2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)

Pulsar: Constraining QDI Circuits Cycle Time Using Traditional EDA Tools

Marcos L. L. Sartori, Rodrigo N. Wuerdig, Matheus T. Moreira, Ney L. V. Calazans
PUCRS - School of Technology - Ipiranga Av., 6681 - Porto Alegre - Brazil, 90619-900
{marcos.sartori,rodrigo.wuerdig,matheus.moreira}@acad.pucrs.br, ney.calazans@pucrs.br

***Abstract*— Asynchronous quasi-delay-insensitive (QDI) circuits are known for their potentially enhanced robustness to PVT variations when compared to synchronous circuits or to bundled-data asynchronous design. They are also a good choice for high-performance circuits used to solve several real-world problems. However, it is often difficult to constrain the minimum performance for QDI circuits. Thus, enhancing the synthesis quality for QDI design is a justifiable effort, especially in rising application fields, such as the Internet of Things and Artificial Intelligence. This work proposes Pulsar, a method based on the extension of SDDS-NCL, a previously proposed asynchronous QDI template and design flow. Pulsar brings four original contributions: (i) two new models for components used to as sequential barriers; (ii) a new model for half buffer pipelines, half-buffer channel network (HBCN); (iii) a linear programming formulation to define a circuit cycle time constraint; (iv) a design flow that enables automating the process to design sequential SDDS-NCL circuits. Experiments comparing synthesis results with Pulsar of a 6-stage, multiply-accumulate (MAC) show that it can guarantee a maximum cycle time of $3.2\,\mathrm{ns}$, while the original Uncle-synthesised circuit without logic optimisation leads to timing violations at a $6\,\mathrm{ns}$ constraint.**

I. Introduction

One of the challenges to design asynchronous circuits is guaranteeing some minimal throughput operating point. This throughput is dependent on the maximum cycle time of the circuit. However, on complex concurrent asynchronous systems the cycle time is not trivial to capture. Synchronous circuits typically rely on register transfer level (RTL) models, where the maximum throughput is limited by a clock period. This not only makes design capture simpler, but also eases the task of optimising a netlist, as every timing path has a same, fixed maximum delay constraint, the clock period. In fact, synchronous RTL models drove decades of development on commercial EDA tools, which provide strong means for designers to explore power, performance and area optimisation in modern technologies. These means are nonetheless very specific, and efforts to abandon the synchronous paradigm in exchange for more powerful design techniques can easily make commercial tools not applicable. Accordingly, the support for asynchronous design lacks behind and, as technologies get less predictable and wire dominated, there is a particular need for new solutions that allow asynchronous circuit optimisation after technology mapping and during physical design.

This article proposes Pulsar, a method to enable constraining the cycle time of an asynchronous QDI circuit. Pulsar takes full advantage of commercial, timing-driven EDA synthesis and optimisation tools. The Pulsar flow uses commercial tools to optimise an NCL circuit to NCL/NCL+ gates under synthesis constraints, producing sequential QDI circuits with a bounded worst-case delay. It integrates the Thonnart et al. pseudo-synchronous design technique [1] with the SDDS-NCL design flow [2]–[4], extending both into the Pulsar method, which brings four main original contributions: (i) suggest two new sequential barrier component models; (ii) model half-buffer pipelines with an adaptation of Beerel et al.'s full-buffer channel networks (FBCNs) [5]; (iii) propose a linear programming formulation to define a target pseudo-clock period, based on a target circuit cycle time; and (iv) generalising the previous SDDS-NCL flow [4] to deal with sequential design.

II. The SDDS-NCL Asynchronous Design Template

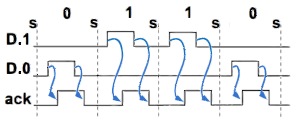
In a 4-phase dual-rail delay-insensitive (DI) channel *D*, a single bit datum is represented using two signals, *D.1* and *D.0* that carry the datum value, and one signal *ack* to control data flow. For the data portion of a channel, as Figure 1(a) depicts, a spacer is classically encoded as a codeword with all signals at 0. Valid data are encoded using exactly one signal at 1, *D.1*=1 for a logic 1 and *D.0*=1 for a logic 0. In this case, both signals at 1 is a codeword that does not correspond to any valid datum and is not used. Figure 1(b) shows an example of data transmission using this convention to demonstrate the control flow allowed by the *ack* signal combined to codewords represented in signals *D.1* and *D.0*. In this example a sender provides dual-rail data in *D.1* and *D.0* to a receiver that acknowledges received data through *ack*. Communication starts with a *spacer*, all signals at 0. Note that the *ack* signal also starts at 0, meaning the receiver side is ready to acquire new data. Next, the sender puts a valid 0 bit in the channel, raising the logic value of *D.0*, which is acknowledged by the receiver raising the *ack* signal, in what is called the *evaluation phase*. After the sender receives *ack*, it produces a spacer to end communication, bringing all data signals in the channel back to 0, beginning the *reset phase*. The receiver then lowers its *ack* signal, after which another communication can take place. Due to its nature (requiring all signals to go to 0 before each data transmission), this 4-phase protocol is also known as return-to-zero (RTZ).

Another 4-phase protocol for dual-rail QDI design is the return-to-one (RTO) protocol [6]. RTO employs the same amount of valid codewords as RTZ, but its data values are inverted compared to the latter. As Figure 2(a) shows, a spacer

978-1-5386-5883-3/19/$31.00 ©2019 IEEE
DOI 10.1109/ASYNC.2019.00023

Code	D.0	D.1
Spacer	0	0
Value 0	1	0
Value 1	0	1

(a) data encoding



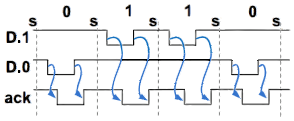
(b) example of data transmission waveform

Fig. 1: RTZ dual-rail channel operation. Adapted from [4].

here is the codeword with all signals at 1 and valid data is represented by one signal at 0, *D.1*=0 for a logic 1 and *D.0*=0 for a logic 0. Figure 2(b) depicts an example RTO data transmission, which starts with all signals at 1 in the data channel (the RTO spacer). As soon as the sender puts valid data in the channel, the receiver may acknowledge it by lowering *ack*. Next, all data signals must return to 1 to denote a spacer, ending 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 but powerful and allows a better design space exploration for QDI circuits, enabling optimizations in power [7] and robustness [8]. Furthermore, as demonstrated in [9], RTZ and RTO can be mixed in a same QDI design and the conversion of values between them requires only an inverter per wire. According to Martin and Nyström, in [10], such conversion is DI and does not compromise the robust functionality of a QDI circuit. This work refers to signals operating under the RTZ (RTO) protocol as RTZ (RTO) signals.

Code	D.0	D.1
Spacer	1	1
Value 0	0	1
Value 1	1	0

(a) data encoding



(b) example of data transmission waveform

Fig. 2: RTO dual-rail channel operation. Adapted from [4].

