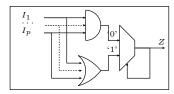


Figure 1: C-element implemented with a MUX.

If Z=0, the MUX selects the AND gate, which will output 1 if and only if $\forall i \in [1,p], I_i=1$. When this condition becomes true, the output of the MUX becomes 1 and the output of the OR is selected to be sent to Z instead of the one of the AND. As $\forall i, I_i=1 \Rightarrow \exists i: I_i \neq 0$, the output is stable at 1. The output remains 1 until all inputs are back to 0. Mutatis mutandis the same proof shows that the output of the gate comes back to 0 when all inputs are 0 and that this value is stable until all inputs are 1 again. Thus the gate correctly implements the rendez-vous with no glitch.

2.6 Asynchronous Computation & Security

2.6.1 Timing Attack

As each gate always waits for every input to be ready before computing its result, the duration of the computations is independent of the data. However a dependency can be generated if the lengths of the wires x_i which implement a signal x are different, thus generating different propagation times for each value of x.

Thus the following necessary condition must hold: for any pair of gates (S, R), connected by a signal x, composed of wires (x_0, x_1, \ldots, x_p) :

- under the 4-phase protocol, the propagation time of the transition from Ω to any value and of the transition from any valid value to Ω S to R must be independent of the value;
- under a 2-phase protocol, the rising and falling times of any output wire must be equal and independent from the former and next value of the signal.

As the condition must be fulfilled by any signal routed through the FPGA, this implies that:

- in any routing channel, all wires must have the same length and the same capacity with respect to V_{dd} or V_{ss} ,
- for any pair of wires in two routing channels connected by a switchbox, the
 propagation time through the switchbox must be the same for all possible
 pairs,

- for any input of a PLB, the propagation time from the network to the processing elements must be uniform,
- for any output, the propagation time to the routing network must be uniform.

If all these conditions are satisfied and if all PLB process information at the same speed the *timing attack* [23, 38, 6, 12, 49] is impossible.

2.6.2 Measurement of Power Consumption

Under the 4-phase protocol, two valid values are separated in time by a Ω value, implemented as all wires at 0. The transition from Ω to a valid value i consist in a rising edge $0 \mapsto 1$ of wire i and the return to Ω is the opposite falling transition.

In order to thwart these attacks the power consumption must be the same for the rising edge of any of the wire x_i which compose a signal x and also for their falling edges. This condition implies that lengths of the x_i through the routing network be the same.

The necessary conditions to thwart the *timing attack* are also necessary here but, in addition the resistances of the output transistors must be equal.

3 4-Phase Protocol

This protocol is the simplest of all three because the instantaneous values of the wires composing any signal are sufficient to determine the value of this signal. We will implement the gates with:

- from 1 to 6 inputs, including the S_{in} signals, and
- from 2 to 4 outputs, **not** including the S_{out} signals.

3.1 Encoding of Signals

Though it is not the only possible one, we shall use the one of Eq. 3 for a signal x in the rest of this article:

if
$$(x_0, x_1) = (0, 0)$$
: $x = \Omega$,
if $(x_0, x_1) = (1, 0)$: $x = \mathbf{0}$,
if $(x_0, x_1) = (0, 1)$: $x = \mathbf{1}$ and
if $((x_0, x_1) = (1, 1)$: forbidden state.

The occurrence of the "(1,1)" forbidden state will always signal either a malfunction or an attack against the system. Fig. 2 depicts the succession of values on a signal X, represented by 2 wires (x_1, x_0) , and, when present, the associated transmissions of the ACK signal by the receiver back to the transmitter.

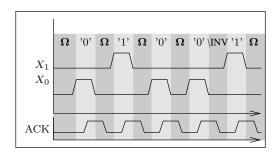


Figure 2: 4-Phase Protocol.

3.2 1-out-of-2,2-input Gates

Let $f(x,y): \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$ a two-variable Boolean function. Its output is a 1-out-of-2 signal represented by two wires O_1 and O_0 . We denote respectively $f^1(x,y)$ and $f^0(x,y)$ the functions computing the values of each wire.

Fig. 3 depicts the minimal structure of a PLB necessary to implement in the most general way a gate with 2 binary inputs. Three signals enter the gate: 2 data signals x and y, respectively implemented by the (x_0, x_1) and (y_0, y_1) pairs of wires, and \mathcal{S}_{in} , the synchronization signal.

The output value (O) is implemented by two $6 \mapsto 1$ LUT, respectively computing the O_0 and O_1 wires. Eq. 4 shows the equations of the outputs. In this equation,

$$O_{1} = \begin{cases} f^{1}(x,y) & \text{if } (x \neq \Omega) \land (y \neq \Omega) \land (\mathcal{S}_{in} = 0) \\ 0 & \text{if } (x = \Omega) \land (y = \Omega) \land (\mathcal{S}_{in} = 1) \\ O_{1} & \text{otherwise.} \end{cases}$$

$$O_{0} = \begin{cases} f^{0}(x,y) & \text{if } (x \neq \Omega) \land (y \neq \Omega) \land (\mathcal{S}_{in} = 0) \\ 0 & \text{if } (x = \Omega) \land (y = \Omega) \land (\mathcal{S}_{in} = 1) \\ O_{1} & \text{otherwise.} \end{cases}$$

$$S_{out} = O_{0} \oplus O_{1}.$$

$$(4)$$

The "memory effect" implied by Eq. 4 is implemented by sending each of O_0 and O_1 to an input of the LUT which drives it. Thus the minimal practical size for the LUT is 64 bits, which can implement any 6-bit \mapsto 1-bit function. As there are two output bits the minimal size of the PLB is 2 LUT.

