Taking Timing Out of the Equation: Moving Fully Automated Synthesis as close as possible to Delay-Insensitive Circuits

Karthi Srinivasan*
Department of Electrical and Computer Engineering
Yale University
New Haven, CT, USA
karthi.srinivasan@yale.edu

Ole Richter*
DTU Compute, Technical University of Denmark
Kgs. Lyngby, Denmark
ECE, Yale University
New Haven, CT, USA
ojuri@dtu.dk

Jordan Schmerge
Department of Electrical and Computer Engineering
Yale University
New Haven, CT, USA
jordan.schmerge@yale.edu

Rajit Manohar
Department of Electrical and Computer Engineering
Yale University
New Haven, CT, USA
rajit.manohar@yale.edu

***Abstract*—In circuit design, delay-insensitivity is regarded as a highly abstract ideal. Delay Insensitive (DI) circuits function correctly independent of specific operating conditions and avoid the need for many technology-specific details. However, their practical application is limited due to the necessity of manual design and the complexity inherent in creating a wide range of circuit features satisfying delay insensitivity.**

Recent theoretical advancements in the field have led to a re-evaluation of these assumptions. This work aims to address the remaining gaps between theory and practice by proposing an approach to move fully automated design processes as close to integrated DI circuits as possible. The ability to automatically synthesize abstract async Hardware Description Language (HDL) into circuit layout realizations fulfilling the requirements of abstract delay insensitivity for Place and Route (P&R) is enabled by these advancements. This work compares the competitiveness of DI-like designs in area and performance to the more common Quasi Delay Insensitive (QDI) and Bundled Data (BD) implementations, discusses the performance overhead in relation to bundled-data designs, and points out the feasibility and cost of delay insensitivity in automated synthesis.

***Index Terms*—Asynchronous circuits, Delay-insensitive circuits, Quasi Delay-insensitive circuits**

I. Introduction

In the current era, data processing plays a pivotal role across all facets of society and work and the need for fast and correct sensing, processing, and transmission hardware is pervasive. However, in particularly challenging environments [1], it is often impractical to design modern hardware systems for an extended operation range, and it is also infeasible to build individual designs for specific operation purposes. Additionally, formal verification of hardware components has become increasingly important, particularly in high-risk and mission critical designs. However, despite these increased efforts, many systems still cannot be verified economically. This is due to a variety of factors, including the number and complexity of internal dependencies as well the "gap" between the level of abstraction that hardware is specified at versus the level it is designed at. Furthermore, these factors are compounded by the enormous scale that is now common in system design.

Asynchronous circuits [2], long overshadowed by its clock-driven subclass, possess inherent concepts for operation condition independent circuits as well as formal verification of designs by reducing physical implementation requirements for correctness. A multitude of asynchronous circuit families exist, imposing various restrictions on the physical design, often in the form of timing. Clocked-synchronous circuits impose setup and hold timing with reference to a global signal, ensuring data and computation consistency, as employed by most commercial design flows. Circuit families not referencing a global synchronization signal, such as BD designs, impose timing constraints on the handshake control between pipeline stages [3]. Speed Independent (SI) circuits demonstrate resilience to variations in gate delay [2]. QDI circuits exhibit robustness against delays introduced by wiring, operating under the assumption that some wire forks are isochronic, meaning that only one branch of the fork must be acknowledged for all the branches to have completed. Finally, DI circuits assert robustness against transmission delays on any input and thus any gate delay [4]. Removing as much dependency on timing as possible is also desirable from a formal verification perspective. The closer the hardware implementation can be made to the mathematical specification, the easier the proofs of correctness generally are.

The DI approach enables circuit designs to operate effec-

* contributed equally

This work was supported in part by US NSF grants CCF-2504588 and ECCS-2415262.

tively within a temperature range of $40K$–$400K$ [5, 6] and beyond, obviating the necessity for specialized design, or the errors that can occur if operating conditions don't match the design parameters. DI also ensures high yield and throughput, with the pipeline's performance evaluated by the average logic compute time, as opposed to the worst-case time, as in synchronous logic and bundled data.

Notwithstanding its merits, DI does carry significant drawbacks. The inherent complexity in designing DI circuits, coupled with their significantly reduced capability compared to BD variants, poses significant challenges. Notably, the design of the computing elements inside the data path without isochronic forks seemed unattainable [7, 8]. Moreover, their larger size and higher power requirements stand in contrast to the ease of implementation and energy efficiency of BD circuits. To address these limitations, QDI and SI designs as compromises gained popularity, eliminating most timing assumptions while maintaining functionality. In these variants, data is carried in n-of-m codes, often referred to as its subclass Dual Rail (DR), avoiding relative timing constraints beyond an isochronic fork [9, 10].

Recent research has demonstrated that the limitation in the design space of DI circuits is attributable to the fact that the utilized gates possess only a single output [11]. Notably, merely augmenting gate outputs to two is sufficient to eliminate the most stringent constraints and facilitate the automated synthesis of a substantial number of DI-like circuits. This paper undertakes a comprehensive analysis of the overhead and disparities that emerge when transitioning from QDI circuit families to DI-like circuits in the context of automated circuit synthesis and P&R.

