2014 20th IEEE International Symposium on Asynchronous Circuits and Systems

# Semi-custom NCL Design with Commercial EDA Frameworks: Is it Possible?

Matheus Moreira*, Augusto Neutzling†, Mayler Martins†, André Reis†, Renato Ribas† and Ney Calazans*
*Pontifícia Universidade Católica do Rio Grande do Sul, †Universidade Federal do Rio Grande do Sul
{matheus.moreira, ney.calazans}@pucrs.br, {andreis, rpribas}@inf.ufrgs.br

***Abstract*—Quasi-Delay-Insensitive design is a promising solution for coping with contemporary silicon technology problems such as aggressive process variation and tight power budgets. However, one major barrier to its wider adoption is the lack of support for automated optimization techniques in semi-custom design flows. This paper proposes an innovative design flow that relies on the use of consolidated commercial EDA frameworks for synthesizing 1-of-n 4-phase Quasi-Delay-Insensitive circuits using Null Convention Logic. Accordingly, asynchronous gates, which are usually not supported by these frameworks, are modelled as conventional logic gates, allowing synthesis tools to perform static timing analysis and pre- and post- mapped design optimizations that can be specified by the designer using conventional timing constraints.**

***Index Terms*—Null Convention Logic, NCL, asynchronous design, technology mapping, automated synthesis**

## I. Introduction

Asynchronous design techniques are becoming an increasingly important topic for the VLSI research community. The abstraction provided by the synchronous paradigm starts to no longer justify the efforts to meet the constraints imposed by a global clock signal under aggressive process variations faced by ultra-deep submicron silicon technologies. Also, correctly distributing the clock signal is becoming a prohibitively expensive task, as the power dissipated by the clock tree can reach 45% of total power of a high-speed processor [1]. Differently from synchronous designs, asynchronous circuits can more easily tolerate process, voltage and temperature variations. Additionally, asynchronous circuits are naturally suited for low-power applications [2], [3]. By avoiding the use of a clock signal, these can employ more relaxed timing constraints, which allows to more efficiently cope with timing discrepancies. Moreover, due to the use of local handshaking [2] for control and sequencing of events, asynchronous logic is only active *when* and *where* required. Parts of the circuit can be quiescent while data flows only through the required path, inherently providing power savings and more easily coping with incoming dark silicon [4] related problems.

Albeit asynchronous circuits can be implemented using many different templates, Martin and Nyström [3] cite that practical circuits most often employ 1-of-n 4-phase Quasi-Delay-Insensitive (QDI) templates, as they allow easier design and timing closure and are more robust. In fact, the last decades have witnessed substantial developments in QDI design techniques. Accordingly, different computer-aided design (CAD) tools and design flows for automating QDI design, or at least part of it, are available in current literature, such as [5]–[16]. However, most of these tools perform logic optimizations only before technology mapping, because QDI circuits require specific gates rather than the ones available in conventional standard-cell libraries and are not directly compatible with consolidated synthesis tools. The drawback is that technology mapping is precisely the first step in the synthesis of a circuit that allows optimizations with realistic cost parameters of the target technology [8]. The only work that supports post-mapping optimizations [8] provides only basic optimizations with custom CAD tools, rather than exploring logic optimizations provided by consolidated commercial EDA Frameworks from vendors like Cadence or Synopsys. In fact, it is a common belief that commercial synchronous Frameworks cannot support QDI circuits direct mapping to standard-cells. In this way, efficient technology mapping of QDI circuits is still a gap to be filled.

The class of Null Convention Logic (NCL) gates [17], [18] allows efficient implementation of 1-of-n 4-phase QDI circuits as demonstrated by the Theseus Logic company through the design and fabrication of several chips [12]. In fact, all revised works on QDI design automation rely on the usage of NCL gates (consider that C-elements are special cases of NCL gates), except for the approach presented in [7] that uses dynamic logic templates. Here the authors consider the use of NCL for QDI design and seek to answer to if and how *"it is possible to employ commercial EDA frameworks to realize technology mapping and design optimization using NCL gates"*. To do so, Section II starts by introducing basic concepts on QDI and NCL. Section III presents an overview of a design flow that can be used with commercial EDA frameworks for mapping logic functions using NCL gates and prove the validity of the resulting circuit. Experimental results are the target of Section IV, together with a discussion for contextualizing this work in the state of the art. Finally, Section V draws conclusions and directions for future work.

## II. Background

### A. Quasi-Delay-Insensitive Design

The design of QDI circuits [2] [3] requires the choice of two basic items: (i) a Delay-Insensitive (DI) code coupled to a handshake-protocol and (ii) a logic style, which sometimes can actually be a mixture of logic styles. There are many ways to encode data in a DI manner. Even though new codes are

| Wire Name | Spacer | Bit ‘0’ | Bit ‘1’ |
|---|---|---|---|
| D.1 | 0 | 0 | 1 |
| D.0 | 0 | 1 | 0 |

(a)

| Wire Name | Spacer | Bit ‘0’ | Bit ‘1’ |
|---|---|---|---|
| D.1 | 1 | 1 | 0 |
| D.0 | 1 | 0 | 1 |

(b)

Fig. 1. 4-phase 1-of-2 data encoding for (a) RTZ and (b) RTO protocols

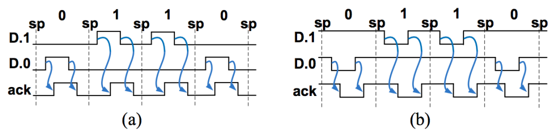


Fig. 2. 4-phase (a) RTZ and (b) RTO 1-of-2 data transmission where sp stands for the spacer

often suggested, the m-of-n class of codes and specifically 1-of-n codes, are widespread in VLSI design [3]. In 1-of-n codes, data is represented using n wires. Data validity is identified when exactly one of the n wires is at a given logic value and data absence can be marked by any of the $2^n - n$ other code words. A code word that marks data absence is called a spacer, as it separates two successive 1-of-n codes in data channels. Classically, the Return-to-Zero (RTZ) protocol is used, where n 0s are the spacer and valid code words are those with a single 1.