Even if an OR gate would be enough, the S_{out} signal is computed by a XOR gate (See 2.4).

As the inputs to the LUT are the same, with the exception of the feedback wires, there can be a single connection box to the routing network, which will divide by 2 the total size of the connection boxes. Fig. 3 shows the minimal structure of the PLB, which allows to implement 2-input gates with synchronization.

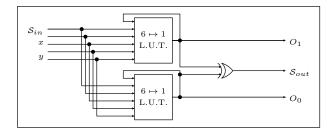


Figure 3: Minimal PLB for 4-phase, 2-input gates.

Remark 2 In Eq. 3, each wire of x and y is loaded with exactly the same number of inputs, as it is necessary to achieve the indiscernability of signals for a malevolent observer.

3.3 1-out-of-2, 3-input Gates

Eq. 4 can be immediately modified into Eq. 5 to add a third input term z and the new equation shows that we need a 7-input LUT with one feedback.

$$O_{1} = \begin{cases} f^{1}(x, y, z) & \text{if } (x \neq \Omega) \land (y \neq \Omega) \land (z \neq \Omega) \land (\mathcal{S}_{in} = 0) \\ 0 & \text{if } (x = \Omega) \land (y = \Omega) \land (z = \Omega) \land (\mathcal{S}_{in} = 1) \\ O_{1} & \text{otherwise.} \end{cases}$$

$$O_{0} = \begin{cases} f^{0}(x, y, z) & \text{if } (x \neq \Omega) \land (y \neq \Omega) \land (z \neq \Omega) \land (\mathcal{S}_{in} = 0) \\ 0 & \text{if } (x = \Omega) \land (y = \Omega) \land (z = \Omega) \land (\mathcal{S}_{in} = 1) \\ O_{0} & \text{otherwise.} \end{cases}$$

$$S_{out} = O_{0} \oplus O_{1}.$$

$$(5)$$

As the 3-input gates need 6 inputs for a 3-variable function, they cannot be implemented in the structure of Fig. 3, on which each $6 \mapsto 1 \text{ LUT}$ has 5 inputs from the routing network and 1 feedback input.

As it is not realistic to use two $7 \mapsto 1$ LUT because of the number of programming points $(2 \times 128 \text{ bits})$, we separate the *rendez-vous* + *computation* function from the *memory* function and introduce a specific component: the **memory** point.

Fig. 4 depicts the **memory point**, which consists in a pair of C-elements, together with a XOR gate, which computes the S_{out} signal. Two MUX, under control of a single programming point, allow to bypass the C-elements. It will be useful when implementing the 2-phase protocols.

Fig. 5 depicts the schematic of the 2-input 1-out-of-3 gate. The ancillary "return to Ω " function is implemented by a specialized 6-input OR gate while the $6 \mapsto 1$ LUT are programmed to compute the rendez-vous and the functions $F^1(x,y,z]$ and $f^1(x,y,z)$.

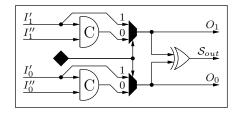


Figure 4: Memory Point.

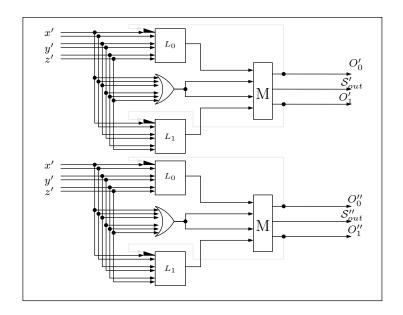


Figure 5: Binary 3-input gate with 4-phase protocol.

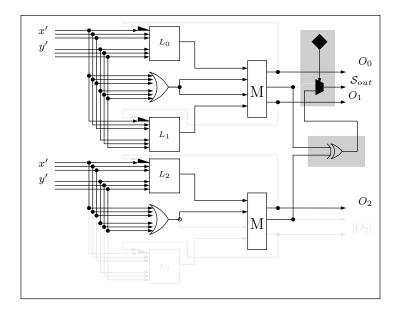


Figure 6: Structure of PLB needed to implement a ternary 2-input gate.

Remark 3 Note that it is much better use of the LUT than the one implied by Fig. 3, in which all bits corresponding to the feedback input set to 1 are filled with '1' to implement the inclusive OR of all 4 input bits.

The 4-LUT PLB can implement two independent 3-input, 1-out-of-2 functions. Ex: a full-adder.

Remark 4 The wiring depicted by Fig. 5 can handle any gate the inputs of which sum up to 6 wires (Ex: one S_{in} + one 1-out-of-2 input + one 1-out-of-3 input; two S_{in} , two 1-out-of-2 inputs, etc...).

Remark 5 The feed-back and the associated MUX at the inputs of LUT could be removed. However they will be useful later for the implementation of the 2-phase-ledr protocol.

3.4 1-out-of-3, 2-input Gates

Just as the 1-out-of-2, 3-Input Gates, the 1-out-of-3, 2-input gates need 6 inputs but they need three outputs, each of them equipped with a memory point. Strictly speaking, a 1-out-of-3 gate needs three LUT, each of them implementing one of the functions $O_i = f^i(x, y), i = 0, 1, 2$.

However as most of the gates in a design will still be binary, the PLB features four $6 \mapsto 1$ LUT. One of them will remain unused and filled with 0 when implementing a 1-out-of-3 gate. The computation of the \mathcal{S}_{out} signal needs some

specialized hardware. Fig. 6 depicts the new PLB needed for a 2-input 1-out-of-3 gate, with the supplementary devices gate in a grey rectangle:

- a MUX, controlled by a programming point, which allows to use the PLB either as two separate 2-binary input, binary output gates or a single combined gate and
- a single XOR gate which computes the XOR of all four outputs of the memory points.