This work demonstrates that in a surprisingly large number of cases, DI-like circuits have no overhead in terms of area ($0.91\times$), throughput ($0.94\times$) and energy ($0.96\times$) compared to QDI. To give the reader another basis of comparison, it specifies the average overhead of $2.15\times$ (area), $0.92\times$ (throughput) and $1.71\times$ (energy) compared to BD.

II. Background

The utilization of the DI concept in literary works is frequently characterized by imprecision in its application or limited in its scope, despite the existence of established definitions and formal concepts. To establish a foundation for the proposed approach, Keller [4] defines DI as follows: "*a [circuit] network is called delay-insensitive if its external behavior remains unchanged, regardless of whether any number of delay elements are inserted into, or removed from any lines.*". As any circuit component or gate delay can also be modelled as a delay on its output lines, DI circuits are automatically SI.

Udding [12] and Ebergen [13] formalized DI with trace-based semantics, including the necessary and sufficient conditions for DI based on transmission and computation interference and the Foam Rubber Wrapper metaphor: any wire can carry at most one transition, and the order of two transitions sent on different wires in the same direction is not preserved. Transitions in opposite directions can be ordered, but if both order variants are possible, the order should not matter. Furthermore, once a component is designated as "ready-to-receive" or "ready-to-send," it cannot be retracted.

Moving to the DI circuit abstraction also benefits formal verification, as timed circuits require more complex state space exploration. In clocked circuits, the state space has to take clock cycles into account regardless of if useful computation is being done; in the asynchronous case, timed interleavings must be constructed [14].

Subsequent to the findings of Keller [4], research was conducted to identify a minimal universal set of components for DI systems. Patra and Fussell [15] proposed a set consisting of standard elements like merges and forks and more complex components like "tria" and "2x2 join". Subsequent research by Lee *et al.* [16] presented a set of examples illustrating simple but not compact gates.

Still unaware of the implications of isochronic forks the research community invested significant efforts into what was thought to be DI systems. Driven by a pragmatic approach to circuit design and efficiency, Martin [17, 18] developed a synthesis method for the creation of circuits from the Communicating Sequential Processes (CSP) language. In a similar vein, Van Berkel and Saeijs [19] proposed a synthesis method for a subset of CSP with automated testability enhancements. Jesshope *et al.* [20] proposed a synthesis based on DI algebra, Sutherland [21] work introduced a pipeline-based approach, while Sparsø and Staunstrup [22] focused on ring-based structures based on minterm synthesis.

During this time Martin [7] pointed out that DI circuits were quite limited, and most circuits had isochronic forks; later Berkel [23] also pointed out how isochronic forks can violate DI properties. With the community being aware of the significantly more restricted set of actually possible DI systems [7, 24], the community moved on to QDI systems allowing isochronic forks. Early on QDI was proven to be universal for computing systems [25]. Consequently, DI systems were deemed ineffective for large-scale real-world computation systems and abandoned.

In practical terms, an (isochronic) signal fork results in a delay from the root of the fork to the leaves, which has a timing assumption. It is imperative to note that the delay must be constrained to be smaller than the gate and wire delays through the control cycle. In most practical scenarios, this is greater than a number of gate delays, and it is often deemed safe; however, in larger systems, this may not always be the case. To address this uncertainty, methodologies have been developed to assess the risk of isochronic forks within QDI systems [26].

However, research has since picked up and restricted the definition of DI to systems where sub-blocks are declared timing safe. This system then can fulfil the original definition of DI for the overall system, excluding the sub-components. For example, Park *et al.* [27] describes a system with a DI control-path while the timing of the data-path is ensured by Static Timing Analysis (STA). According to Saito *et al.*

[28], establishing a boundary concept for components fulfils the restricted definition, yet the responsibility of ensuring timing within sub-components falls upon the designer. In contrast, Meekins *et al.* [29] proposes the utilization of hand-designed hard macros with guaranteed timing within a more extensive FPGA system. The concept is further expanded in Global Asynchronous Local Synchronous (GALS) systems with clocked sub-components [30–32]. In these systems, the limited high-level DI systems possess various timing constraints that are concealed within black boxes, frequently leading to the removal of inherent advantages such as extreme operation temperature range and formal verification capabilities. Regrettably, the timing implications are frequently not comprehended, isochronic signal forks are disregarded or are not systematically ensured by the designer. Consequently, the outcomes violate DI conditions and frequently manifest as self-timed, QDI, SI cases, where the need to check timing assumptions by the designer is easily overlooked [33–37].

In circuit families where isochronic forks are confined to the data path, their "elimination" can be achieved by employing ternary logic gates, thereby removing the fork between the true and false gates of DR logic using a single gate [38].

In a similar fashion, Nyström [39] proposed the incorporation of the isochronic fork within a single binary DR cell. Given that single gates are the hardwired macro cells employed in P&R, this approach exhibits remarkable compatibility with large systems. Similar complex gates where presented by Mokhov *et al.* [40] but ruled out for their (design) overhead and non availability in foundry libraries. More recently, Manohar and Moses [11] showed that, in fact, grouping gates into pairs is sufficient to build arbitrary DI computations, under the assumption that the timing of the hardwired standard cell is known and safe, circumventing the limitations to previously researched DI circuits [8].