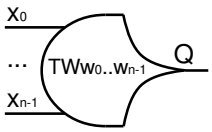
As Fant and Brandt discuss in [11], while a QDI logic block is transitioning between a spacer and valid data, output values of the block should be either a spacer or valid data. Therefore, NCL gates must also account for situations where an input combination is neither in the ON-set nor in the OFF-set[1]. While in these states no gate output should transition. This leads to the definition of the correct behaviour for NCL gates.

[1]ON-set and OFF-set are classical definitions from the logic synthesis domain. The ON-set of a function is the subset of points of the function domain for which the function output evaluates to 0. The OFF-set is the subset of points for which the function output evaluates to 1. For example, a 2-input AND function has ON-set={11} and OFF-set={00, 01, 10}. Incompletely specified functions can be described using three, instead of two domain subsets: ON-, OFF-, and the DC-set, the latter describing the points of the domain for which the output is not specified (don't-cares). Three sets can also describe C-elements and/or NCL/NCL+ functions: ON-, OFF-, and the HOLD-set, the latter being the subset of domain points for which the function output keeps its previous value. The reader can verify that the OFF-sets of NCL and INCL+ functions always have cardinality 1, the same being true for the ON-set of NCL+ and INCL functions.

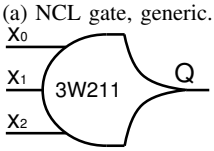
Definition 1. *An n-input* **NCL gate** *is a logic gate with a threshold value $T \in N^*$, a weight $w_i \in N^*$ assigned to each variable x_i $(i = 1, \ldots, n)$, and a hysteresis mechanism such that the gate output Q at each instant of time t is given by:*

$$Q_t = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 0, & \sum_{i=1}^{n} x_i = 0 \\ Q_{t-1}, & 0 < \sum_{i=1}^{n} w_i x_i < T \end{cases} \quad (1)$$

Figure 3(a) shows a generic NCL gate symbol, where n is the number of inputs of the gate and T is the threshold of the *threshold logic function* (TLF) it implements, for which each input has a weight w_i. If a weight w_i is omitted, $w_i = 1$ is assumed. Weights come after the W specifier. As an example, Figure 3(b) shows the symbol of a 3-input NCL gate with threshold 3 and weights 2, 1 and 1 (in the order from the topmost input down). Figure 3(c) shows the truth table for this latter example, computed from Equation (1). Accordingly, the output of the gate only switches to 0 when all inputs are at 0. Also, because x_1 has weight 2, x_2 and x_3 have weight 1 and the threshold is 3, the gate only switches to 1 when x_1 is at 1 and at least one of the other inputs is at 1. In all other cases the output remains unchanged.



(a) NCL gate, generic.



(b) Specific NCL gate.

X_0	X_1	X_2	Q_t
0	0	0	0
0	0	1	Q_{t-1}
0	1	0	Q_{t-1}
0	1	1	Q_{t-1}
1	0	0	Q_{t-1}
1	0	1	1
1	1	0	1
1	1	1	1

(c) 3W211 gate truth table.

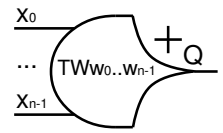
Fig. 3: Characteristics of NCL gates.

NCL supports the design of QDI circuits that follow the RTZ protocol and, to meet the requirements of associated templates, NCL gates are typically restricted to implement only positive unate functions. This limits the potential for logic optimisation and complicates achieving compatibility with conventional EDA tools. NCL+ gates are similar to NCL gates, but, because they target the RTO protocol [6] and related logic templates [12], they need to be able to detect spacers encoded by all wires at 1 and compute valid data signalled by logic 0s. The definition of an NCL+ gate is then straightforward.

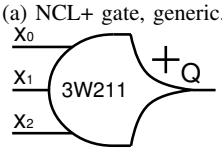
Definition 2. *An n-input* **NCL+ gate** *is a logic gate with a threshold value $T \in N^*$, specific weights $w_i \in N^*$ assigned to each variable x_i $(i = 0, \ldots, n-1)$, and a hysteresis mechanism such that the gate output Q at each instant of time t is given by:*

$$Q_t = \begin{cases} 1, & \sum_{i=0}^{n-1} \overline{x_i} = 0 \\ 0, & \sum_{i=0}^{n-1} w_i \overline{x_i} \geq T \\ Q_{t-1}, & 0 < \sum_{i=0}^{n-1} w_i \overline{x_i} < T \end{cases} \quad (2)$$

The symbol of an NCL+ gate is similar to that of an NCL gate, but with a + sign on its top right corner, see Figure 4(a). Figure 4(b) shows an example of a 3-input, threshold 3 NCL+ gate, with respective weights (from top to bottom) 2, 1 and 1. The truth table of this gate appears in Figure 4(c). This gate output only switches to 1 when all inputs are 1. Also, its output only switches to 0 when x_0, which has a weight of 2, and at least one of the other inputs, both with weight 1, are 0. For all other combinations of inputs, the output keeps its value.



(a) NCL+ gate, generic.



(b) Specific NCL+ gate.

X_0	X_1	X_2	Q_t
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	Q_{t-1}
1	0	0	Q_{t-1}
1	0	1	Q_{t-1}
1	1	0	Q_{t-1}
1	1	1	1

(c) 3W211 gate truth table.

Fig. 4: Characteristics of NCL+ gates.

NCL+ gates allow building QDI circuits using DI codes. Figure 5 shows the NCL+ version of a generate path for a 1-bit Kogge-Stone adder. The circuit topology is exactly the same as would be for an NCL version of the circuit. In fact, NCL and NCL+ gates are similar, the only difference being their underlying assumptions on data and spacers representation. Clearly, in the NCL+ circuit all internal nodes and primary inputs and outputs follow the RTO protocol.

The advent of NCL+ enables mixing NCL and NCL+ gates in a single circuit, because RTO signals can be translated to RTZ signals and vice-versa just using inverters [9]. As [2] discusses, mixing NCL and NCL+ also allows mixing positive and negative unate functions, improving optimisation opportunities and expanding the QDI circuit design space. For each positive unate NCL (NCL+) gate, a negative unate gate can be defined, where the latter has as OFF-set the ON-set of the former. To differentiate negative unate gates from positive unate ones, these are named *Inverted* NCL and NCL+ gates (or INCL and INCL+, respectively) and defined accordingly.

Definition 3. *An n-input* **INCL gate** *is a logic gate with a threshold value $T \in N^*$, specific weights $w_i \in N^*$ $(i = 0, \ldots, n-1)$, assigned to each variable x_i and a hysteresis mechanism such that the gate output Q at each instant of time*

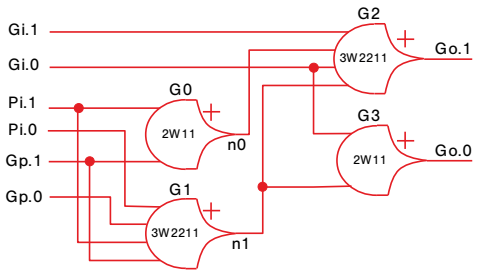


Fig. 5: Example of an NCL+ circuit: the generate path of part of a Kogge-Stone adder. Adapted from [2].

t is given by:

$$Q_t = \begin{cases} 1, & \sum_{i=0}^{n-1} x_i = 0 \\ 0, & \sum_{i=0}^{n-1} w_i x_i \geq T \\ Q_{t-1}, & 0 < \sum_{i=0}^{n-1} w_i x_i < T \end{cases} \quad (3)$$

An INCL+ gate is similarly defined *mutatis mutandis* [4].