For the same reason of compatibility with the binary gates, the inputs to the pairs of LUT are split into two groups. The load to each of the 12 input wires is exactly the same, thus equalizing the power consumptions of all possible transitions on inputs.

Remark 6 The OR gates which compute the "return to Ω " signal are not grouped but will compute output the same value as their input are the same.

3.5 Conclusion as for the 4-Phase Protocol

In order to implement 2- and 3-inputs gates under the 4-phase protocol, the PLB must at least consist of four $6 \mapsto 1$ LUT, named L_0 , L_1 , L_2 and L_3 . One input of each LUT can be replaced with a feedback signal equal to the output pin.

The schematic depicted on Fig. 6 is general: it can implement any gate with:

- inputs consisting in any combination of 6 wires or less, including the S_{in} signals, and
- outputs consisting of any combination of 4 wires, **not** counting the S_{out} signals: 2 binary outputs, with separate S_{out} signals, 1 ternary output with a single acknowledge-out signal or 1 quaternary output with an S_{out} signal.

4 2-Phase Protocols

4.1 Phase of a Signal

Under the 2-phase protocols valid values of a signal are not separated by " Ω " markers. However, as the arrival of a new value (possibly identical to the preceding one) is indicated by the toggling a exactly one wire, the parity of the Hamming weight of the wires which represent a signal toggles at each new data.

In the following pages, the phase of the signal X, denoted " $\phi(X)$ ", is by definition, the parity of the Hamming weight of the wires representing X.

Remark 7 For Acknowledge signals, which consist in a single wire, the phase is equal to the value of the wire itself. The name of an Acknowledge signal A will thus be used instead of $\phi(A)$.

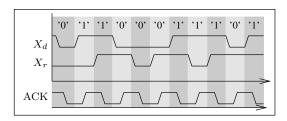


Figure 7: Transmission of a Signal and the acknowledge under the 2-phase-ledr protocol.

At the beginning of the computation, all wires are set to a known value. 2-phase protocols require that, after initialization and before any computation is started, the parities of all signals be the same, say even. A simple way of ensuring this even parity is to initialize all wires to 0.

As the phase of a signal toggles with every new valid value, a given gate is ready to compute its output when the phases of all "data" signals at its inputs are the same, different from the current phase of the output and the phase of the S_{in} signal, if present, the same as the output phase.

After the gate has performed its computation, the phase of its outputs become the common one of the data inputs and thus the S_{out} signal toggles.

4.2 2-Phase, LEDR Protocol

This protocol is referred to as "level-encoded dual-rail", or LEDR [13].

4.2.1 Transmission of a Signal

Fig. 7 shows the transmission protocol of the successive values of a signal, together with the acknowledge signal. One can see that:

- a signal X is represented by two wires: the "data wire": (X_d) and the "repeat" wire: (X_r) ;
- each time a value is sent, exactly one wire toggles;
- the value of the signal X is the value of the X_d signal, thus the oncoming a a new value, different from the preceding one is signalled by the toggling of X_d ;
- the oncoming of a new value, identical to the preceding one is signalled by X_r toggling; thus the instantaneous value of X_r is irrelevant, only its toggling are significant.

Remark 8 The 2-phase-ledr protocol is restricted to binary signals. Otherwise, the transition between two values would imply that more than a single wire toggle.

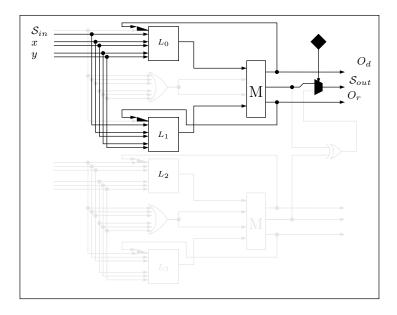


Figure 8: 2-input Gate under the 2-phase-ledr protocol.

4.2.2 Binary 2-input Gates

Let $f(x,y): \mathbb{F}_2 \times \mathbb{F}_2 \mapsto \mathbb{F}_2$ a two-variable Boolean function. The inputs are represented by 4 wires: x_d , x_r , y_d and y_r , to which a synchronization signal, S_{in} , may added and the output signal O represented by two wires: O_d and O_r , together with an acknowledge output S_{out} .

The equations of the output wires are:

$$O_{d} = \begin{cases} fx_{d}, y_{d}) & \text{if } (\phi(x)) = 0) \land (\phi(y)) = 0) \land (\mathcal{S}_{in} = 1), \\ fx_{d}, y_{d}) & \text{if } (\phi(x)) = 1) \land (\phi(y)) = 1) \land (\mathcal{S}_{in} = 0), \\ O_{d} & \text{otherwise}, \end{cases}$$

$$O_{r} = \begin{cases} fx_{d}, y_{d}) & \text{if } (\phi(x)) = 0) \land (\phi(y)) = 0) \land (\mathcal{S}_{in} = 1), \\ fx_{d}, y_{d}) & \text{if } (\phi(x)) = 1) \land (\phi(y)) = 1) \land (\mathcal{S}_{in} = 0), \\ O_{r} & \text{otherwise}. \end{cases}$$

$$S_{out} = O_{d} \oplus O_{r}.$$

$$(6)$$

Eq. 6 shows that each of (O_d, O_r) is a a function of 6 variables:

- two input data signals, represented by 4 wires,
- one S_{in} signal, represented by a single wire and
- one feed-back signal, also 1 wire.

These functions can be implemented in the same hardware as the corresponding gate under the 4-phase protocol. Fig. 8 shows the assignment of the wires.