In parallel to the development history of DI and QDI circuits, the community developed many automated flows for and approaches for automated synthesis for QDI and BD designs. Both in the early days [41–44] and more modern and recent tool flows [45–52].

III. Our concept for automated DI-like synthesis and layout

Most handshake and phase control implementations [3, 17, 27, 53], C-element based [8] or otherwise, already fulfil the requirements of DI for some, if not all, pipeline control elements. This leaves the data path as the critical outstanding cause for delay sensitivity, timing forks and assumptions. An initial focus on a single process is prudent to tackle the critical datapath.

Toward this end, we take a pragmatic approach. We expect that the final chip implementation will use an automated P&R flow. This means that the placement of individual cells from the cell library will be strategically optimized to minimize the aggregate wire length estimation [54]. Since cells are automatically placed, we do not necessarily have tight control over the wire delays for any input to a cell since those

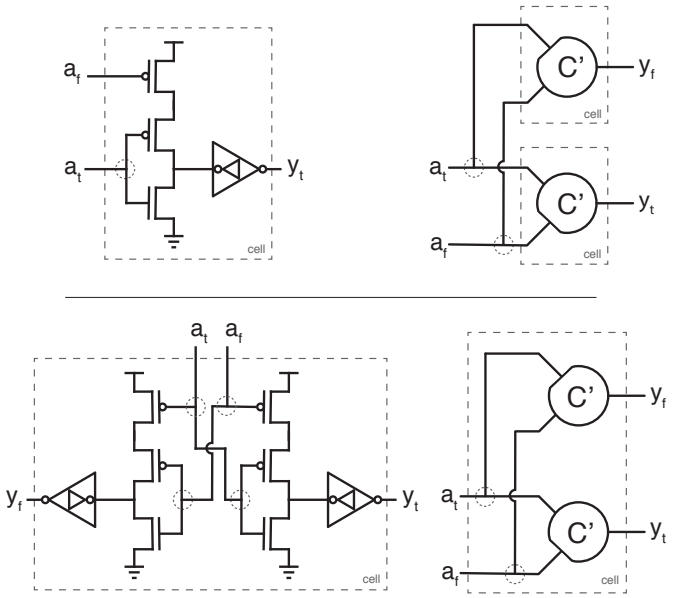

Fig. 1. Depiction of a DR buffer gate in its QDI (top) and DI-like (bottom) variant. The signal forks are marked with circles. As proposed by Manohar and Moses [11] fusing the two state-holding gates into a single cell incorporates the two isochronic signal forks into the cell. Inside the cell with a fixed physical layout, the timing of the signal forks is governed by the same much weaker timing constraint as in single output gates. Transistor sharing and custom cell optimisations similar to Nyström [39] are recommended for cell area optimisation.

will depend on the automatically generated routing. However, *within a cell*, internal wire delays are essentially fixed and controlled by the manual cell design.

To approach the signal forks inside the logic of a single process, we strive to integrate all isochronic forks inside single cells. This objective can be achieved by integrating multiple gates into a single cell that share the same inputs. By explicitly designing the physical attributes of the signal fork based on the layout of the cells, we eliminate all timing uncertainties in P&R that matter for the DI requirement. We effectively replace the undefined timing constraint of an isochronic fork with a much weaker condition of the physically designed wire fork within a cell must, under all operation conditions and fabrication variation statistics, be shorter than the gate delay.

The reader should note that this timing constraint is already present in all standard cell libraries, *even when we consider single output gates*. For example, if we view the internal wire from the input to the gate of the n-transistor and p-transistor of an inverter to be a wire fork and if the two branches of the fork could have arbitrary relative delay, then there could be a short circuit (i.e. an error) during operation. Hence, we already assume that this internal wire fork within a inverter cell is in fact isochronic. This condition is fulfilled for all sufficiently compact Complementary Metal Oxide Semiconductor (CMOS) cells in larger technologies and can be automatically validated during cell characterisation in extracted simulations for state-of-the-art technologies. Starting from QDI data flow methodology [55] and employing 1-of-2 codes for signal wires, also known as DR encoding, enables the construction of