From a functional point of view, the only difference between an NCL (NCL+) and an INCL (INCL+) gate is that their ON- and OFF-sets are swapped. However, both still eventually rely on a hysteresis behaviour to ensure the respect of QDI properties. INCL/INCL+ gate symbols differ from the respective non-inverted gate symbol by a circle added to its output. Using inverted gates it is possible to convert signals from RTZ to RTO, and vice versa. Because each time an inverted gate is used the protocol changes (including the codeword to represent spacers), circuits using (I)NCL and (I)NCL+ gates are called *spatially distributed dual spacer NCL* (or SDDS-NCL) [2].

Figure 6 shows an example SDDS-NCL circuit, equivalent to that in Figure 5. When this circuit inputs have spacers, it issues a spacer in its output. Here, all INCL gates have 1 in their outputs, which means that the INCL+ gates have 1 in all their inputs and a 0 in their outputs. In other words, all first level wires (in blue) will be at 0 and all second level wires (in red) will be at 1. From this state, whenever the inputs become valid dual-rail data, exactly two of the INCL gates fire, setting their outputs to 0, which causes exactly one of the INCL+ gates to fire, setting its output to 1.

The two inverters *G4* and *G5* (1W1 gates) are required to ensure that all the inputs of gates *G2* and *G3* are in the same domain (RTO). Also, because there are only two levels of logic and all gates are negative unate, the external inputs and outputs are RTZ. If the output was expected to be RTO, inverters should be added after gates *G2* and *G3*, or, of course, these gates could be mapped to NCL+ ones. In fact, different combinations of NCL, INCL, NCL+ and INCL+ gates can be explored, depending on the requirements of the circuit's input and output channels.

A final important concept relevant to the SDDS-NCL flow is that of *virtual functions* (VFs). A VF captures the behaviour of the gate at the evaluation phase only. The concept applies

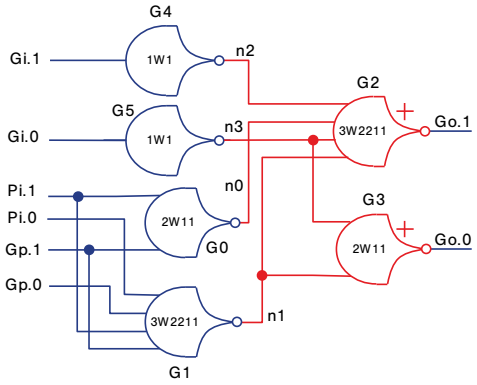


Fig. 6: Example of an SDDS-NCL circuit: generate path for a Kogge-Stone adder. Adapted from [2].

at the logic optimisation and technology mapping phases of the SDDS-NCL synthesis flow to select gates [4].

Definition 4. *A* **virtual function (VF)** *is an n-input Boolean function associated with an n-input NCL, NCL+, INCL or INCL+ gate, called its* support gate. *The truth table of a virtual function f is defined as follows:*

1) *if the support gate of f is an NCL gate θ, the ON-set of f is the same as the ON-set of θ. The OFF-set of f comprises all other n-input patterns;*
2) *if the support gate of f is an NCL+ gate ϕ, the OFF-set of f is the same as the OFF-set of ϕ. The ON-set of f comprises all other n-input patterns;*
3) *if the support gate of f is an INCL gate ψ, the OFF-set of f is the same as the OFF-set of ψ. The ON-set of f comprises all other n-input patterns;*
4) *if the support gate of f is an INCL+ gate υ, the ON-set of f is the same as the ON-set of υ. The OFF-set of f comprises all other n-input patterns.*

If the support gate of f is of types NCL or NCL+, f is a positive VF. *Otherwise, f is a* negative VF.

First, from the definition it is easy to conclude that VFs are completely specified functions, comprising only an ON-set and an OFF-set. Also each VF always has exactly one NCL and one NCL+ support gates or one INCL and one INCL+ support gates. Reference [2] provides the detailed description of a method to compute the support gates for VFs. As another consequence of Definition 4, all positive VFs are positive unate functions, but not all positive unate functions are positive VFs. Furthermore, all negative VFs are negative unate functions, but not all negative unate functions are negative VFs. Hence, all VFs are unate functions, but not all unate functions are VFs. As an example, consider the 3-input NCL gate 3W211 depicted in Figure 3. The reader can verify that a virtual function f_1 for this gate can be expressed by $f_1 = x_0.(x_1 + x_2)$. Another example is the 3-input NCL+ gate 3W211 depicted in Figure 4 has a virtual function f_2 expressible by $f_2 = x_0 + x_1.x_2$. As expected and can be verified, these two VFs are positive unate.

III. A Pseudo-Synchronous Model for SDDS-NCL

As Moreira et al. explore in [2], [4], SDDS-NCL is a useful asynchronous design template to build QDI combinational logic. It allows the designer to leverage industry standard tools and flows with unprecedented compatibility with logic minimisation algorithms. In fact, the template focus on the modelling of NCL and NCL+ gates to allow logic optimisation while preserving DI encoding. However, it assumes that all logic paths start at primary inputs and end at primary outputs, which enables specifying maximum delay constraints from inputs to outputs, but cannot deal with sequential logic descriptions. Sequential design with SDDS-NCL so far requires manual design.

Thonnart et al. [1] propose the pseudo-synchronous flow, which relies on a clever modelling of asynchronous components and on standard static timing analysis (STA) tools to optimise sequential logic in a specific QDI design template, WCHB. These authors relate the timing arcs of a resetable C-element, a basic building block of WCHB pipelines, to those of edge-triggered flip-flops, a basic building block of synchronous pipelines. From this analysis arises the proposal of a flop-like model to represent sequential barriers in the WCHB template (realised with resetable C-elements). The new model allows the authors to propose a *pseudo-synchronous* synthesis flow for WCHB, supported by conventional STA tools. The implementation of the flop-like model for resetable C-elements consists in new Open Liberty files (.lib) containing timing tables that substitute the original resetable C-element characterisation files. The reasoning is that conventional tools can analyse acyclic timing paths bounded by well defined start points and end points. A start point for an acyclic timing path is either a primary input or the clock pin of a flip-flop; and an end point is either a primary output or the data input pin of a flip-flop. The clock signal provides a stable reference for timing analysis, the propagation of signals in an acyclic path must terminate before the arrival of a clock event in the end point flip-flop. This model allows the analysis of paths that start and/or end in pseudo-flops without adding significant error to the actual delay of these C-elements. The work details the specification of a pseudo-clock that guides the synthesis decision making process and constrains each logic stage during synthesis. The work also include a methodology to characterise the pseudo-flops.