The hardware elements which are not used to implement this gate are represented in dashed lines:

- the 6^{th} input to the $6 \mapsto 1$ LUT, which is replaced by the feed-back,
- the 6-input OR gate,
- the memory element, which is programmed as "transparent" using its internal programming point (See Fig. 4).

Note that, opposite to the case of the 4-phase protocol, here, the S_{out} value must be computed by a XOR gate.

4.2.3 3-input Gates

Let $f(x,y,z): \mathbb{F}_2^3 \mapsto \mathbb{F}_2$. Eq. 7 shows the expressions of the output wires.

$$O_{d} = \begin{cases} f(x, y, z) & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ f(x, y, z) & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ O_{d} & \text{otherwise}, \end{cases}$$

$$O_{r} = \begin{cases} \frac{f(x, y, z)}{f(x, y, z)} & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ f(x, y, z) & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ O_{r} & \text{otherwise}. \end{cases}$$

$$S_{out} = O_{d} \oplus O_{r} = (7)$$

Eq. 7 shows that each of O_d and O_r is a variable of 7 input variables and cannot thus be implemented in a $6 \mapsto 1 \text{ LUT}$.

4.2.4 Practical Implementation

Under the 4-phase protocol the outputs were set back to 0 by the rendez-vous of the 0 coming from the LUT and the 0 coming from the 6-input OR gate. Under the 2-phase protocol a OR gate cannot express the "return to 0" condition. Therefore the wiring of Fig. 5 is modified according to Fig. 9.

Two MUX, controlled by a programming point, are added, which allow to replace the 6-in OR gate by the two other $6 \mapsto 1 \text{ LUT}$ of the PLB. This way, each of O_d and O_r is now a rendez-vous of the outputs of 2 LUT:

$$O_d = rendez\text{-}vous(L_0, L_2)$$

 $O_r = rendez\text{-}vous(L_1, L_3)$

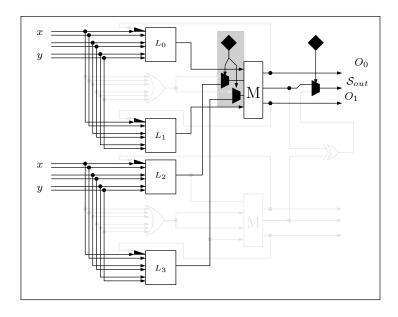


Figure 9: Implementation of the 3-input gate under the 2-phase protocol.

Eq. 8 shows the programming of LUT L_0 and L_2 and Eq. 9 shows the programming of LUT L_1 and L_3 .

$$L_{0} = \begin{cases} f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ 0 & \text{otherwise}, \end{cases}$$

$$L_{2} = \begin{cases} f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ 1 & \text{otherwise}. \end{cases}$$

$$(8)$$

When the conditions for a transition are fulfilled, L_0 and L_2 have the same value. Thus the *rendez-vous* occurs and O_r takes its new value. Otherwise $L_0 = 0$ and $L_2 = 1$, the C-element within the memory element has different

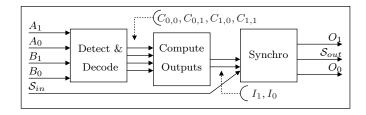


Figure 10: Global Structure of a 2-input 2-Phase-Edge gate.

values on its inputs and O_d is locked.

$$L_{1} = \begin{cases} \frac{f(x_{d}, y_{d}, z_{d})}{f(x_{d}, y_{d}, z_{d})} & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ 0 & \text{otherwise}, \end{cases}$$

$$L_{3} = \begin{cases} \frac{f(x_{d}, y_{d}, z_{d})}{f(x_{d}, y_{d}, z_{d})} & \text{if } (\phi(x) = 0) \land (\phi(y) = 0) \land (\phi(z) = 0) \land (\mathcal{S}_{in} = 1), \\ f(x_{d}, y_{d}, z_{d}) & \text{if } (\phi(x) = 1) \land (\phi(y) = 1) \land (\phi(z) = 1) \land (\mathcal{S}_{in} = 0), \\ 1 & \text{otherwise}. \end{cases}$$

$$(9)$$

Mutatis mutandis the same demonstrations shows the validity of O_r .

4.2.5 Conclusion on the 2-Phase, LEDR Protocol

Apart from the shaded area in Fig. 9 the 2-phase-ledr protocol needs the same resources as the 4-phase protocol.

As for security, all inputs to the gates have an equal load but the value of a signal X is the value of one of x_d . This is a potential security risk, which will have to be investigated as soon as the ICs have been delivered.

4.3 2-Phase, Edge Protocol

Signals under the 2-phase-edge protocol can take an arbitrary number of values. Binary signals are represented by 2 wires, ternary signals are represented by 3 wires, etc... However the complexity of the gates is quadratic in the number of wires per signal. Thus the use of this protocol is in practice limited to binary signals. The complexity of the gates is also quadratic in the number of inputs. Again this limits in practice the number of inputs to 2. In the sequel signals are binary and a signal X is thus represented by 2 wires: (x_0, x_1) .

The coding of the signals relies exclusively on toggling of wires. the instantaneous values of the wires is always irrelevant. This means that the current state of 4 wires has to be stored. Thus even for a 2-input gate all four LUT of the PLB will have to be used.

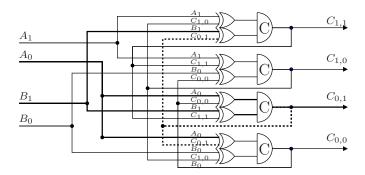


Figure 11: $2 \times 2 - DW$.