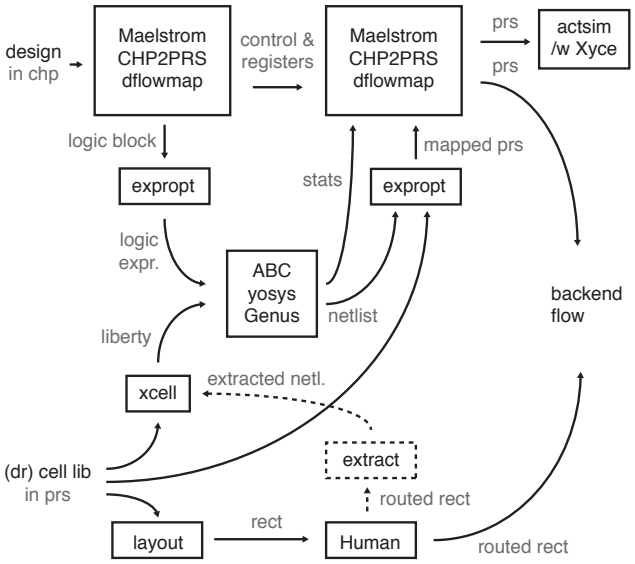

Fig. 2. The EDA synthesis flow shows the integration of logic optimisation into the ACT framework for custom cell libs. The synthesis hands the logic blocks to exropt, which passes them further as Verilog to the chosen standard logic synthesis and mapping tool. After being mapped with the custom liberty cell library definition, the resulting circuit netlist is passed back and converted to the internal data structure of the synthesis tool.

a cell library that fuses just two normal gates into a single cell. This concept is shown in Fig. 1. The size of these cells remains sufficiently compact to employ readily available circuit design tools, enabling automated design. The design of a new cell library or a cell library add-on is a shared burden with other QDI/SI [41, 56–59] and even some BD implementations [60]. The necessary design of the cell library can be semi-automated with existing tools [48]. The construction of DR equivalents of standard logic cells allows for the utilization of established circuit synthesis, logic optimization, and cell mapping techniques. The equivalent DR cells execute the same logic function by integrating two state-holding C-element gates into a single cell with DR in- and outputs. The necessary cell characterization (liberty file) for the synthesis and mapping can be a generic configuration but should follow the intended logic optimisation and mapping flow shown in Fig. 2.

The same concept holds true for register cells; fusing two gates responsible for the true and false wire of the DR variable or read port suffices to hide all isochronic signal forks in the register variable circuits.

To illustrate the automated DI synthesis of processes, the steps previously outlined were integrated into the Maelstrom synthesis software [55]. Maelstrom is a novel open-source tool and synthesis method that transforms Communicating Hardware Processes (CHP) programs into asynchronous circuits. It produces BD circuits with significant improvements in energy efficiency, speed, and size compared to state-of-the-art asynchronous synthesis techniques like Balsa [44] and CHP2PRS [48]. Furthermore, it is circuit family agnostic and able to be extended to support various data path implementations and communication protocols. For the QDI/DI circuit synthesis, we use the standard datapath style, which synthesizes each variable in the program into a single register with the appropriate number of write ports.

Note that Maelstrom only synthesises the control circuits and register units. To synthesize combinational logic, we use the exropt package (seen in Fig. 2), which acts as an interface between the ACT flow and conventional logic optimization software such as ABC [61]. We use ABC, via exropt, to obtain gate-level implementations of a logic expression, following which, the cell definitions within the mapped circuit are simply replaced with BD, QDI or DI equivalents respectively, to obtain the requisite circuit.

In conclusion, for a single process, with a readily available DI or fully C-element based control, DI registers, and the data path modelled with our combined DR cells, the biggest difference between a classic QDI and our proposed DI-like approach is solely on a netlist and P&R cell level, while Production Rule Set (PRS) or transistor representations are nearly identical for single processes.

IV. Multiple processes

Large-scale asynchronous circuits are normally defined as a collection of communicating processes that interact using channel interfaces. QDI circuits use channel communication protocols that are in fact delay-insensitive. Hence, any isochronic fork in such typical QDI designs will necessarily be contained within an individual process. This means that the techniques for developing a DI process in Section III can be used to create a large-scale DI system.

There are two scenarios where there can be a much larger difference between DI and QDI circuits that we discuss next.

A. Shared variables

The first scenario where there is a non-trivial difference between a QDI and DI circuit is if a QDI system is designed with shared variables among the participating processes. In this case, an isochronic fork between processes can be used to hide non-trivial timing requirements. As a concrete example, consider the following CHP program:

$$
\begin{array}{ll}
*[\ L?x;\ \ y := f(x);\ \ R!\ ] & \textit{process}\ \ A \\
\| & \\
*[\ [\overline{R}];\ \ Z!g(y);\ \ R?\ ] & \textit{process}\ \ B
\end{array}
$$

R is a pure synchronization channel, and the variable y is shared between the two processes. This combination produces $g(f(x))$ on the channel Z because of the synchronization operation on channel R. However, if we examine the wires that cross from process A to B, they resemble a bundled-data communication protocol—even though technically this is a shared-variable QDI design! The fact that y in process A is viewed as the same value as the variable y in process B

translates into an isochronic fork requirement at the circuit level, and this fork will span both processes.[1]

To translate such a shared variable program, we can first factor out the shared variable into a separate process, and then proceed as in Section III. We remark that this is often how such CHP programs are translated into QDI circuits as well. In the specific example above, the change would be to introduce process Y as follows:

$$\begin{array}{ll} & *[[\overline{RY} \longrightarrow RY!y \;\; []\, \overline{WY} \longrightarrow WY?y]] \\ \| & *[\; L?x;\;\; WY!f(x);\;\; R!\;] \\ \| & *[\; [\overline{R}];\;\; RY?t; Z!g(t);\;\; R?\;] \end{array}$$

The variable y has been extracted as a single-variable register, and the access channels are used in the two original processes.