Figure 7 details four models covered in the present work. Figure 7(a) schematically depicts the conventional model of the 2-input resetable C-element. Its characterisation is given in a .lib file as power and delay figures as a function of input ramps (the slew) and output load capacitances for each individual transition of any of the inputs (A, B, Reset) causing a change in output Z. Figure 7(b) depicts the pseudo-flop model proposed by Thonnart et al. for the same component. The right side of Figure 7 depicts the new models proposed in this work, explained in more detail later in this Section.

In the Thonnart et al. model, the arcs in the .lib file are modified to mimic those of a flip-flop (Figure 7(b)) [1]. Here,

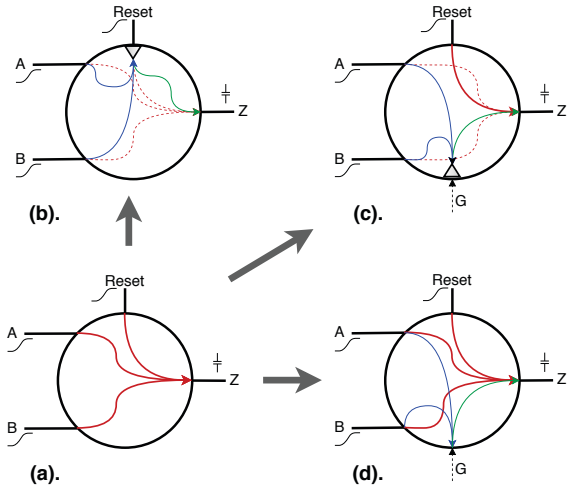


Fig. 7: Different characterisation models for a resetable C-element: (a) the original C-element arcs; (b) Thonnart et al. [1] model; (c) the flop-based model used here for synthesis; (d) the latch-based model used here for sign-off simulation.

the C-element Reset becomes a pseudo-clock pin and original propagation arc is split in two: (i) a clock propagation delay (in green), from Reset to Z (a function of the slew in Reset and of the capacitance in Z); and (ii) setup constraints (in blue), from A to Reset (a function of the slew in A and of the slew in Reset) and from B to Reset (a function of the slew in B and of the slew in Reset). Since the sequential nature of Reset is only a mechanism to create the pseudo-clock, its slew can be ignored. The generated delay tables are thus a single row (or column) that depends only on the slew of A or B and on the capacitance driven by Z. The reasoning is that the original arcs can be built from the sum of a clock propagation delay and a setup constraint on the new model. For example, the original A to Z arc is represented by the propagation delay from Reset to Z added to the setup constraint from A to Reset. Every time the STA tool encounters a pseudo-synchronous flop model of a C-element, it identifies a new start point for paths starting at output Z and a new end point for paths ending at inputs A or B. This is crucial to enable STA in WCHB pipelines, as every pipeline stage is marked by a group of C-elements (e.g. 2 for a 1-of-2, 4-phase encoding, 1-bit data channel). The logic paths between these C-elements are optimised to meet the defined period (λ) of the pseudo-clock connected to the Reset pin.

Figure 8 shows how these paths are analysed in a WCHB pipeline. Here, a single C-element forms the memory of each pipeline stage and two types of logic paths: (i) the forward logic (usually where useful computation happens); and (ii) the backward logic (usually used for flow control). Between every pair of C-elements in a cycle there are forward and backward logic paths constrained by a single pseudo-clock, with period λ, entering at the Reset pin. There are also paths between the input channel and the first C-element and the last C-element and the output channel.

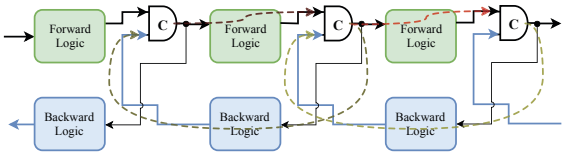


Fig. 8: Simplified view of a 1-bit data channel WCHB pipeline, showing the inner cycles controlled by the pseudo-clock.

A major drawback of the Thonnart et al. method is that it causes an error in the computation of delays for the C-elements, as a result of the regeneration of the original arc from the sum of the two new arcs. This takes place because the new arcs do not describe the delay of the cell as an explicit function of the slew in its inputs and the capacitance in its output. It rather relies on two independent delay values, which are then added, one as a function of the capacitance driven by Z and another as a function of the slew in A or B.

Another drawback is that the reset network is tied to a pseudo-clock network, and optimising these signals independently becomes challenging, since balancing reset signals conflicts with the tool trying to synthesise a clock tree for the pseudo-clock. Also, the annotation of delays for post-implementation simulations in this model is tricky. The created arcs are in accordance to the pseudo-synchronous model, not fitting the original C-element arcs, and will not reflect the real delay of the circuit. Thonnart et al. suggest that the original models be provided to the tool for delay annotation. However, this causes the automatic insertion of loop breakers, which complicates analysis.

To deal with these issues, this work suggests creating a fictitious clock pin (G) and proposes two new models to characterise C-elements and other sequential barrier components: (i) a D-type flop model (Figure 7(c)), used during synthesis; (ii) a latch model (Figure 7(d)) that preserves all original propagation arcs, only used to extract delay annotations for post-synthesis simulation. The fictitious pin G only exists in the .lib file representing the cell timing arcs, it is absent in both the cell layout and abstract views. The use of pin G allows preserving the original Reset pin timing arcs. This enables using STA to design a reset network meeting the constraints extracted from the cell characterisation process.

The pseudo-latch model relies on the fundamental behaviour of latches. A latch is a sequential element that has two distinctive operation modes, transparent and opaque. During the transparent mode operation, a latch has a direct arc from its data input to its data output and during the opaque mode it holds data. Transitions between these modes are governed by a clock signal, which imposes sequential arcs on the latch. Therefore, a latch timing model typically includes a setup constraint and a clock to output delay (as in flip-flops), and a combinational arc from its input to its output (to serve in those cases when the latch is transparent). This model respectively creates valid start and end points for STA at the outputs and inputs of latches. During the pseudo-latch characterisation, this feature is leveraged to preserve the original arcs of the

C-element, while avoiding the insertion of loop breakers. In simulation, it is enough that the behavioural model consider the cell as always in the transparent mode to reflect these delays, ignoring any sequential arc.

Although this paper only explores the use of C-elements with reset, the use of a fictitious pin for the pseudo-clock allows any NCL(NCL+) gates to be characterised both using pseudo-flop or pseudo-latch models, if they are to be used as sequential barrier components. Gates characterised using this technique can be modelled as *sequential combo cells*, composed of a combinational gate with the VF of the original gate followed by the sequential element. This enables EDA tools to differentiate pseudo-flop functions during synthesis.