Figure 1(a) shows the RTZ 1-of-2 code, which uses two wires, called *D.1* and *D.0*, to carry a single bit of information. A 0 value is denoted by *D.0* at 1, and a 1 value by *D.1* at 1. In 1-of-n RTZ conventions, any code word with more than a wire at 1 represents invalid data. Figure 2(a) shows data transmission in a system using the RTZ protocol. Communication starts with all wires at 0. Next, the sender puts data in the channel (*D.0*, *D.1*) which is acknowledged by the receiver with the *ack* signal. After the sender receives the acknowledgement, it produces a spacer to end communication by putting all wires in the channel to 0. The receiver then lowers the *ack* signal, after which another communication can take place.

The Return-to-One (RTO) protocol [19] is similar to RTZ. The only difference is that data wire values are reversed. Figure 1(b) shows the conventions for the 1-of-2 RTO protocol. A spacer is represented by n wires in 1 (all-1s). Here, a 1 value is given by *D.1* at 0 and a 0 value by *D.0* at 0. As Figure 2(b) shows, differently from RTZ, RTO data transmission starts after the all-1s value is in the data channel. As soon as the sender puts valid data in the channel (*D.0*, *D.1*) the receiver may acknowledge it, by lowering the *ack* signal. Next, all data wires must return to 1 to denote a spacer, ending the transmission. When the spacer is detected by the receiver, it raises the *ack* signal and new data can follow. The idea behind the RTO protocol is simple, and albeit a 1-of-2 example is used here, any m-of-n code can support both protocols. Furthermore, an RTO-RTZ domain interface for a same m-of-n code requires exactly n inverters. As a generalization for m-of-n codes, an RTO *D.x* wire logic value can be translated from RTZ by:

$$\{x \in N | 0 \leq x \leq m-1\}, RTO(D.x) = \overline{RTZ(D.x)} \quad (1)$$

Here, expressions *RTO(D.x)* and *RTZ(D.x)* correspond to the logical values of each wire in the RTO and RTZ domains, respectively. In this way, according to Martin [3], the conversion of data from one domain to another is DI.

## *B. Null Convention Logic*

A threshold logic function (TLF) $t$ is a $n$-variable unate function that implements a Boolean function defined by a threshold value $T$ and a specific weight $w_i$ assigned to each variable $x_i$ such that:

$$t = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

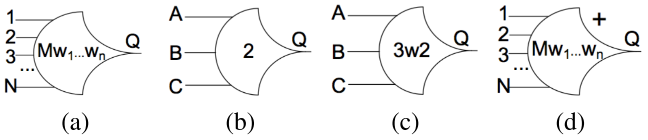


Fig. 3. Symbols for: (a) basic NCL gate, (b) 2 of 3 NCL gate, (c) 3 of 3 with weights (2, 1, 1) NCL gate and (d) basic NCL+ gate

In NCL design, basic components are sometimes called threshold gates, but this is imprecise. In fact, NCL gates couple a TLF [20] with *positive* integer weights assigned to inputs to the use of a hysteresis mechanism to guarantee a QDI compatible behavior. Figure 3(a) shows the NCL gate symbol: *N* is the number of gate inputs, *M* is either the gate threshold or a threshold function, and each input has weight $w_i$. Wherever no weight is specified, $w_i$=1 is assumed. Weights always come after the *w* specifier. The output switches to 0 when all *N* inputs are 0 and to 1 when the sum of weights for inputs at 1 reaches threshold *M*, or satisfies the threshold function. Otherwise, the previous output value is maintained. This enables QDI circuit design using 1-of-n data encoding and 4-phase handshaking, based on RTZ, as discussed in [17], [18]. In fact, the ON-set of an NCL gate is defined by the ON-set of a TLF and the OFF-set consists of the minterm corresponding to all inputs at 0.

For instance, say that the threshold of a 3-input NCL gate is 2 and all inputs have weight 1. In such a gate, showed in Figure 3(b), the output will only switch to 0 when all inputs are at 0 and to 1 when at least 2 of the inputs is at 1. Now, consider a 3-input (A, B, C) NCL gate with threshold 3, where the weight of the inputs is (2, 1, 1) respectively. In this gate, shown in Figure 3(c), the output will only switch to 0 when all inputs are at 0. However to switch to 1, input A necessarily needs to be at logic 1, together with any of the remaining inputs, B or C, to reach the function threshold. Note that for such gate only weights bigger than 1 are made explicit in the symbol.

The recently proposed NCL+ design style [21], is similar to NCL. However the assumption here is the use of RTO rather than RTZ. In fact, NCL+ gates also have a threshold *M* (written inside each gate symbol). However, as defined in Equation (1), the assumption of the RTO protocol mandates the switching function of an NCL+ gate to be the reverse of its NCL counterpart: the output will only switch to 1 when all inputs are at 1 and will only switch to 0 when threshold *M* is reached by the inputs at logic 0. For other combinations of inputs, the output keeps the previous value. The symbol that represents NCL+ gates appears in Figure 3(d). It is identical to that used for NCL, except for the "+" symbol on the top right corner. Note that NCL and NCL+ gates have the same functionality when the threshold is identical to the number of inputs. This special case implements the functionality of a C-Element [2], [17], [18]. Therefore the class of NCL gates comprises C-Element, required by several state-of-the-art QDI synthesis methods and tools as those mentioned in Section I.

Several ways to design NCL gates exist, like those explored by Parsan and Smith [22]. All can be employed to design NCL+ gates, *mutatis mutandis*. Yet, the classic static implementation is the most employed, due to power, area and speed characteristics. Therefore, throughout this work we assume the use of static implementations.

## III. Proposed Design Flow

Technology mapping is the process of translating a generic logic network into a technology-specific logic network. In other words, it is the task of transforming a netlist composed by generic logic functions into a netlist composed only by gates of a given technology library. However, conventional CAD tools cannot explicitly handle NCL and NCL+ gates, as these present sequential behavior. This Section proposes a design flow to overcome that and enable CAD tools to use NCL and NCL+ for synthesizing 1-of-n, 4-phase QDI circuits. Figure 4 provides an overview of the proposed design flow. The following subsections scrutinize the flow details.