Hence, unless the QDI circuit is directly generated with shared variables in place, we can resort to the per-process translation approach outlined above to generate a DI circuit from the CHP program.

B. Physically distributed process

The second scenario where a difference arises between QDI and DI circuits is when an isochronic fork is contained within a process, but the process itself is best implemented in a manner that is *physically* distributed across the chip—for example, for wire planning reasons. A concrete example of such a situation is a split bus.

A split bus is a process that routes a data value to one of a number of destinations, where the destination is specified by a control token. In CHP, this can be written as simply as:

$$*[\; D?d,\;\; C?c;\;\; OUT[c]!d\;] \quad \text{(IV-B.1)}$$

Suppose D is a wide channel with 32-bit data, and the values must be sent to one of many destinations. An example of this is the write data bus in a 32-bit register file, or sending a data value from one register to one of a number of different function units in a CPU datapath (e.g. [41, 56, 57]). If we treat each $OUT[i]$ channel has a separate channel, then the number of wires that must be distributed across the datapath are 64×N (assuming a dual-rail data encoding), where N is the number of destinations. This can easily become prohibitively expensive in wiring cost.

Instead, a better floorplan would be to *distribute* the implementation of this split process, where part of it is physically adjacent to each function unit—making the individual $OUT[i]$ channels *local*, but where a shared set of wires distributes 64 signals across all the function units. This difference is illustrated in Figure 3. In this case, the isochronic fork on the data bus wires is what makes the resulting circuit QDI. The DR signals are only closing the control loop via the acknowledge for the addressed channel; for the non addressed channels it is assumed that they receive the data at the same time (timing constraint).

To preserve the wiring-optimized floorplan, a DI alternative to the split can be implemented as follows: instead of sending

[1]The fork is between the write-completion logic for y in process A, and the gates that use the value of y in process B.

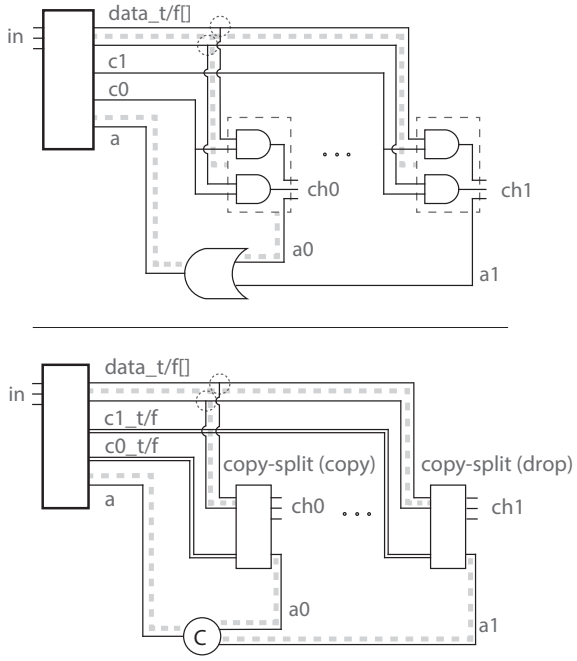

Fig. 3. The distributed split variants: QDI (top - CHP: IV-B.1) and copy-split (bottom - CHP: IV-B.2). Both splits share their DR databus to save wiring space. Thus, the endpoints fork off the signal. The top QDI variant uses the isochronic fork assumption on the shared data path (circle) and requires manual floorplanning/wireing. The variant is "safe" as long as the wire delay from the fork point is smaller than the acknowledge and control roundtrip. The copy-split variant validates the signal on each endpoint, acknowledges receipt, and closes the control loop for each wire fork, adding latency overhead. Each endpoint needs to be told if it should forward or drop the data token instead of only the receiving endpoint in the above variant; thus, the control signal is required as DR.

the data value to just one destination, we send it to *all* destinations along with a control bit that indicates if the destination should accept the data or not. All destinations, so all forks, close the control loop with an acknowledge to remove any isochronic assumption between them. Note that while this may seem extremely costly (multiple data copies for each destination), we can use the same floorplan as shown in Figure 3 to ensure that the output drives just one set of bus wires. Hence, the true overhead will be additional control logic and completion detection. We can write this alternative implementation at the CHP level as follows:

$$\begin{array}{l} *[\; D?d, C?c;\;\; (, i :: Do[i]!d, Dc[i]!(c = i))\;] \\ \| \;\; (\| \;\; i :: \;\; *[\; Do[i]?l_i, Dc[i]?c_i; \\ \qquad\qquad [c_i \longrightarrow OUT[i]!l_i \\ \qquad\qquad []\neg c_i \longrightarrow skip \\ \qquad\qquad]] \\ \quad) \;\; \text{(IV-B.2)} \end{array}$$

The second collection of parallel components corresponds to the per-process circuit that checks to see if the received data is to be delivered locally or ignored. This collection of processes can now be physically distributed without an isochronic fork that spans the length of the bus.