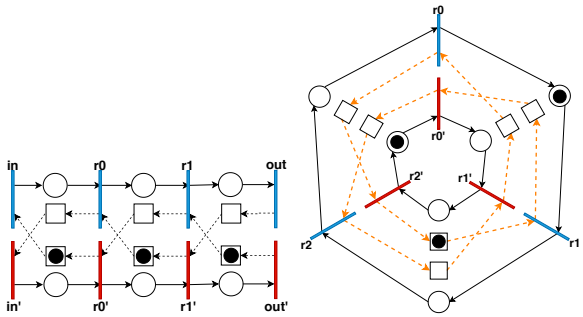
IV. SDDS-NCL Cycle Time Optimisation

Sections II and III explored SDDS-NCL components and models that render commercial tools compatible with the design of combinational and sequential QDI circuits. It is possible to leverage state-of-the-art logical and physical EDA tools to synthesise and analyse QDI circuits more effectively. However, the performance optimisation of these cannot be achieved by merely setting a (pseudo-)clock period as a target maximum delay between start and end points, as in traditional synchronous design. Rather, QDI performance is dictated by the complex process to determine the circuit cycle time.

Beerel, Najibi et al. [13], [14] proposed the use of timed marked graphs and linear programming to analyse the maximum cycle time of a circuit and improve its performance by inserting slack matching buffers. For cycle time analysis, the circuit is modelled as a timed marked graph with delays on places. The graph largest cycle time is computed using a variation of Magott's algorithm [15]. In this algorithm, an arrival time inequation $a_j \geq a_i + d(p) - m_0(p)\tau$ is defined for each place p, where a_j is the arrival time at the transition succeeding (triggered by) p, a_i is the arrival time at the transition preceding (feeding) place p, $d(p)$ is the delay associated with the place, τ is the maximum cycle time of the graph, and $m_0(p)$ is the number of tokens the place holds at the initial time. The inequation states that transition a_j should occur at least $d(p)$ time units after a_i has occurred. Also, $-m_0(p)\tau$ means that if there is a token in p the transition a_i has occurred at least τ time units before. To find out the maximal cycle time, we need to find the *minimal value of* τ that satisfies the system of inequations. This can be easily computed by a linear programming (LP) solver.

The method is extensible to a synthetic approach to design a circuit that meets a certain maximum cycle time. Instead of defining the maximum cycle time τ of a given circuit, these models and inequations can determine a maximum delay constraint λ for places, which is required to meet a desired, given maximum cycle time (ϕ).

To compute the maximum delay constraint λ, circuits are modelled as a *Half Buffer Channel Network* (HBCN), a modification of the FBCN concept proposed by Beerel et al. in [5]. This is a second original contribution of this work. Figure 9 shows two example HBCNs. HBCNs are marked graphs, a type of Petri net with no choice, meaning that a single transition follows each place. In HBCNs, each sequential barrier is modelled as a pair of transitions, each transition indicating the arrival of a type of token: (i) data, in blue; and (ii) spacer (null), in red. The environment is also modelled as transitions, allowing to capture input and output delays. Each logic propagation path is modelled as a pair of places: (i) one indicating the propagation of a data token; and (ii) another indicating the propagation of spacers. Square and round places respectively represent backward and forward propagation paths. Figure 9(a) illustrates this. Note that the handshake protocol is evident by analysing the cross-connecting null backward propagation place of a stage to the data transition of the previous stage and the data back-propagation place to the previous stage null transition.



(a) A 2-stage linear buffer. (b) A 3-stage circular buffer.

Fig. 9: Example HBCNs modelling 4-phase, half-buffer QDI circuits. Blue lines are valid data transitions; red lines are spacer transitions; squares are backward propagation places; circles are forward propagation places.

In this model, the delay of a place ($d(p)$) is equivalent to the delay of a propagation path. Because it is possible to constrain these propagation paths using SDDS-NCL and the pseudo-synchronous flow, it is fair to assume that all these paths will have a same maximum delay. Therefore, a single delay λ is assigned to each of them, which is the desired maximum delay, used as a pseudo-clock period. Consequently, the arrival time inequation is slightly modified to $a_j \geq a_i + \lambda - m_0(p)\phi$. A system of arrival time inequations is extracted from the HBCN model, the maximum cycle time value ϕ is set to the desired value and the value of λ is maximised using a linear programming solver. This is the third original contribution of this article. The computed maximum value of λ is used to constrain the timing arcs of the original circuit, see Figure 8. The intuition behind the method is to discover "how slow" each stage can be without violating the ϕ constraint. Logic stages "slower" than λ may impact the maximum cycle time, violating ϕ.

To demonstrate the method, the LP formulation in Table I is used to compute the maximum delay constraint for the 3-stage circular buffer shown in Figure 9(b) to the maximum cycle time of 5ns. It states that the value of λ should be maximised, subject to $\phi = 5$ and the system of arrival inequations. Each

line corresponds to a place in the HBCN and states that the transition on the postset of that place should occur at least λ time units after the transition on the preset, except on places with tokens where the transition can occur ϕ time units ahead. Running this model on Gurobi LP solver results in $\lambda = 833ps$.

Maximise:	λ
Subject To:	$\phi = 5$
$a_{r0} \geq a_{r1} + \lambda - \phi$	$a_{r1} \geq a_{r2} + \lambda$
$a_{r2} \geq a_{r0} + \lambda$	$a_{r0'} \geq a_{r1'} + \lambda$
$a_{r1'} \geq a_{r2'} + \lambda$	$a_{r2'} \geq a_{r0'} + \lambda - \phi$
$a_{r2'} \geq a_{r1} + \lambda - \phi$	$a_{r1} \geq a_{r0'} + \lambda$
$a_{r0'} \geq a_{r2} + \lambda$	$a_{r2} \geq a_{r1'} + \lambda$
$a_{r1'} \geq a_{r0} + \lambda$	$a_{r0} \geq a_{r2'} + \lambda$

TABLE I: LP formulation to compute the maximum delay λ to constrain the HBCN from Figure 9(b) to a maximum cycle time of 5ns.

V. The Pulsar Optimisation Flow

Figure 10 depicts the Pulsar optimisation flow, the fourth original contribution of this article. Pulsar is a set of TCL scripts driving the Cadence™ framework. Logic synthesis and optimisation employ Genus™ 18.1. Placement and routing are achieved with Innovus™ 18.1.

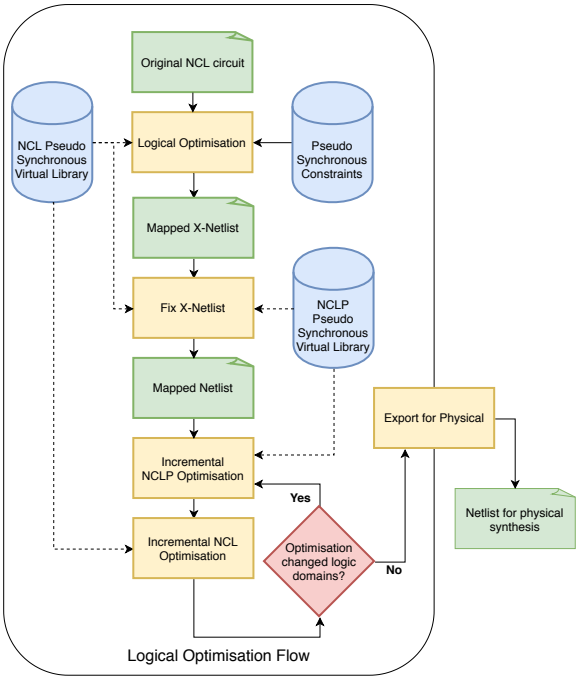


Fig. 10: The Pulsar optimisation flow.