4.3.1 Structure of a Gate

The global structure of a 2-input gate under the 2-phase-edge protocol is depicted by Fig. 10. The operation of the gate is divided in three steps:

Detection: waits for an edge on A_i and one on B_j and toggles the corresponding $C_{i,j}$,

Computation: toggles $I_{f(i,j)}$ and

Synchronization: toggles $O_{f(i,j)}$ and S_{out} if and only if S_{in} has toggled since the last data output.

Detection: 2x2-decision wait The detection and the decoding of the input data is performed by the circuitry known as the "**2x2-decision wait**" or, shorter, the " $2 \times 2 - DW$ ". The circuitry, shown on Fig. 11, works as follows:

- 1. assume an initial state such that, for each C-element, the inputs are equal, (as this is the initial state, with all wires set to 0, the recurrence can start),
- 2. after an input value $i \in \{0, 1\}$ has arrived on input port A and an input value $j \in \{0, 1\}$ on input port B, A_i and B_j have toggled (double-thickness continuous lines)
- 3. at this point:
 - one input to $C_{i,1-j}$ has toggled $\Rightarrow C_{i,1-j}$ is unchanged,
 - one input to $C_{1-i,j}$ has toggled $\Rightarrow C_{1-i,j}$ is unchanged,
 - both inputs to $C_{i,j}$ have toggled $\Rightarrow C_{i,j}$ toggles,
- 4. the new value of $C_{i,j}$ is sent to the next stage and to the appropriate XOR gates to cancel the unwanted toggling of $C_{i,1-j}$ and $C_{i,1-j}$ (double-thickness dashed lines),
- 5. all four C-elements now have their inputs identical, which was the initial situation and $C_{i,j}$ has toggled, indicating to the next stage that:

- both input ports A and B have received a new data,
- the data just arrived on A was i and
- the data just arrived on B was j.

Each of the $C_{i,j}$ can be expressed as:

$$C_{i,j} = rendez\text{-}vous(A_i \oplus C_{i,1-j}, B_j \oplus C_{1-i,j})$$

$$\tag{10}$$

Each of the $C_{i,j}$ is a 5-term expression depending of:

- three feedback lines: $C_{i,j}$ (itself), $C_{i,1-j}$ and $C_{1-i,j}$ and
- two input lines: A_i and B_j .

Though the expression would fit in a $6 \mapsto 1$ LUT, the feedback from one LUT to the other would have to be routed through the general routing network, which has the following drawbacks:

- it consumes routing resources,
- the timings of the feedbacks will be different between the feedback of a LUT to itself (which is routed inside the PLB) and other, routed outside. This could be an attack point;
- the 2-input gate will always need 2 PLB: one for the 2×2 DW and one for the computation itself.

Therefore, these feedback have been added to the PLB, as shown on Fig. 12, which. As on preceding figures, the black triangles at the inputs of the LUT are MUX controlled by programming points, which are denoted by "[./.]" in Eq. 11.

With these notations the equations of the $4.6 \mapsto 1$ LUT are:

$$L_{0} = \text{LUT} \left(\left[I'_{0}/L_{0} \right], \left[I'_{1}/L_{1} \right], \left[I'_{2}/L_{2} \right], \left[I'_{3}/L_{3} \right], I'4, I'5 \right) \right) L_{1} = \text{LUT} \left(\left[I'_{0}/L_{0} \right], \left[I'_{1}/L_{1} \right], \left[I'_{2}/L_{2} \right], \left[I'_{3}/L_{3} \right], I'4, I'5 \right) \right) L_{2} = \text{LUT} \left(\left[I''_{0}/L_{0} \right], \left[I''_{1}/L_{1} \right], \left[I''_{2}/L_{2} \right], \left[I''_{3}/L_{3} \right], I''4, I''5 \right) \right) L_{3} = \text{LUT} \left(\left[I''_{0}/L_{0} \right], \left[I''_{1}/L_{1} \right], \left[I''_{2}/L_{2} \right], \left[I''_{3}/L_{3} \right], I''4, I''5 \right) \right)$$

$$(11)$$

To implement the $2 \times 2 - DW$ the input lines are assigned as in Eq. 12 and depicted by Fig. 13:

in which "NC" means "not connected" and Eq. 13 shows the interconnection of the feedbacks needed to implement the $2 \times 2 - DW$.

$$L_{0} = C_{0,0} = LUT(C_{0,0}, C_{0,1}, C_{1,0}, B_{0}, A_{0}, A_{1})$$

$$L_{1} = C_{0,1} = LUT(C_{0,0}, C_{0,1}, B_{1}, C_{1,1}, A_{0}, A_{1})$$

$$L_{2} = C_{1,0} = LUT(C_{0,0}, B_{0}, C_{1,0}, C_{1,1}, A_{0}, A_{1})$$

$$L_{3} = C_{1,1} = LUT(B_{1}, C_{0,1}, C_{1,0}, C_{1,1}, A_{0}, A_{1})$$

$$(13)$$

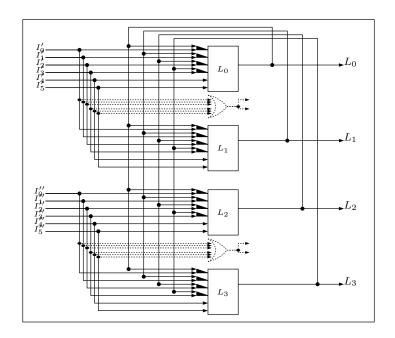


Figure 12: PLB with all feedbacks for the 2-phase-edge protocol.