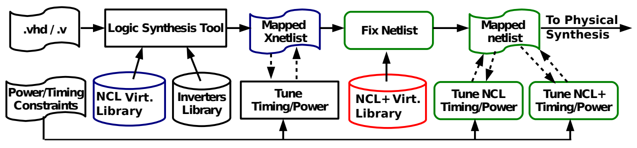


Fig. 4. Proposed Design Flow

### A. Definitions

First, a basic set of definitions helps understanding the proposed flow.

**Definiton 1 (Boolean virtual function)**: A Boolean virtual function (BVF) is a completely specified function $f$ that implements a TLF with:

$$f : \{0,1\}^n \rightarrow \{0,1\} \tag{3}$$

**Definiton 2 (Boolean virtual functions library)**: A BVF library (*BVFL*) is a 5-tuple

$$BVFL =< G, Z, O, g, h > \tag{4}$$

where *G* is a set of BVFs, *Z* is a set of NCL gates, *O* is a set of NCL+ gates and *g* and *h* are completely specified functions such that $g : G \rightarrow Z$ and $h : G \rightarrow O$.

In practice, a BVF library is split in two standard-cell libraries: *NCL Virtual Library* and *NCL+ Virtual Library*. These contain the physical implementation of *Z* and *O*, respectively. To enable the use of these libraries in conventional CAD tools, masking of the cells sequential behavior occurs by defining the equivalent BVF for each NCL and NCL+ gate. This is done by modifying the Liberty (.lib) file of the library, editing the logic functions of gates and replacing them by the correspondig BVFs. In this way, the physical implementation at the standard-cell level of specific NCL and NCL+ gates must be available for each BVF.

### B. NCL Gate Identification

To define the NCL gate that implements a specific BVF, we start with a typical logic function that is understood by CAD tools. Next, we use the method proposed by Neutzling *et al.* in [23] to identify the TLF. For instance, assume the example function used in [23]: $Q = A(B + C)$. Table I shows the associated truth table. First, we verify if the function is a TLF and can be represented as a BVF, as specified in Definition 1. Next, inequalities are computed from the relationship between input weights and the gate threshold to a TLF, as defined in Equation (2). Input weights and the threshold value can be computed from these inequalities. The relationship between input weights and the gate threshold to a TLF for BVF $Q = A(B + C)$ is shown in the rightmost column of Table I. Each sum of input weights greater than the threshold belongs to the *greater side set* of inequalities and each sum of weights which is less than the threshold belongs to the *lower side set*. Table II shows the sets for the BVF of Table I.

TABLE I
Input weights and threshold relationship to Q=A(B+C).

| A | B | C | Q | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $0 < T$ |
| 0 | 0 | 1 | 0 | $w_C < T$ |
| 0 | 1 | 0 | 0 | $w_B < T$ |
| 0 | 1 | 1 | 0 | $w_B + w_C < T$ |
| 1 | 0 | 0 | 0 | $w_A < T$ |
| 1 | 0 | 1 | 1 | $w_A + w_C \geq T$ |
| 1 | 1 | 0 | 1 | $w_A + w_B \geq T$ |
| 1 | 1 | 1 | 1 | $w_A + w_B + w_C \geq T$ |

TABLE II
Greater and smaller sets of inequalities from Table I.

| greater side | | lower side |
|---|---|---|
| $w_A + w_C$ | $> T >$ | 0 |
| $w_A + w_B$ | $> T >$ | $w_C$ |
| $w_A + w_B + w_C$ | $> T >$ | $w_B$ |
| | $T >$ | $w_B + w_C$ |
| | $T >$ | $w_A$ |

Since greater side set elements are greater than the threshold value, and lower side set elements are smaller than such threshold, each greater side element is greater than each lower side element. Accordingly, the inequalities system are generated by performing the Cartesian product of the greater side set and the lower side set, as showed in Table III. Some relationships in Table II are not useful. For instance, since we have the relation $w_A + w_B \geq T$ and the input weights are always positive, the relation $w_A + w_B + w_C \geq T$ is redundant, since is contained in the former. In the method of Neutzling et al. [23], these redundancies are avoided. In fact for the example $Q = A(B + C)$ the algorithm creates only the inequalities 4, 5, 9 and 10 from Table III. Moreover, besides generating only irredundant inequalities, the algorithm simplifies each of them when possible. The simplification occurs when the variable weight appears on both sides of the same inequality. When this happens, the variable is removed from such inequality. Considering for instance inequality 4. It is simplified removing *C*, resulting on a new inequality $w_A > w_B$. Since all input weights are positive, the algorithm also discards the inequalities having no weight (or 0) in the smaller side as these inequalities are not useful.

TABLE III
GENERATED INEQUALITIES FROM TABLE II

| **Cartesian product: greater side × lower side** | | | |
|---|---|---|---|
| **1** | $w_A + w_C > 0$ | **9** | $w_A + w_B > w_B + w_C$ |
| **2** | $w_A + w_C > w_C$ | **10** | $w_A + w_B > w_A$ |
| **3** | $w_A + w_C > w_B$ | **11** | $w_A + w_B + w_C > 0$ |
| **4** | $w_A + w_C > w_B + w_C$ | **12** | $w_A + w_B + w_C > w_C$ |
| **5** | $w_A + w_C > w_A$ | **13** | $w_A + w_B + w_C > w_B$ |
| **6** | $w_A + w_B > 0$ | **14** | $w_A + w_B + w_C > w_B + w_C$ |
| **7** | $w_A + w_B > w_C$ | **15** | $w_A + w_B + w_C > w_A$ |
| **8** | $w_A + w_B > w_B$ | | |

After simplification, input weights are computed by selecting inequalities with only one input in the greater side. A single input in the greater side denotes that this input must have larger weight than all inputs in the lower side. From example $Q = A(B + C)$, the selected inequalities are only $w_A > w_B$ and $w_A > w_C$. Therefore, the weight from the greater side (input A in both cases) is defined as the key of the inequality. Next, a temporary variable is created, which controls the value assigned to the weights. Such temporary variable is initialized with a minimum value (being initially set to 1) and is increased by one during the iterations. The order of weight assignment starts from variables that have the lowest weight, and continues in ascending order. Each time a weight is assigned, the consistency of the corresponding inequalities is checked. If the current value of the temporary variable satisfies the inequalities, this value is indeed the weight of the variable. Otherwise, the value is increased to satisfy the inequalities or until an upper limit is reached. This procedure is repeated for each variable.