The flow takes a pre-synthesised NCL netlist as input. This netlist contains both combinational and sequential elements. Pseudo-flops are explicitly instantiated as sequential elements in pipeline stages. NCL gates implement combinational logic for both forward and backward propagation between sequential elements. Alternatively, Boolean expressions can be used to describe the behaviour of combinational logic as VFs.

The pipeline structure is that in Figure 8. Forward-propagation logic is always positive unate, meaning it preserves the logic protocol of the inputs. Conversely, backward-propagation (completion detection) logic is negative unate, meaning that it inverts the protocol of its inputs. Hence, sequential elements have their inputs at opposite domains.

SDDS-NCL requires that the logic be strictly unate, meaning that all inputs of a gate must reside on the same domain. Since SDDS-NCL is only used for combinational logic, there is no problem in having sequential gates with their inputs at different domains. To solve this, a "size_only" constraint is set on all sequential elements to preserve their logic function.

Genus synthesises and optimises the combinational logic in both backward and forward propagation paths. Synthesis and optimisation must meet some requirements to generate a fixable X-Netlist: (i) they must not generate intermediate circuits with binate functions, as gates in these circuits have inputs at different domains (RTZ and RTO), which would break the circuit behaviour; (ii) they must not produce circuits with gate inputs tied to a constant, as a constant at gate inputs may inhibit the set or reset behaviour of NCL(NCL+) gates.

Since in our design flow the input always implements unate functions, all internal nodes should be unate in the paths start points, i.e. primary inputs and sequential barrier outputs. However, according to Das et al. [16] it is possible that the synthesis tool to perform a binate realisation of a unate input function. To overcome this issue Moreira [17] proposes using design for testability (DFT) techniques, guaranteeing that requirement (i) above is always meet. Requirement (ii) is fulfilled setting "iopt_force_constant_removal=true" on Genus.

Assuming the initial circuit description is NCL (i.e. RTZ), the output of sequential gates are guaranteed to be RTZ, since they were preserved during the initial synthesis and optimisation steps. It is also safe to assume from this condition that all primary inputs are also RTZ.

To produce a circuit with the correct behaviour, gates which are in the RTO domain must be NCL+ and gates in the RTZ domain must be NCL. The combinational logic in both forward and backward propagation paths are processed using a variation of the Fix X-Netlist algorithm proposed by Moreira et al. in [4]. In the Fix X-Netlist algorithm, a gate is considered in RTZ if its non-inverting fan-ins are RTZ and its inverting fan-ins are RTO. Conversely, a gate is considered in RTO if its non-inverting fan-ins are RTO and its inverting fan-ins are RTZ. This is shown in Figure 11.

If a combinational gate is classified as both RTO and RTZ, the X-Netlist is deemed unsuitable for correction and the synthesis is invalid. The only gates allowed to be in both RTO and RTZ domains are the sequential barriers, but these do not need to be processed by the Fix X-Netlist algorithm.

The initial netlist correction may result in a circuit that no longer meets the design constraints. Thus, further optimisation steps are performed to meet these. These steps are run on each set of gates independently, to avoid undoing the initial

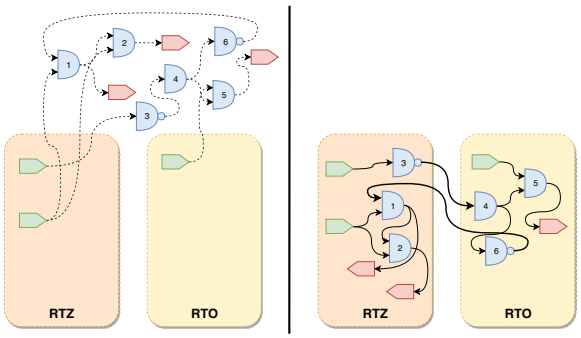


Fig. 11: Gates classification into RTO and RTZ

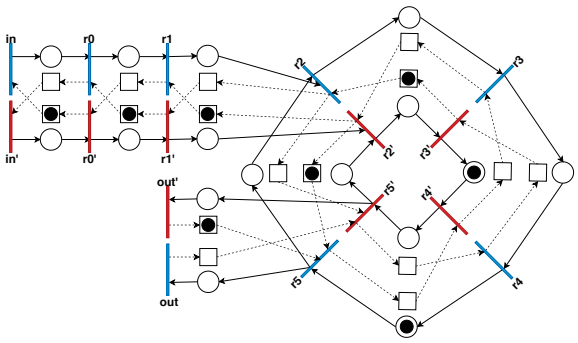


Fig. 12: HBCN of the MAC circuit used in the experiments.

netlist correction. After optimising a set of gates, the X-Netlist correction algorithm is re-run to correct any domain changes introduced by the optimisation. These steps are iteratively run until no gate is replaced by the correction algorithm.

After logic synthesis, optimisation and correction steps, the final netlist can be placed and routed by a physical synthesis tool. However, meeting timing constraints during physical synthesis may require some additional optimisation steps. If the logic is changed during these steps, logical correctness of the circuit may be compromised.

Prior to place and route, all NCL+ gates are marked with "dont_touch" and "dont_use", all NCL gates are marked "size_only" and buffers and inverters are left free for the physical synthesis tool to use. This guarantees the logic will not suffer any changes during physical synthesis and gives the physical synthesis tool enough freedom to place buffers and resize gates to meet timing.

VI. Experimental Results

A 32-bit integer multiply-and-accumulate (MAC) design serves here as a case study to validate the Pulsar method. The target technology is the 65nm bulk CMOS from STMicroelectronics, for which we have implemented the ASCEnD-ST65-NCL library [18], containing more than 600 NCL cells. This library is compatible with the foundry core library of the selected technology. Uncle [19] produces the input NCL netlist for Pulsar. It was chosen to design the example MAC circuit due to its design capture flexibility, making it easy to develop complex circuits while maintaining a degree of control over the pipeline architecture. Uncle performs dual-rail expansion and generates the completion detection circuit for each pipeline stage automatically. It also does latch balancing (retiming), which improves circuit performance. Besides taking advantage of the design capture, dual-rail expansion and retiming, Uncle is also a basis for comparison.

Figure 12 shows the MAC architecture HBCN model. This model is simplified by merging parallel paths, i.e. the two multiplier inputs are acknowledged by a same completion detector. Therefore, these are represented as a single input. The same is true for any of the registers and the output.

The MAC comprises a 2-stage linear pipeline and a 4-stage loop. A multiplier is at the linear pipeline input and an adder is at the point where the linear pipeline encounters the loop. Logic spreads in these stages using Uncle's latch-balancing optimisation. A 4-stage loop design was chosen to allow the simultaneous propagation of bubbles and tokens. The 2-stage linear pipeline provides room for the latch-balancing optimisation to spread the multiplier logic.

The MAC HBCN model allows calculating the pseudo-clock period that constrains it to a given maximum cycle time. Using the LP method proposed in Section IV, it was found that the pseudo-clock period constraint must be set to one fourth of the desired maximum cycle time constraint.