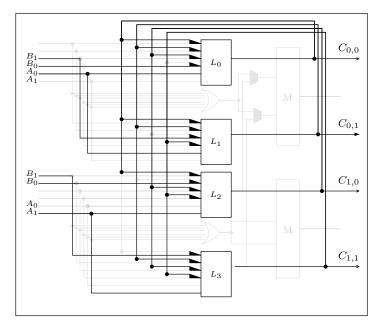


Figure 13: Wiring used to implement the $2\times 1\text{-decision-wait}.$

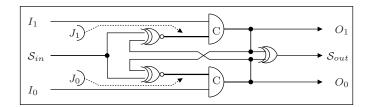


Figure 14: 2×1 -decision-wait.

Remark 9 A_1 is useless to compute $C_{0,0}$ and $C_{0,1}$ and A_0 is useless to compute $C_{1,0}$ and $C_{1,1}$. The reason why these inputs are connected to the network but ignored in the programming of the LUT is that B_0 and B_1 are connected twice from the network to the PLB and that the loads on this network must be identical for both variables.

Computation & synchronization The 2×2 – DW stage provides a decoded output: $C_{i,j}$ toggles if i and j data have arrived on inputs A and B respectively.

Computing the outputs is then straightforward: each of O_1 and O_0 outputs is the XOR of the relevant $C_{i,j}$. Let's see some examples:

Gate	O_1	O_0
AND	$C_{1,1}$	$C_{0,0} \oplus C_{0,1} \oplus C_{1,0}$
NAND	$C_{0,0} \oplus C_{0,1} \oplus C_{1,0}$	$C_{1,1}$
OR	$C_{1,1} \oplus C_{0,1} \oplus C_{1,0}$	$C_{0,0}$
NOR	$C_{0,0}$	$C_{1,1} \oplus C_{0,1} \oplus C_{1,0}$
XOR	$C_{0,1}\oplus C_{1,0}$	$C_{0,0}\oplus C_{1,1}$
NXOR	$C_{0,0} \oplus C_{1,1}$	$C_{0,1} \oplus C_{1,0}$

Synchronization The synchronization is performed by a device called " 2×1 -decision-wait" (or, shorter: $2 \times 1 - DW$). Fig. 14 depicts the schematic of the $2 \times 1 - DW$.

The 2×1 – DW works as follows:

- 1. In the initial state, the following relations hold: $O_1 = I_1$, $O_0 = I_0$ and $\underline{\mathcal{S}_{in}} = O_0 \oplus O_1$, which imply $J_1 \neq I_1$ and $J_0 \neq I_0$. (because $J_1 = \overline{\mathcal{S}_{in} \oplus O_0} = \overline{O_0 \oplus O_1 \oplus O_0} = \overline{O_1} = \overline{I_1}$, idem for J_0);
- 2. Assume I_i toggles and thus becomes equal to J_i , the C-element transmits the common value of its inputs to O_i ,
- 3. as O_i toggles, J_{1-i} toggles too and becomes equal to O_{1-i} .
- 4. until S_{in} toggles, we have $I_0 = J_0$ and $I_1 = J_1$: even if one of the inputs toggles, the C-elements will remain stable;

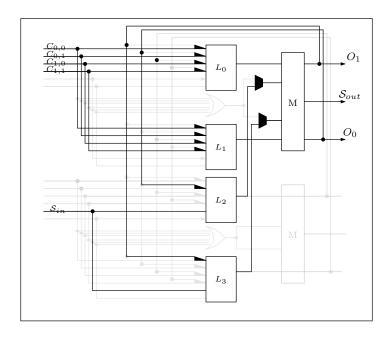


Figure 15: Wiring used to implement the 2×1 -decision-wait.

5. when S_{in} toggles, J_0 and J_1 toggle and the system is back in the initial state.

If one wants to combine the computation stage with the 2×1 – DW, it cannot be done in 2 lut.

It is not because of the complexity of the functions: each of O_1 and O_0 is a function of 2 feed-backs, 1 S_{in} and at most 3 $C_{i,j}$, at least if one does not want to implement trivial, constant functions.

However, the set of 2 LUT together would need 2 feed-backs, 1 S_{in} and 4 $C_{i,j}$, which is one more than the number of available wires. Thus we must use a full PLB.

If we use the full PLB, the memory element will provide the necessary C-element and the LUT become purely combinatorial. The inputs will be assigned following Eq. 14 and depicted on Fig. 15:

$$(I'_0, I'_1, I'_2, I'_3) = (C_{0,0}, C_{0,1}, C_{1,0}, C_{1,1}) \text{ and } I''_4 = \mathcal{S}_{in}$$
 (14)

Then the LUT are programmed as by Eq. 15

$$L_{0} = f^{1}(C_{0,0}, C_{0,1}, C_{1,0}, C_{1,1})$$

$$L_{1} = f^{0}(C_{0,0}, C_{0,1}, C_{1,0}, C_{1,1})$$

$$L_{2} = \overline{O_{0} \oplus S_{in}}$$

$$L_{3} = \overline{O_{1} \oplus S_{in}}$$
(15)

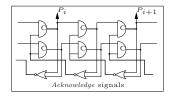


Figure 16: FIFO memory for programming.

4.3.2 Conclusion on the 2-Phase, Edge Protocol

The 2-phase-edge protocol is difficult to implement in a FPGA without special hardware added to the PLB: it takes two PLB to implement a single 2-input gate.

However this protocol has advantages as for security because the instantaneous value of the wires is not significant in itself. For instance '1' is represented alternatively by the rising and the falling edge of a given wire. An attacker trying DPA, for instance, would have to exhibit the difference between the average consumption of both edges on wire '1' and the same average on wire '0'.

5 Programming the FPGA

The FPGA can be partially programmed: it is divided in square blocks which can be programmed separately from the other.

The programming chain is a set of asynchronous FIFO memories. An elementary stage of these FIFO is depicted by Fig. 16.