In the example $Q = A(B + C)$, the assignment is first done to the weights of inputs B and C. Since there are no inequalities constrains for these keys, value 1 is assigned. The temporary variable increases to 2. Then, the weight of A must be assigned. For this key there are two identical inequalities: $w_A > 1$. The method assigns the temporary value 2 and checks the inequalities. The inequalities are true since $2 > 1$. As all weights are already assigned, this step finishes. Therefore, the weights assigned for the BVF $Q = A(B + C)$ are $w_A = 2$, $w_B = 1$ and $w_C = 1$. According to the authors of [23], this bottom-up approach ensures that the weights assigned by the algorithm are always the minimum possible, allowing the gate implementation be synthesized using minimum circuit area. The threshold value is defined as the sum of weights of the smallest value of the greater side. For $Q = A(B + C)$, the calculated threshold is 3, from $w_A + w_B$ or $w_A + w_C$ (refer to Table III). Accordingly, this defines an *NCL3W2-of-3* gate.

As Table IV shows, the selected NCL gate for the example BVF has the same ON-set (in red). On the other hand, the OFF-set, in blue for the BVF, is not the same. In NCL gates, the output will switch to 0 only when all inputs are at 0, as the table shows in green. In fact, the ON-set of NCL gates signals valid data, which generates varying logic values in the inputs of such gates. However, because we target 1-of-n 4-phase QDI circuits, it is guaranteed that between each valid data there will be a spacer, i.e. all wires will go to 0. Therefore, the OFF-set of NCL gates synchronizes spacers, guaranteeing that there is no early spacer generation. This ensures that the 1-of-n 4-phase QDI properties are respected. For instance, assume that the NCL3W2-of-3 gate signals 1 in its output, which indicates valid data to the next gate. This output may only switch back to 0 when a spacer is detected, i.e. when all its inputs are at 0.

BVFs can also be inverted logic functions and can be mapped to an equivalent NCL gate. However, in this case, the algorithm proposed by Neutzling *et al.* must be applied searching for the OFF-set of the BVF. Take for example a 2-input NAND ($Q = \overline{AB}$). By applying the algorithm, we find the threshold (T=2) and the weight of the inputs ($w_A$=1 and $w_B$=1), which corresponds to an *inverted NCL2-of-2* gate. The truth table of the BVF and the NCL gate are shown in Table V. As the table shows, the gate also respects QDI definitions and will only switch its output to 1 when a spacer is detected (all inputs at 0) and the OFF-set is the same as that of the BVF, ensuring that valid data will propagate correctly. Note that this gate has an inverted output.

TABLE IV
EQUIVALENT NCL AND NCL+ GATES FOR BVF Q=A(B+C).

| $A$ | $B$ | $C$ | $Q_{BVF}$ | $Q_{NCL}$ | $Q_{NCL+}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | - | 0 |
| 0 | 1 | 0 | 0 | - | 0 |
| 0 | 1 | 1 | 0 | - | 0 |
| 1 | 0 | 0 | 0 | - | 0 |
| 1 | 0 | 1 | 1 | 1 | - |
| 1 | 1 | 0 | 1 | 1 | - |
| 1 | 1 | 1 | 1 | 1 | 1 |

TABLE V
EQUIVALENT NCL AND NCL+ GATES FOR A NAND BVF.

| $A$ | $B$ | $Q_{BVF}$ | $Q_{NCL}$ | $Q_{NCL+}$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | - | 1 |
| 1 | 0 | 1 | - | 1 |
| 1 | 1 | 0 | 0 | 0 |

## C. Technology Mapping using NCL Gates

Using a defined set of BVFs and generating the equivalent NCL gates, an *NCL Virtual Library* can be generated. Note that an automated flow exists for generating NCL gates in the standard-cell level, with automatic electrical characterization, as presented in [24]. The generated *NCL Virtual Library* is compatible with commercial IC design frameworks. In fact, the input of the proposed design flow is an *NCL Virtual Library* together with a library containing inverters and buffers (as these gates do not jeopardize QDI properties) and a behavioral Verilog or VHDL description of a 1-of-n 4-phase QDI circuit, as showed in Figure 4. Such descriptions can be output of some state-of-the-art asynchronous synthesis framework like the ones listed in Section I with little modifications, like replacing NCL gates for their respective BVF, so that the tool will be able to perform optimizations. Additionally, albeit the presented design flow can be employed in similar tools, throughout this work we assume the use of RTL Compiler.

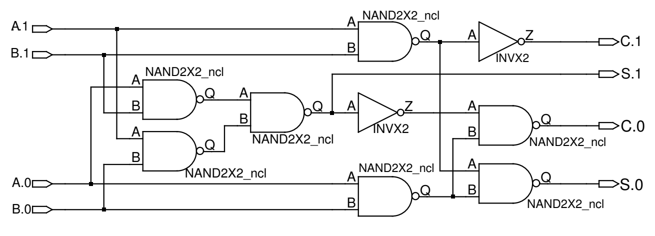
So far, the proposed design flow is not exactly a novelty. Many works on the state-of-the-art perform template-based mapping, as presented in [11]. However, what kept designers from using commercial CAD optimization algorithms was the fact that for some logic optimizations, the resulting netlist after being mapped to an *NCL Virtual Library* can be corrupted. Take for example the following behavioral Verilog description of a 1-of-2 4-phase QDI 2-input half adder:

```
1 module HA (A.1, A.0, B.1, B.0, C.1, C.0, S.1, S.0);
2  input  A.1, A.0, B.1, B.0;
3  output C.1, C.0, S.1, S.0;
4  wire   C.1, C.0, S.1, S.0;
5  assign S.1=(A.1 & B.0)|(A.0 & B.1);
6  assign S.0=(A.1 & B.1)|(A.0 & B.0);
7  assign C.1=(A.1 & B.1);
8  assign C.0=(A.1 & B.0)|(A.0 & B.0)|(A.0 & B.1);
9 endmodule
```