An RTL-like Verilog description composed of latches, registers and arithmetic operators derives from this HBCN. This description is the source to synthesise a synchronous netlist of Uncle's ANDOR2 gates using Genus. The synchronous synthesis step uses a 0 "max_delay" constraint on all paths to produce the shortest path arithmetic circuit.

Uncle expands each gate of the synchronous netlist to a dual-rail netlist of its internal library of NCL gates and implements the completion detection circuitry. It also optimises the NCL netlist by moving registers to balance the pipeline, and merges NCL gates according to pre-established patterns.

The netlist of Uncle NCL gates is mapped to equivalent gates of the ASCEnD-ST65-NCL with arbitrary drive strength. This netlist is used in two synthesis flows: one that preserves gate functions using a "size_only" constraint, hence called *the Uncle flow*; and *the Pulsar flow* where logic optimisation is allowed. In both flows a pseudo-clock was used to constrain the maximum cycle time and guide the synthesis effort.

Note that despite being called Uncle flow, the synthesis flow which preserves the gate functions includes optimisations not present in the flow proposed by Reese et al. [19]. Genus is used to resize gates and insert buffers in the design prior to physical synthesis. This is required to allow a fair comparison and to adjust the netlist to use the appropriate drive strengths available on the ASCEnD-ST65-NCL library.

ASCEnD-ST65-NCL has three PVT corners available: the *Best Corner*, characterised with fast transistors at $-40\,^{\circ}\mathrm{C}$ and $1.1\,\mathrm{V}$; the *Nominal Corner*, characterised with typical transistors at $25\,^{\circ}\mathrm{C}$ and $1\,\mathrm{V}$; the *Worst Corner*, characterised with slow transistors at $125\,^{\circ}\mathrm{C}$ and $0.9\,\mathrm{V}$. Both Uncle and Pulsar flows were used to perform a pseudo-clock period sweep

to evaluate how each flow is affected by timing constraints. All synthesis were performed using the *Worst Corner* for timing analysis, and the *Best Corner* for power analysis using Genus multi-mode, multi-corner (MMMC) synthesis. Timing annotation was based on SDF files extracted from each layout at the three different PVT corners. An MMMC corner is used to switch the library to a version containing the pseudo-latch gates for these annotations.

The final layout netlists were simulated using delays extracted from the layout for a period of $10\,\mu s$ using Questa 10.6e. The simulation environment is ideal, it reacts immediately to handshakes on the circuit inputs and outputs, acting as the ideal sink and source model to the HBCN. This approach keeps the pipeline always full and ensures that the worst case cycle times are internal to the circuit.

The environment collects three performance metrics: (i) the cycle time at the input, which is the time between two subsequent data token insertions; (ii) the cycle time at the output, i.e. the time between expelling two subsequent data token from the circuit; and (iii) the effective latency, i.e. the time required for a token at the input to produce a new token at the output. Cycle time measurements translate to the time between firings of the data token transition in the HBCN.

Figure 13 shows the distribution density of these performance metrics for the Worst Corner simulation of the circuit synthesised using the Pulsar flow with a pseudo-clock constraint of $0.8\,\mathrm{ns}$, corresponding to a cycle time of $3.2\,\mathrm{ns}$.

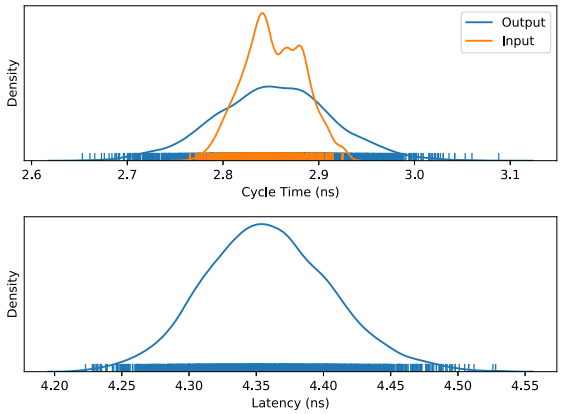
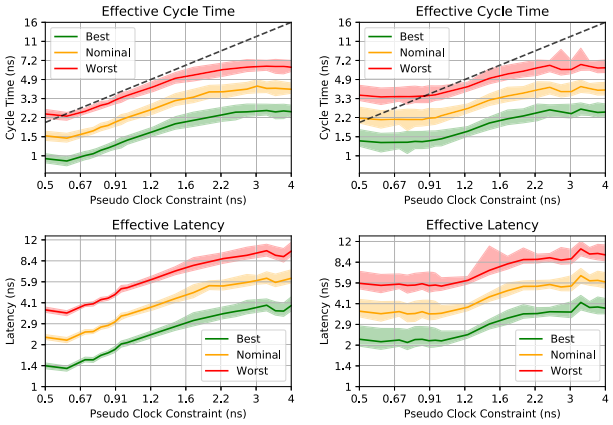


Fig. 13: Distribution density of the performance metrics for the Worst Corner simulation of a MAC synthesised using the Pulsar flow, with a pseudo-clock period constraint of $0.8\,\mathrm{ns}$, corresponding to a cycle time constraint of $3.2\,\mathrm{ns}$.

Cycle time measurements taken at the input and output are affected by the circuit internal cycle times. They evaluate the throughput of the circuit at its input and output under ideal conditions, serving as a measure of the overall circuit performance. The input and output cycle time measurements combined are called the *effective cycle time*.

Figure 14 shows a compilation of the performance measurements extracted from post-layout simulations of the MAC for different target pseudo-clock constraints. It shows how the pseudo-clock affects the synthesised circuit cycle time and latency under the Uncle and Pulsar flows. Measurements at each PVT corner appear as a range of values around the mean effective cycle time. Black dashed lines indicate the maximum cycle time constraint calculated for the circuit, depending on the pseudo-clock constraint.



(a) Pulsar flow. (b) Uncle flow.

Fig. 14: Effective cycle time and latency of the 32-bit MAC synthesised under pseudo-clock constraints. Extracted from post-layout, delay annotated simulation, under different PVT conditions. The dashed black line is the cycle time constraint.

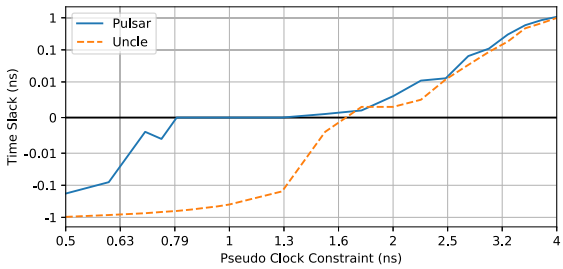
The time slack (Figure 15(a)) is the difference between the worst path delay and the pseudo-clock period constraint, and is reported by the STA tool. A negative slack value indicates that the circuit is not guaranteed to meet the cycle time constraint.

Energy results in Figure 15(b) come from static power analysis using switching activity annotations (in SAIF) from delay simulation at the Best Corner. The estimated power (in mW) is multiplied by the mean cycle time (in ns) to produce the average energy consumption per operation values (in pJ).