TABLE I

RESULTS FROM SIMULATION PROGRAM WITH INTEGRATED CIRCUIT EMPHASIS (SPICE) SIMULATIONS OF SYNTHESIZED SINGLE PROCESS CHP PROGRAMMS PRESENTED IN SRINIVASAN AND MANOHAR [55] IN A 65NM PROCESS. LOWER IS BETTER ACROSS ALL METRICS. MAELSTROM AND CHP2PRS BOTH USE ABC FOR DATAPATH LOGIC OPTIMISATION AND MAPPING. ALL METHODS ARE 4-PHASE HANDSHAKE VARIANTS, AND ALL DATA VARIABLES AND CHANNELS HAVE A WIDTH OF 8. LATENCY, CYCLE AND ENERGY ARE PRE-LAYOUT SPICE FOR RELATIVE COMPARISONS.

Program	**CHP**	**Method**	**Area** (μm^2)	**Ratio**	**Cycle / Latency** (ns)	**Ratio**	**Energy** (pJ)	**Ratio**	**Unique Cells**
Buffer	$*[L?x;\ R!x]$	Maelstrom DI	1624.09	1	1.694 / 0.988	**1** / 1	0.502	1	18
		Maelstrom QDI	1678.54	1.03	1.703 / 0.991	1.01 / 1	0.503	1	18
		chp2prs QDI	2142.74	1.32	1.720 / 0.938	1.02 / 0.95	0.588	1.17	**13**
		Maelstrom BD	975.30	**0.58**	2.142 / 0.930	1.26 / **0.94**	0.363	**0.72**	14
Sequence	$*[L_1?x_1; R_1!x_1;$ $L_2?x_2; R_2!x_2;$ $L_3?x_2; R_3!x_3;$ $L_4?x_4; R_4!x_4]$	Maelstrom DI	6374.42	1	5.787 / 4.710	**1 / 1**	2.217	1	18
		Maelstrom QDI	6591.79	1.03	5.793 / 4.715	**1 / 1**	2.221	1	18
		chp2prs QDI	8352.13	1.31	6.595 / 5.492	1.14 / 1.17	2.418	1.09	**13**
		Maelstrom BD	3771.17	**0.59**	6.514 / 5.323	1.13 / 1.13	1.472	**0.66**	14
Parallel	$*[L_1?x_1, L_2?x_2,$ $L_3?x_3, L_4?x_4;$ $R_1!x_1, R_2!x_2,$ $R_3!x_3, R_4!x_4]$	Maelstrom DI	6735.48	1	1.990 / 1.087	**1** / 1	2.216	1	19
		Maelstrom QDI	6957.22	1.03	1.992 / 1.088	**1** / 1	2.219	1	19
		chp2prs QDI	8742.24	1.30	2.638 / 1.470	1.33 / 1.24	2.561	1.16	**14**
		Maelstrom BD	4139.63	**0.61**	2.344 / 1.043	1.18 / **0.96**	1.557	**0.70**	16
Split	$*[L?x; C?c;$ $[c = 0 \rightarrow R_1!x$ $[]c = 1 \rightarrow R_2!x\ ]]$	Maelstrom DI	3381.39	1	3.222 / 2.226	1 / 1	0.912	1	20
		Maelstrom QDI	3485.72	1.04	3.243 / 2.241	1.01 / 1.01	0.913	1	20
		chp2prs QDI	3310.85	0.98	2.938 / 1.944	**0.91 / 0.87**	0.748	0.82	**13**
		Maelstrom BD	1958.95	**0.58**	3.247 / 1.953	1.01 / 0.88	0.718	**0.76**	19
Merge	$*[C?c;$ $[c = 0 \rightarrow L_1?x$ $[]c = 1 \rightarrow L_2?x];$ $R!x]$	Maelstrom DI	8478.72	1	4.794 / 3.682	1 / 1	2.182	1	20
		Maelstrom QDI	8951.05	1.06	4.855 / 3.733	1.01 /1.01	2.196	1.01	20
		chp2prs QDI	6140.28	0.73	3.741 / 2.928	**0.78** / 0.80	1.451	0.66	14
		Maelstrom BD	4762.37	**0.56**	4.534 / 2.624	0.95 / **0.71**	1.329	**0.61**	21
Adder	$*[L_1?x_1,$ $L_2?x_2;$ $R!(x_2 + x_1)]$	Maelstrom DI	5987.66	1	3.437 / 2.127	**1** / 1	1.666	1	25
		Maelstrom QDI	6781.51	1.13	3.571 / 2.186	1.04 / 1.03	1.700	1.02	23
		chp2prs QDI	7687.76	1.28	3.811 / 2.334	1.11 / 1.10	1.827	1.10	**18**