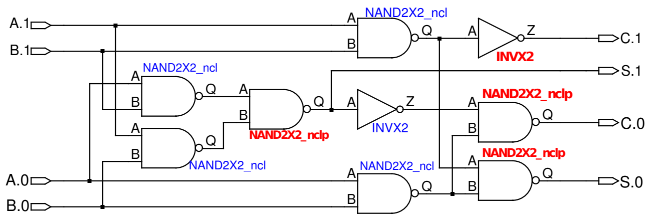
Now, assume that we feed the synthesis tool only with an *NCL Virtual Library* containing only two-inputs NANDs, together with a library containing only inverters the minimum set of gates. Applying de Morgan and other Boolean equivalence laws, synthesis tools can convert the sums of products of lines 5, 6 and 8 to a set of NANDs. In fact, this is the result we obtained for the example source code, as showed in Figure 5(a). Note that in this step, the netlist is already mapped to NCL gates of the target technology. Also note that *NCL Virtual Library* gates will always have a *_ncl* suffix to denote that they implement NCL logic. Recalling Figure 4, at this point, the designer can already provide timing and power constraints to the tool so it can optimize the design accordingly. Note that this can be done because static timing analysis (STA) is enabled by the use of functionalities known by the tool. To do so, the designer can use conventional timing constraints such as *set max_delay*. However, it is important to keep in mind that this netlist may have corrupted the correct functionality of the circuit. For this reason, at this stage, we call it *Xnetlist*. For instance, analyzing the circuit of Figure 5(a), assume that all inputs are at 0 (spacer), making all outputs also go to 0. Now, assume that inputs A.1 and B.1 switch to 1, signaling a 1 value in A and B. Accordingly, C.1 will switch to 1, signaling a 1 value in the carry out signal (C). However S.0 will not switch to 1, which denotes that the circuit functionality has been corrupted. This is because the BVF of the gate that generates this output, a NAND, has an OFF-set that is not covered by its corresponding NCL gate, the actually mapped gate. This problem arises when inverted logic is used. In fact, unless no inverters are provided for the tool, one cannot guarantee that the functionality will not be compromised. As classically there was no way to guarantee the coverage of BVFs ON-set of inverted functions using NCL gates, because such functions alter the domain from RTZ to RTO as defined in Equation (1), this was one of the major barriers that prevented designers to use conventional CAD tools to map QDI circuits.

### D. Netlist Fixing using NCL+ Gates

This was true until recently, with the proposition of NCL+ [21]. In fact, NCL+ gates can be employed for covering the OFF-set and ON-set of non-inverted and inverted BVFs, respectively. Also, recalling Definition 2, together with an *NCL Virtual Library* ($Z$), a BVF library also requires an *NCL+ Virtual Library* ($O$). For generating the latter, the same method



(a)



(b)

Fig. 5. Example of *Xnetlist* with only two inputs NAND BVFs and inverters (a) and the netlist after fix steps (b).

of Neutzling *et al.* can be employed, but searching for inputs that are at 0 when defining the input weights and gate threshold relationships. This means that for BVF $Q = A(B+C)$ we can obtain the greater and lower side sets of inequalities showed in Table VI.

TABLE VI
NCL+ GREATER AND SMALLER SETS OF INEQUALITIES FROM TABLE II

| **greater side** | | **lower side** |
|---|---|---|
| A+B+C | $> T >$ | B |
| A+B | $> T >$ | C |
| A+C | $> T >$ | 0 |
| A | $> T$ | |
| B+C | $> T$ | |

Employing the algorithm, this leads to a threshold $T$=2 and weights $w_A$=2, $w_B$=1 and $w_C$=1. In this way, the NCL+ gate for BVF $Q = A(B+C)$ (complementary for an NCL3W2-of-3) is the NCL+2W2-of-2. Accordingly, the truth table of such a gate appears in Table IV. As it can be seen, this gate presents the same OFF-set of the BVF. Moreover, it guarantees that the output will only switch to 1 when all inputs are at 1, allowing synchronization of RTO spacers. Similarly, NCL+ gates can be defined for inverted BVFs, like the two inputs NAND. For this function, the corresponding gate is an inverted NCL+1-of-2 ($T$=1, weights $w_A$=1 and $w_B$=1). This covers the ON-set of the BVF, as Table V shows. Therefore, taking for instance the two-input NAND BVF of a given BVF library, one must guarantee the availability of an NCL2-of-2 and an NCL+1-of-2 gates, as specified in Definition 2.

Because NCL and NCL+ gates can be complementary for a same BVF, together the correct gates cover both the ON-set and the OFF-set of the BVF, as Tables IV and V show. In fact, in the definition of NCL gates for a given BVF we look for 1s in the inputs for generating the inequalities, while in the definition of NCL+ gates, we look for 0s in the inputs. Accordingly, if an NCL gate presents the same ON-set of a

BVF, there is an NCL+ gate that presents the same OFF-set and vice-versa.

Using an *NCL+ Virtual Library*, a corrupted *Xnetlist* can be fixed. From Equation (1), whenever a signal from an RTZ domain is inverted, it becomes an RTO signal and vice-versa. Furthermore, by definition, NCL gates assume the usage of RTZ, while NCL+ gates assume the usage of RTO. Therefore the inputs of NCL gates must always belong to the RTZ domain and the ones of NCL+ from the RTO domain. Therefore, because NCL+ gates provide the complementary BVF functionality of NCL gates, if an NCL gate of an *Xnetlist*, has inputs from the RTO domain, it must be replaced by its complementary NCL+ gate from a *NCL+ Virtual Library*. In this way, valid data will always be correctly computed through either the OFF-set or the ON-set of each BVF (refer to Tables IV and V). Fixing an *Xnetlist* can be done using a very simple algorithm:

```
for each gate of the Xnetlist:
 if current gate is NCL and # of inversions is odd:
    replace by complementary NCL+
```