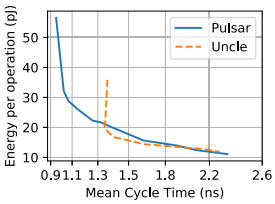
Area results in Figure 15(c) are evaluated through silicon efficiency. This is calculated by dividing the worst throughput by the area consumption. The worst throughput is calculated from the maximum cycle time of the worst case simulation and the area is extracted from the physical synthesis reports.

VII. Conclusions and Ongoing Work

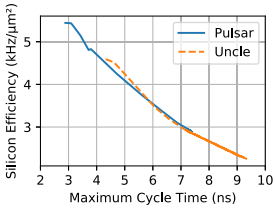
Pulsar enables the sign off of target cycle times for QDI circuits using commercial EDA tools. This is a major breakthrough for QDI designers, as they can now safely bound worst case performance metrics for their target applications. Moreover the flow enables designers to naturally trade performance for power or area optimisations, whenever there is slack in timing budgets. Figure 14(a) shows that Pulsar enables the implementation of circuits with tighter constraints than those achievable with traditional approaches. Tools such as Uncle limit post-technology mapping optimisation to only resizing and re-buffering, as Figure 14(b) makes clear. This is



(a) Post-layout time slack from pseudo-clock constraint.



(b) Energy per operation.



(c) Silicon Efficiency.

Fig. 15: Results highlighting the Pulsar optimisation potential.

further evidenced by the pseudo-clock time slack graph in Figure 15(a), where Pulsar shows a positive slack at smaller pseudo-clock period constraints. Even when violating timing, Pulsar shows a small negative slack between 0.6 ns and 0.8 ns that can either be corrected using Engineering Changing Orders (ECO) in Innovus for sign-off, or is tolerable in some applications, without a significant reduction in cycle time. To produce circuits that meet more restrictive timing constraints, Pulsar introduces some power overhead, as Figure 15(b) demonstrates. When timing is met by both flows, these obtain similar power consumption. Pulsar shows better overall silicon efficiency, as Figure 15(c) depicts. This advantage is more prominent on the regions where Uncle is no longer able to meet the timing constraint but Pulsar is.

A circuit with negative slack may still meet the cycle time constraint because the approach presented in Section IV is conservative. That is, every stage is constrained equally to meet the requirements of the maximum cycle time, with no regard for available slack. This can be seen on the model in Figure 9(b): to achieve a 5 ns cycle time the only places that need to be constrained to 833 ps are the 6 places participating in the maximum cycle, i.e. the orange dashed line. All other places have free slack and could be as slow as 1.66ns without affecting the maximum cycle time. This can be optimised by introducing additional pipeline stages to balance the free slack. For example, the MAC shown in Figure 12 has a 4-stage loop that breaks the 6-place worst cycle of the 3-stage loop in two 4-place cycles, each with its own token. The introduction of pipeline stages could be automated using the technique proposed by Beerel [13]. Alternatively, this free slack can be exploited to allow more relaxed timing constraints on certain arcs. Exploring these optimisations is ongoing work.

ACKNOWLEDGEMENTS

This research was partially funded by the CNPq under grants no. 200147/2014-5 and no. 312917/2018-0 (Brazil).

REFERENCES

[1] Y. Thonnart, E. Beigné, and P. Vivet, "A pseudo-synchronous Implementation Flow for WCHB QDI Asynchronous Circuits," in *IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2012, pp. 73–80.

[2] M. T. Moreira, G. Trojan, F. G. Moraes, and N. L. V. Calazans, "Spatially Distributed Dual-Spacer Null Convention Logic Design," *Journal of Low Power Electronics*, vol. 10, no. 3, pp. 313–320, 2014.

[3] M. T. Moreira, A. Neutzling, M. Martins, A. Reis, R. Ribas, and N. L. V. Calazans, "Semi-custom NCL Design with Commercial EDA Frameworks: Is it possible?" in *IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2014, pp. 53–60.

[4] M. T. Moreira, P. A. Beerel, M. L. L. Sartori, and N. L. V. Calazans, "NCL Synthesis With Conventional EDA Tools: Technology Mapping and Optimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 6, pp. 1981–1993, 2018.

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

[6] M. T. Moreira, R. A. Guazzelli, and N. L. V. Calazans, "Return-to-One Protocol for Reducing Static Power in QDI Circuits Employing m-of-n Codes," in *Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2012.

[7] ——, "Return-to-One DIMS Logic on 4-phase m-of-n Asynchronous Circuits," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2012.

[8] M. T. Moreira, R. A. Guazzelli, G. Heck, and N. L. V. Calazans, "Hardening QDI Circuits Against Transient Faults Using Delay-insensitive Maxterm Synthesis," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2014, pp. 3–8.

[9] M. T. Moreira, J. J. H. Pontes, and N. L. V. Calazans, "Tradeoffs between RTO and RTZ in WCHB QDI Asynchronous Design," in *International Symposium on Quality Electronic Design (ISQED)*, Mar. 2014, pp. 692–699.

[10] A. Martin and M. Nyström, "Asynchronous Techniques for System-on-Chip Design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[11] K. M. Fant and S. A. Brandt, "Null convention logictm: a complete and consistent logic for asynchronous digital circuit synthesis," in *International Conference on Application Specific Systems, Architectures and Processors (ASAP)*, Aug 1996, pp. 261–273.

[12] M. T. Moreira, C. H. M. Oliveira, R. C. Porto, and N. L. V. Calazans, "NCL+: Return-to-one Null Convention Logic," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2013, pp. 836–839.

[13] P. A. Beerel, A. M. Lines, M. Davies, and N.-H. Kim, "Slack Matching Asynchronous Designs," in *IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, Mar. 2006, pp. 184–194.

[14] M. Najibi and P. A. Beerel, "Slack Matching Mode-based Asynchronous Circuits for Average-case Performance," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 219–225.

[15] J. Magott, "Performance Evaluation of Concurrent Systems using Petri Nets," *Information Processing Letters*, vol. 18, no. 1, pp. 7–13, 1984.

[16] D. K. Das, S. Chacraborty, and B. B. Bhattacharya, "Irredundant binate realizations of unate functions," *International Journal of Electronics Theoretical and Experimental*, vol. 75, no. 1, pp. 65–73, 1993.

[17] M. T. Moreira, "Asynchronous Circuits: Innovations in Components, Cell Libraries and Design Templates," Ph.D. dissertation, Pontifícia Universidade Católica do Rio Grande do Sul, FACIN-PPGCC, 2016.

[18] M. T. Moreira, B. S. Oliveira, J. J. H. Pontes, and N. L. V. Calazans, "A 65nm Standard Cell Set and Flow Dedicated to Automated Asynchronous Circuits Design," in *IEEE International System on Chip Conference (SoCC)*, 2011, pp. 99–104.

[19] R. B. Reese, S. C. Smith, and M. A. Thornton, "Uncle - An RTL Approach to Asynchronous Design," in *IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2012, pp. 65–72.