		Maelstrom BD	2346.41	**0.39**	3.627 / 1.984	1.05 / **0.93**	0.854	**0.51**	19
Multiplier	$*[L_1?x_1,$ $L_2?x_2;$ $R!(x_2 \cdot x_1)]$	Maelstrom DI	13933.44	1	4.137 / 2.512	**1** / 1	3.578	1	25
		Maelstrom QDI	16299.61	1.17	4.347 / 2.617	1.05 / 1.04	3.685	1.03	27
		chp2prs QDI	17166.24	1.23	4.585 / 2.764	1.11 / 1.10	3.874	1.08	22
		Maelstrom BD	3831.61	**0.27**	4.604 / 2.468	1.11 / **0.98**	1.240	**0.35**	**21**
Division	$*[L_1?x_1,$ $L_2?x_2;$ $R!(x_2/x_1)]$	Maelstrom DI	27274.51	1	9.948 / 5.444	1 / 1	7.321	1	32
		Maelstrom QDI	31616.40	1.16	10.793 / 5.883	1.08 / 1.08	7.677	1.05	28
		chp2prs QDI	32508.09	1.19	11.069 / 6.030	1.11 / 1.11	7.871	1.08	**25**
		Maelstrom BD	5910.51	**0.22**	8.127 / 4.223	**0.82 / 0.78**	2.406	**0.40**	27
Modulo	$*[L_1?x_1,$ $L_2?x_2;$ $R!(x_2\%x_1)]$	Maelstrom DI	33434.12	1	10.283 / 5.655	1 / 1	8.723	1	34
		Maelstrom QDI	38765.67	1.16	11.154 / 6.110	1.08 / 1.08	9.149	1.05	30
		chp2prs QDI	39712.49	1.19	11.389 / 6.258	1.11 / 1.11	8.222	0.97	**25**
		Maelstrom BD	6733.84	**0.20**	8.849 / 4.587	**0.86 / 0.81**	3.271	**0.39**	29
GCD	Appendix A	Maelstrom DI	28331.60	1	60.392 / -	1	49.894	1	35
		Maelstrom QDI	31219.35	1.10	62.861 / -	1.04	50.958	1.02	33
		chp2prs QDI	25985.43	0.92	60.022	**0.99**	31.213	0.63	**27**
		Maelstrom BD	12681.01	**0.45**	71.121 / -	1.18	25.094	**0.50**	35
Fibonacci		Maelstrom DI	23055.38	1	60.380 / -	1	34.061	1	34
		Maelstrom QDI	25507.27	1.11	62.571 / -	1.04	34.616	1.02	30
		chp2prs QDI	23953.75	1.04	53.862 / -	**0.89**	31.094	0.91	**24**
		Maelstrom BD	13014.24	**0.56**	66.679 / -	1.10	23.223	**0.68**	31
Bresenham		Maelstrom DI	49920.95	1	12.185 / -	**1**	10.623	1	40
		Maelstrom QDI	55384.91	1.11	12.636 / -	1.04	10.808	1.04	35
		chp2prs QDI	51574.41	1.03	13.391 / -	1.10	8.754	0.82	**27**
		Maelstrom BD	28096.44	**0.56**	16.851 / -	1.38	7.986	**0.75**	37
Average ratio for primitives	**Maelstrom DI vs. Maelstrom QDI**		**0.96**		**0.99 / 1**		**1**		**1**
	Maelstrom DI vs. chp2prs QDI		0.89		0.97 / 0.99		1.02		1.28
	Maelstrom DI vs. Maelstrom BD		1.71		0.90 / 1.09		1.45		1.15
Average ratio for logic	**Maelstrom DI vs. Maelstrom QDI**		**0.86**		**0.94 / 0.95**		**0.96**		**1.07**
	Maelstrom DI vs. chp2prs QDI		0.82		0.90 / 0.91		0.95		1.29
	Maelstrom DI vs. Maelstrom BD		3.70		1.04 / 1.15		2.42		1.22
Average ratio	**Maelstrom DI vs. Maelstrom QDI**		**0.91**		**0.96 / 0.97** (latency avg.		**0.98**		**1.05**
	Maelstrom DI vs. chp2prs QDI		0.89		0.95 / 0.95 only Prog.		1.04		1.31
	Maelstrom DI vs. Maelstrom BD		2.15		0.92 / 1.11 1 to 9)		1.71		1.15