At RESET time, all C-elements are set to zero by a general RESET wire. Then the programming bits are fed to the FIFO, separated by Ω values. The last stage of each FIFO is particular: the *Acknowledge* signal is controlled by an external pin. During the programming of the block, the *Acknowledge* signal is held low. This way the programming bits are stacked in the FIFO and the FPGA becomes functional.

If a partial reconfiguration is wanted, the chosen blocks are cleared by allowing the *Acknowledge* signal of their last stage to acknowledge the value in the last stage. Then the FIFO is activated again until all bits have gone thought it. At this point, the *Acknowledge* signal is blocked again and the FIFO is ready to receive a new set of configuration bits.

During the configuration of the FPGA, all outputs of PLB are kept at 0 to avoid short-circuits. The PLB are programmed first, while all switchboxes are left in an insulation mode. Then the switchboxes are programmed to connect the newly reconfigured part to be connected to the still working part. It is the designer's responsibility to ensure that the new part can create no conflict with the existing part.

6 Conclusion

We have presented the programmable logic block of an asynchronous FPGA, which is oriented towards security rather than performance. In particular we have chosen not to implement one of the advantages of an asynchronous design, which usually allows to compute in average time: the early evaluation. This choice is deliberate as early evaluation is a security risk [40].

The FPGA can accommodate various sizes of data as well as various styles of asynchronous control, thus making it possible for the end user to design mixed styles of logic, depending on the applicative requirements. Incidentally, this FPGA is also a valuable prototype that allows to perform comparisons between styles of asynchronous protocols.

A silicon is being manufactured and will be used for intensive testing. The different resistances of the various protocols against SCA will be evaluated. In particular the strict link under the 2-phase-ledr protocol between the value of a signal X and the one of the X_d wire will decide whether this protocol is suitable at all for a secure implementation.

References

- [1] Achronix semiconductors. http://www.achronix.com/.
- [2] Elena Trichina & Antonio Bellaza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In *LNCS CHES 2002*, volume 2523, pages 98–113, 2002.
- [3] K.v. Berkel, R. Burgress, J. Kessels, A. Peeters, M. Roncken, and F. Schalij. A fully-asynchronous low-power error corrector for dcc player. *IEEE Journal of Solid-State Circuits*, 29:1429–1439, 1994.
- [4] F. Bouesse, M. Renaudin, A. Witon, and F. Germain. A clock-less low-voltage aes crypto-processor. In *European Solid State Circuits Conference* (ESSCIRC 2005), pages 12–16, Grenoble, September 2005.
- [5] Bert den Boer & Kerstin Lemke & Guntram Wicke. A dpa attack against the modular reduction within a crt implementation od rsa. In *LNCS CHES 2002*, volume 2523, pages 228–243, 2002.
- F. P.-A. Ρ. J.-[6] Jean-F. Dhem, Koeune, Leroux, Mestre. Quisquater, and J.-L. Willems. Α practical implementa-In *CARDIS*, pages 167–182, 1998. tion of the timing attack. http://citeseer.nj.nec.com/dhem98practical.html.
- [7] Laurent Fesquet, Bertrand Folco, Mathieu Steiner, and Marc Renaudin. State-holding in look-up tables: application to asynchronous logic. In VLSI-SoC 2006, pages 12–17, Nice, France, October 16-18 2006. IEEE.

- [8] Laurent Fesquet, Jérôme Quartana, and Marc renaudin. Asynchronous systems on programmable logic. In *Reconfigurable Communication-centric SoCs (ReCoSoC)*, pages 105–112, Montpellier, France, June 27-29 2005.
- [9] B. Gao. A globally asynchronous locally synchronous configurable array architecture for algorithlm embeddings. PhD thesis, University of Edinburg, December 1996.
- [10] Jovan Dj. Golic. Multiplicative masking, power analysis of AES. http://citeseer.nj.nec.com/529351.html.
- [11] Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, and Renaud Pacalet. The "backend duplication" method. In *CHES-2005*, volume 3659 of *LNCS*, pages 383–397. Springer-Verlag, August 2005.
- [12] G. Hachez, F. Koeune, and J. J. Quisquater. Timing attack: what can be achieved by powerful adversary. In A. M. Barb/'e et. al., editor, 20th Symp. on Information Theory in the Benelux, pages 63–70, Haasrode (B), 27-28 1999. Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL). http://citeseer.nj.nec.com/hachez99timing.html.
- [13] Daniel H. Linder & James C. Harden. Phased logic: Supporting the synchronous desing paradigm with delay-insensitive circuitry. *IEEE transactions on computers*, 45(9):1031–1044, September 1996.
- [14] Scott Hauck, Gaetano Boriello, and Carl Ebeling. Montage: An fpga fo synchronous an dasynchronous circuits. In 2nd International Workshop on Field-Programmable Logic and Applications, Vienna, August 1992.
- [15] Quoc Thai Ho, J.-B. Rigaud, L. Fesquet, M. Renaudin, and R. Rolland. Implementing asynchronous circuits on lut based fpgas. In 12th International Conference on Field Programmable Logic and Applications (FPL), pages 36–46, Montpellier (La Grande-Motte), France, September 2002.
- [16] Philippe Hoogvorst, Sylvain Guilley, Sumanta Chaudhuri, Jean-Luc Danger, Alin Razafindraibe, Taha Beyrouthy, Laurent Fesquet, and Marc Renaudin. A Reconfigurable Cell for a Multi-Style Asynchronous FPGA. pages 15–22, June 2007. ReCoSoC, Montpellier, France.
- [17] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin. Fpga architecture for multi-style asynchronous logic. In *Design Automation and Test in Europe* (*DATE*), pages 32–33, München, Germany, March 7-11 2005.
- [18] K. Itoh, T. Izu, , and M. Takenaka. Address-bit differential power analysis on cryptographic schemes ok-ecdh, ok-ecdsa. In LNCS, pages 129–143, 2002.
- [19] Louis Goubin Jean-Sébastien Coron. On Boolean, arithmetic masking against differential power analysis. LNCS, 1965:231–??, 2001. http://citeseer.nj.nec.com/coron00boolean.html.