For each NCL gate of the *Xnetlist*, the algorithm verifies if the number of inverted gates from an input is odd. If the condition is true, this means that the gate has RTO inputs and it must be replaced by the respective NCL+ gate. Note that any input can be used for this verification, as the result will always be the same. This is guaranteed by the implementation, given that 1-of-n 4-phase QDI design will always be implemented based on the propagation of valid data, because spacers synchronization is guaranteed by the gates that implement the circuit. This algorithm was implemented using TCL scripts and integrated in the RTL Compiler environment for automatically fixing *Xnetlists*, given that a *NCL+ Virtual Library* is provided, as showed in Figure 4. The result of this fix is a functional mapped *netlist*. For instance, the *Xnetlists* of Figure 5(a), after being fixed using the algorithm, would generate the correct *netlist*, as showed in Figure 5(b). Note that incorrectly placed NCL gates were changed by their complementary NCL+ gates, denoted by an *_nclp* suffix. That said, depending of the specified constraints, different driving strengths may have been selected for generating an *Xnetlist*. Our automation scripts select similar driving strengths for the corresponding NCL+ gates that will substitute incorrect NCL ones. Even though, this can change performance figures.

### E. Post-fix Optimizations

The post-fix netlist may also need to be optimized. To do so, the designer must optimize NCL and NCL+ gates separately, because they all share the same BVF in their *.lib* file and the synthesis tool may end up replacing NCL gates by NCL+ or vice-versa in wrong places. In this way, the designer needs to disable either the *NCL Virtual Library* or the *NCL+ Virtual Library*. In RTL Compiler this is done by defining the attribute *preserve* to *true* to all gates of the library. With that, the tool is able to perform STA and optimize the design to meet the specified constraints without compromising the correct functionality of the design. As Figure 4 shows, tuning NCL and NCL+ gates can be done iteratively until required constraints are met. Next, formal verification can be performed as usual, as all BVFs are known by conventional tools. For simulation, the employed behavioral models must reflect the functionality of the actual NCL and NCL+ gates, rather than BVFs. In this way it is possible to verify that QDI properties were not compromised. After the logic synthesis is completed and verified, the mapped netlist can be employed by physical synthesis tools for generating the layout of the circuit. Note that all libraries employed in the logic synthesis must be provided for the physical synthesis tools. However optimizations in the design must be done separately for NCL and NCL+ gates, like in the logic synthesis. Layout verification steps can be performed as usual. Therefore, given a behavioral Verilog or VHDL description of a 1-of-n 4-phase QDI circuit, the adoption of the design flow described herein produces a mapped netlist functionally equivalent to the provided input description that respects 1-of-n 4-phase QDI definitions.

## IV. Experiments and Discussion

### A. Experimental Setup and Results

A set of case study circuits was synthesized using the proposed design flow: an 8-bit ripple carry adder (8bRC), an 8-bit Kogge Stone adder (8bKS), a 32-bit Kogge Stone adder (32bKS), a 32-bit Arithmetic Logic Unit (32bALU) and a 16-bit shift and sum multiplier (16bMULT). The choice for such case studies was due to the fact that they present different gate counts and, more importantly, different numbers of gates in series in the critical path, which allows displaying the functionality of the proposed design flow for different logic depths. Synthesis targeted the STMicroelectronics 65nm bulk CMOS process and all models were based on typical corners. For simplicity, apart from the library of inverters, with tens of different driving strengths, we employed a minimum BVFL, containing only the NAND BVF. NCL and NCL+ Virtual libraries had gates with 6 different driving strengths, for allowing timing optimizations, and are a subset of the ASCEnD library [24]. A library containing only the NAND BVF is useful at this stage as it will generate long logic paths, stressing the proposed design flow.

After synthesis, we exported the netlist and the annotated delay to perform timing simulation. Simulation allowed us to evaluate the correctness of the design and to perform performance measurements in terms of forward propagation delay, the time it takes for valid data to propagate from the inputs to the outputs, and transmission delay, the time it takes for both data and spacer to propagate through the circuit. Simulations were performed for 5 scenarios, all simulated for 1ms: (i) the circuit was filled of spacers and there was no data injection (0%) and (ii), (iii), (iv) and (v) the circuit was operating with random data being injected 25%, 50%, 75% and 100% of the time. During simulation internal signals activity was annotated and used as the source to conduct static power analysis. In this way, (i) allowed us to measure quiescent power and the remaining scenarios provided results on how the power of the case studies increased as activity increased.

TABLE VII
CASE STUDIES SYNTHESIS, SIMULATION AND POWER ANALYSIS RESULTS.

| | **Gates** | **% NCL** | **% NCL+** | **% INV** | **# CP** | $\mathbf{D_{Trans.}}$ | $\mathbf{D_{Fwd.}}$ | $\mathrm{Pwr.}_{0\%}$ | $\mathrm{Pwr.}_{25\%}$ | $\mathrm{Pwr.}_{50\%}$ | $\mathrm{Pwr.}_{75\%}$ | $\mathrm{Pwr.}_{100\%}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8bRC | 247 | 41 | 27 | 32 | 60 | 7.71 ns | 3.9 ns | 0.014 mW | 2.926 mW | 5.835 mW | 8.744 mW | 11.649 mW |
| Fast 8bRC | 282 | 35 | 28 | 37 | 48 | 2.72 ns | 1.41 ns | 0.021 mW | 4.261 mW | 8.496 mW | 12.736 mW | 16.964 mW |
| Improvement | -14% | 15% | -4% | -16% | 20% | 65% | 64% | -50% | -46% | -46% | -46% | -46% |
| 8bKS | 422 | 49 | 27 | 24 | 24 | 4.41 ns | 2.33 ns | 0.030 mW | 5.436 mW | 10.837 mW | 16.239 mW | 21.631 mW |
| Fast 8bKS | 521 | 41 | 24 | 35 | 18 | 3.07 ns | 1.55 ns | 0.041 mW | 11.002 mW | 21.957 mW | 32.903 mW | 43.840 mW |
| Improvement | -23% | 16% | 11% | -46% | 25% | 30% | 33% | -37% | -103% | -103% | -103% | -103% |
| 32bKS | 2942 | 48 | 26 | 26 | 36 | 5.13 ns | 2.74 ns | 0.124 mW | 21.392 mW | 42.641 mW | 63.892 mW | 85.104 mW |
| Fast 32bKS | 3446 | 42 | 22 | 36 | 26 | 3.84 ns | 1.96 ns | 0.249 mW | 59.503 mW | 118.700 mW | 177.897 mW | 237.009 mW |
| Improvement | -17% | 13% | 15% | -38% | 28% | 25% | 28% | -101% | -178% | -178% | -178% | -178% |
| 32bALU | 4924 | 45 | 23 | 32 | 44 | 2.27 ns | 1.23 ns | 0.334 mW | 36.577 mW | 72.794 mW | 109.077 mW | 145.143 mW |
| Fast 32bALU | 5141 | 40 | 22 | 38 | 34 | 1.56 ns | 0.84 ns | 0.371 mW | 63.171 mW | 125.770 mW | 188.358 mW | 250.870 mW |
| Improvement | -4% | 11% | 4% | -19% | 23% | 31% | 32% | -11% | -73% | -73% | -73% | -73% |
| 16bMUL | 18371 | 42 | 25 | 33 | 432 | 53.68 ns | 27.02 ns | 1.188 mW | 10.345 mW | 19.474 mW | 28.646 mW | 37.791 mW |
| Fast 16bMUL | 21041 | 39 | 22 | 39 | 216 | 40.73 ns | 20.53 ns | 1.529 mW | 18.315 mW | 35.087 mW | 51.862 mW | 68.569 mW |
| Improvement | -15% | 7% | 12% | -18% | 50% | 24% | 24% | -29% | -77% | -80% | -80% | -81% |

Table VII provides an overview of the obtained results for a synthesis with relaxed timing constraints and strict timing constraints (lines with the word Fast before the name of the case study). Note that the latter employed constraints iteratively defined in order to achieve maximum speed possible with the provided BVFL. Definitions of the constraints were done defining *max_delay* attributes. Also, case studies presented different gates count: from 247 to 18371 for relaxed constraints. From these, always more than 40% were NCL gates and less than 30% NCL+ gates. The remaining were inverters required by the logic. Note that the increased percentage of inverters in the design is a consequence of providing only a NAND BVF. As constraints were strict, gate count increased up to 23% and the percentage of NCL and NCL+ gates decayed, a consequence of the logical optimizations that the tool performed and the possible insertion of extra inverters. The table also shows the number of gates in the critical path (# CP), which varied from 24 to 432 for the relaxed timing designs. Note that as timing constraints were strict, substantial reductions on the critical path were observed, up to 50% in the case of *16bMUL*. This demonstrated that the design flow is capable of allowing the synthesis tool to perform logic optimizations according to the specified constraints.

A direct consequence of that is observed in the $D_{Trans.}$ column of the table. Accordingly, as we strict timing constraints, is substantially reduced, up to 65%. Another interesting result was that forward propagation delay ($D_{Fwd.}$) always kept around 50% of transmission delay for all designs. This is a very important aspect for asynchronous circuits, as a short $D_{Fwd.}$ allows starting communication with interconnected modules earlier. Moreover, if one desires to reduce forward propagation delay, is just a matter of tuning the NCL and NCL+ gates that compose the employed libraries, unbalancing low to high and high to low delays. Finally, power results help demonstrating how QDI circuits are very well suited for low power applications. As the table shows, power scales as the activity in the circuit is increased. This is due to a natural characteristic of such circuits: modules are active only *when* and *where* required. Also, note that as timing constraints were strict, the case studies presented increase in power. In this way, the proposed design flow allows designers to trade-off performance figures in a 1-of-n 4-phase QDI design.

For verification sake, we also picked the 16bMUL circuit and introduced all combinations of valid data and spacer in the inputs. This allowed us to validate that the circuit would only generate valid data/spacers in the output after all inputs had valid data/spacers, respecting QDI principles.

## B. Contributions of this Work

Related works either employ custom made tools that map the design using template-based approaches or impose severe restrictions to the synthesis tool that avoids taking advantage of logic optimization algorithms. Albeit many articles about asynchronous circuits synthesis are available on current literature, we selected a set that supports the same class as our design flow. A common problem with most of these [8]–[10], [12]–[14], is the fact that they focus on pre-mapping synthesis optimizations and then translate the generic netlist to a mapped netlist using a template-based approach that relies on the usage of NCL gates, which is not effective. An important issue raised by Cheoljoo and Nowick in [8] is the fact that optimizations during synthesis may introduce orphans in QDI circuits. Fortunately, this can be solved by our design flow by employing an initial DIMS-like circuit description [8] and avoiding optimizations of multi-input C-elements. Designing scripts for automating this task is ongoing work.

Another flow, presented by Kondratyev and Lwin in [11] presents an optimization of template-based methods that allow some post-mapping optimization. In fact, the approach is quite similar to the one proposed in this manuscript. For the synthesis steps, NCL gates are represented by their set Boolean functions so that the synthesis tool can use them. The drawback is that because the authors employ only NCL gates, they are able to optimize only the set phase of functions. Thus, these authors assume that all functions are non-inverted and use only the ON-sets for performing logic optimizations. This clearly avoids taking advantage of more efficient optimizations, which are supported by our design flow through the use of NCL+ gates. In addition, it is not clear how the authors cope with inverted logic that can be inserted in the design, which as we demonstrated here can jeopardize correct functionality. In fact, the possibility of having the complement of an NCL gate, defined as an NCL+ gate by Definition 2, is the core of the proposed design flow. This explains why it was not possible to use commercial tools for mapping NCL gates until the advent of NCL+. Also, note that basic functions like AND, NAND, OR and NOR can all be implemented as TLFs and can be used to implement any logic circuit. Therefore, having only one of these basic functions as a BVF guarantees the possibility of implementing any 1-of-n 4-phase QDI circuit description.