- [20] Kouichi Itoh & jun Yajima & Masahiko Takenaka & Naoya Torii. Dpa countermeasures by improving the window method. In *LNCS CHES* 2002, volume 2523, pages 303–317, 2002.
- [21] J. Kessels and P. Marston. Designing asynchronous standby circuits for a low-power pager. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 257–267, 1997.
- [22] P. Kocher, J. Jaffe, and B. Jun. Timing attacks on implementations of diffie-hellman, rsa, dsa, and other systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
- [23] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Timing attacks on implementations of diffie-hellman, rsa, dss., other systems. *LNCS*, 1109:104–113, 1996. http://citeseer.nj.nec.com/kocher96timing.html.
- [24] M. Renaudin L. Fesquet. A programmable logic architecture for prototyping clockless circuits. In *Field Programmable Logic (FPL)*, pages 293–298, Tampere, Finland, August 24-26 2005.
- [25] P.-Y. Liardet and N. P. Smart. Preventing spa/dpa in ecc systems using the jacobi form. In *LNCS CHES 2001*, volume 2162, pages 391–401, 2001.
- [26] Jacques Patarin Louis Goubin. Des, differential power analysis the *Duplication* method. http://citeseer.nj.nec.com/364562.html.
- [27] Kapilan Makeswaran and Venkatesh Akella. Pga-stc: Programmable gate array for implementating self-timed circuits. *International Journal of Elec*tronics, 84:255–267, 1998.
- [28] A.J. Martin, S.M Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus. The first asynchronous microprocessor: The test results. In *Computer Architecture News*, volume 17, pages 95–98, 1989.
- [29] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J.Hazewindus. Advanced Research in VLSI, chapter The design of an asynchronous microprocessor, pages 351–373. MIT Press, 1989.
- [30] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving smart card security using self-timed circuits. In *ASYNC'02*, pages 211–218, April 2002. Manchester, United King.
- [31] L.S. Nielsen, C. Niessen, and C.H. van Berkel. Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. *IEEE Transactions on VLSI Systems*, 2:391–397, 1994.
- [32] B. Jun P. Kocher, J. Jaffe. Differential power analysis: Leaking secrets. In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.

- [33] Robert Payne. Self Timed Field Programmable Gate Array Architectures. PhD thesis, University of Edinbugh, 1997.
- [34] 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.
- [35] M. Renaudin. Asynchronous circuits and systems: a promising design alternative. *Microelectronic Engineering*, 54:133–149, 2000.
- [36] M. Renaudin, P. Vivet, and F. Robin. Aspro-216: a standar cell q.d.i. 16-bit risc asynchronous microprocessor. In IEEE, editor, Proc. of the Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 22–31, 1998.
- [37] Jean-Baptiste Rigaud. Spécification de Bibliothèques pour la Synthèse de Circuits Asynchrones. PhD thesis, http://www.inpg.fr/, december 2002.
- [38] Werner Schindler. A timing attack against rsa with the chines remainder theorem. In *LNCS CHES 2000*, volume 1965, pages 109–124, 2000.
- [39] Jens Spars and Steve Furber, editors. Principles of Asynchronous Circuit Design: A Systems Perspective. Kluwer Academic Publishers, Boston / Dordrecht / London, 2001.
- [40] Daisuke Suzuki and Minoru Saeki. Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style. In CHES, volume 4249 of LNCS, pages 255–269, 2006. http://dx.doi.org/10.1007/11894063_21.
- [41] Taha Beyrouthy and Alin Razafindraibe and Laurent Fesquet and Marc Renaudin and Sumanta Chaudhuri and Sylvain Guilley and Philippe Hoogvorst and Jean-Luc Danger. A Novel Asynchronous e-FPGA Architecture for Security Applications. Dec 2007. FPT'07, Kokurakita, Kitakyushu, Japan.
- [42] Kouichi Itoh & Tetsuya Uzu & Masahiko Takenaka. Address-bit differential power analysis of cryptographic schemes ok-ecdh, ok-ecdsa. In *LNCS CHES 2002*, volume 2523, pages 129–143, 2002.
- [43] John Teifel and Rajit Manohar. Programmable asnchronous pipeline arrays. In *International Workshop on Field-Programmable Logic and Applications*, Lisbon, Portugal, September 2003.
- [44] John Teifel and Rajit Manohar. An asynchronous dataflow fpga architecture. *IEEE Transactions on Computers*, 53(11):1376–1392, 2004.
- [45] John Teifel and Rajit Manohar. Highly pipelined asynchronous fpgas. In 2th ACM International Symposium on Field-Programmable Gate Arrays, Monterey, CA, February 2004.

- [46] Tiri, Hwang, Lai, and Yang Schaumont Verbauwhede. Prototype ic with wddl, differential routing dpa resistance assessment. In B. Sunars J.R. Rao, editor, *LNCS CHES 2005*, volume LNCS 3659, pages 354–365, 2005.
- [47] Marc Joye & Christophe Tymen. Protections against differential analysis for elliptic curve cryptography —, algebraic approach —. In LNCS CHES 2001, volume 2162, pages 377–390, 2001.
- [48] T. Verhoeff. Delay-insensitive codes an overview. Distrib. Comput., 3:1–8, 1988.
- [49] J. Quisquater W. Schindler, F. Koeune. Unleashing the full power of timing attack, 2001.