This paper stands off by proposing a design flow for 1-of-n 4-phase QDI synthesis based on NCL that for the first time uses IC design commercial frameworks features extensively. This allows taking advantage of well defined optimization algorithms rather than relying in template-based approaches and optimizations. Salient features of the method include to enable the exploration of STA and pre- and post-mapping optimizations. Furthermore, the flow is capable of mapping behavioral HDL into inverted and/or non-inverted NCL and NCL+ gates, which allows generating shorter logic paths than the ones generated when using only non-inverted NCL gates. Another important aspect is that it allows exploring timing/power/area optimizations with conventional constraints descriptions and allows post-synthesis optimizations. It also allows handling arbitrary networks with high fanin and complex gate types. Moreover, albeit in this work we employed static NCL implementations, the proposed flow allows automatic choice for different NCL topologies when these are available in the BVFL. In fact, the synthesis tool decides the best trade-offs based on power/timing/area models and constraints. Another major contribution is the definition of a systematic approach for implementing NCL and NCL+ Virtual Libraries to use in the proposed design flow. Finally, a very important aspect of the proposed flow is that it can be used together with previously proposed design flows, allowing post mapping optimizations for QDI circuits generated with them. Hence, the flow can be a complement for existing synthesis environments.

## V. Conclusions

This paper demonstrated that usual synchronous commercial EDA Frameworks can indeed be employed for mapping logic functions into NCL gates and performing logic optimizations. We have introduced a design flow that automates the task of implementing 1-of-n 4-phase QDI circuits using commercial Frameworks intended for synchronous designs. The results we obtained using the flow demonstrate that conventional tools can indeed be efficiently used to optimize circuits composed of NCL gates. As future work, we will explore the automatic translation from conventional combinational circuits to 1-of-n 4-phase QDI and also submit a test-chip for fabrication to validate the result of our flow on silicon. Finally, it is also future work the development of scripts for enabling STA for sequential designs. To do so, the method proposed here can be performed iteratively disabling feedback loops.

## VI. Acknowledgements

Authors acknowledge the support of CNPq under grants 401839/2013-3, 200147/2014-5 and 310864/2011-9 and the support of FAPERGS under grant 11/1445-0.

## References

[1] V. Tiwari, D. Singh, and S. Rajgopal, “Reducing power in high-performance microprocessors,” *Proceedings of the 35th Design Automation Conference*, pp. 732–737, 1998.

[2] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[3] A. J. Martin and M. Nystrom, “Asynchronous techniques for system-on-chip design,” *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[4] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in *Proceeding of the 38th annual international symposium on Computer architecture*. ACM Press, 2011, p. 365.

[5] A. Bardsley, “Balsa: An asynchronous circuit synthesis System,” Ph.D. dissertation, University of Manchester, 1998.

[6] A. Bardsley, L. Tarazona, and D. Edwards, “Teak: A token-flow implementation for the balsa language,” in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, 2009, pp. 23–31.

[7] P. A. Beerel, G. D. Dimou, and A. M. Lines, “Proteus: An asic flow for ghz asynchronous designs,” *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 36–51, 2011.

[8] J. Cheoljoo and S. Nowick, “Technology mapping and cell merger for asynchronous threshold networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 659–672, 2008.

[9] I. David, R. Ginosar, and M. Yoeli, “An efficient implementation of boolean functions as self-timed circuits,” *Computers, IEEE Transactions on*, vol. 41, no. 1, pp. 2–11, 1992.

[10] B. Folco, V. Brégier, L. Fesquet, and M. Renaudin, “Technology mapping for area optimized quasi delay insensitive circuits,” in *IFIP Conference on Very Large Scale Integration Systems*, 2005, pp. 55–69.

[11] A. Kondratyev and K. Lwin, “Design of asynchronous circuits using synchronous cad tools,” *IEEE Design Test of Computers*, vol. 19, no. 4, pp. 107–117, 2002.

[12] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev, “Asynchronous design using commercial HDL synthesis tools,” in *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2000, pp. 114–125.

[13] F. A. Parsan, W. K. Al-assadi, and S. C. Smith, “Gate mapping automation for asynchronous null convention logic circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, p. 14, 2013.

[14] R. Reese, S. Smith, and M. Thornton, “Uncle - an rtl approach to asynchronous design,” in *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2012, pp. 65–72.

[15] N. P. Singh, “A design methodology for self-time systems,” Cambridge, MA, USA, Tech. Rep., 1981.

[16] K. Van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, “The vlsi-programming language tangram and its translation into handshake circuits,” in *Proceedings of the European Conference on Design Automation*, 1991, pp. 384–389.

[17] K. Fant and S. Brandt, “Null convention logic$^{TM}$: a complete and consistent logic for asynchronous digital circuit synthesis,” in *Proceedings of International Conference on Application Specific Systems, Architectures and Processors*, 1996, pp. 261–273.

[18] K. M. Fant, *Logically Determined Design*. Hoboken, NJ: Wiley, 2005.

[19] M. Moreira, R. Guazzelli, and N. Calazans, “Return-to-one protocol for reducing static power in qdi circuits employing m-of-n codes,” in *25th Symposium on Integrated Circuit and Systems Design*, 2012.

[20] S. L. Hurst, “An introduction to threshold logic: A survey of present theory and practice,” pp. 339–351, 1969.

[21] M. Moreira, C. Oliveira, R. Porto, and N. Calazans, “Ncl+: Return-to-one null convention logic,” in *IEEE International Midwest Symposium on Circuits and Systems*, 2013, pp. 836–839.

[22] F. Parsan and S. Smith, “Cmos implementation comparison of ncl gates,” in *Int. Midwest Symp. on Circ. and Systems*, 2012, pp. 394–397.

[23] A. Neutzling, M. G. Martins, R. P. Ribas, and A. I. Reis, “Synthesis of threshold logic gates to nanoelectronics,” in *26th Symposium on Integrated Circuits and Systems Design*, 2013, pp. 1–6.

[24] M. Moreira, C. Oliveira, R. Porto, and N. Calazans, “Design of ncl gates with the ascend flow,” in *IEEE Latin American Symposium on Circuits and Systems*, 2013, pp. 1–4.