TABLE II

4-WAY SPLIT VARIANTS AS SEEN IN FIG. 3. THE LOCAL 4-WAY SPLIT ANALOG TO TABLE I, IS USING INDIVIDUAL READ-PORTS AND DOES NOT SHARE WIRES. RESULTS ARE FROM SPICE SIMULATIONS OF SYNTHESIZED CHP PROGRAMMES IN A 65NM PROCESS. LOWER IS BETTER ACROSS ALL METRICS. ALL METHODS ARE 4-PHASE HANDSHAKE VARIANTS, AND ALL DATA VARIABLES AND CHANNELS HAVE A WIDTH OF 8. LATENCY, CYCLE AND ENERGY ARE PRE-LAYOUT SPICE FOR RELATIVE COMPARISONS.

Program	**CHP**	**Method**	**Area** (μm^2)	**Ratio**	**Cycle / Latency** (ns)	**Ratio**	**Energy** (pJ)	**Ratio**	**Unique Cells**
Distributed copy-split (DI)	(IV-B.2)	Maelstrom DI	16957.24	1	**3.005** / 3.059	**1** / 1	5.351	1	23
Distributed QDI split	(IV-B.1)	Maelstrom QDI	6669.99	0.39	3.220 / **2.030**	1.07 / **0.66**	1.741	0.33	24
Local DI split	Appendix A	Maelstrom DI	**6255.22**	**0.37**	3.689 / 2.478	1.23 / 0.81	**1.391**	**0.26**	**20**

TABLE III

AREA COMPARISON FOR MORE COMPLEX DESIGNS INTRODUCED IN SRINIVASAN AND MANOHAR [55].

design	Maelstrom DI	Maelstrom QDI	Maelstrom BD
ALU	16027μm^2 (1)	18692μm^2 (1.17)	3389μm^2 (0.21)
FP16 add	2289μm^2 (1)	2487μm^2 (1.09)	527μm^2 (0.23)

V. RESULTS

This work aims to analyze the discrepancy and overhead associated with the selection of DI-like over QDI. To this end, an examination of single CHP process circuits is presented in Table I. The synthesis of CHP programs is conducted using CHP2PRS with logic optimization by ABC and CHP for Maelstrom for QDI, and a comparison is made with the previously described DI-like synthesis method implemented in Maelstrom. The comparison to syntax-directed-translation (chp2prs) and token-ring QDI (Maelstrom) synthesis supports the broader generalisation of the presented method. The single process programs and the technology are inspired to the ones analyzed in [55] for reference and further comparison with BD circuit variants.

Table I shows almost identical results for QDI and DI-like for the first five test programs. This result is due to the lack of datapath logic and consists solely of control circuitry (and registers), thus being identical in both cases. The subsequent seven cases incorporate computation within the datapath, and as previously observed, the area of the combined cells can be arranged more compactly when adhering to the same density guideline, moreover the slightly differing liberty files play a role in logic optimisation. However, the cycle time and energy remain unaltered apart from transistor scaling, as the PRS remains the same. A minor drawback is evident in the elevated number of custom standard cells necessitated by DI like, whereas QDI can repurpose single output gates for multiple standard logic functions.

A comparison of the synthesized DI-like circuits with the 4-phase BD variants [55] reveals that the cycle time is competitive ($0.92\times$) and the latency not far off ($1.11\times$) for DI-like. This has two reasons. First, the delay line is synthesised with some overhead, and second, it always constitutes the worst-case delay of the logic. In DI like, no timing overhead or save margin is needed, and the timing is the average delay while the additional validity computation reduces this advantage. However, it should be noted that the area and energy requirements are heavily affected by the DR encoding, resulting in significant overhead - $2.15\times$ area ($4.5\times$ for complex circuits) and $1.71\times$ energy. Nonetheless, DI circuits offer distinct advantages, including full timing independence, operational versatility within a broad range, and formal verification down to the layout level.

In all the single process examples, the area of the DI-like variant is slightly superior due to the larger but fewer cells being more compactly packable with less overhead in a grid-based placement system with the same target density (in our experiment 90%).

In contrast, for the case of the physically distributed split with a shared bus, the DI-like variant has a significant penalty in area ($2.56\times$) and energy ($3.03\times$) as seen in Table II. The throughput is better as it is now a pipelined process ($.93\times$). If the split is used with an output First In First Out (FIFO), the overhead in the area of copying the data once is absorbed.

VI. DISCUSSION AND CONCLUSION

This work presents a promising solution to the DI isochronic fork problem in a practical manner for most larger-scale designs employing automated synthesis, a prospect that was previously thought impossible [7, 23]. DI is in the eye of the beholder; by placing isochronic forks within compact standard cells, the isochronic fork timing constraints are substituted by the more manageable fork timing constraints present and satisfied in all standard cell logic gates. This approach effectively isolates the isochronic forks from the unpredictable delays caused by P&R. We emphasize the qualitative distinction between previous approaches, which defined more extensive macros as "timing-safe," and the compact custom, P&R-compatible standard cells proposed in this work. While working with the same principle, the former increases the manual workload of the designer and the risk of unrecognized timing violations. Furthermore, previous approaches lose the ability to ensure time safety across extreme operation conditions.

This method is not universally low-cost and compact. In certain instances, the combination of two gates proves inadequate, as illustrated by the requirement of an additional isochronic fork for certain circuit styles like Pre-Charge Half Buffer (PCHB), necessitating the additional integration of an or2 gate within the DR cell that integrates latching (register) and logic. These multi-gate cells will be larger but could be sufficiently compact, depending on the complexity of the

PCHB logic stage. The considerable overhead associated with distributed copy-splits warrants careful consideration. A cost-benefit analysis can be performed by comparing the overhead to the area performance improvements of the proposed workflow. However, the extent of copy-splits in the design ultimately determines the overhead of this approach.

When it comes to scan chain testing with the composite cells, one might want to treat the cells separate for stuck-at modiling. From a possible coverage perspective the situation remains unchanged as both the scan insertion points as well as the circuit remains unchanged.

The DI abstraction encompasses the majority but not all potential timing-based design flaws in digital systems. For instance, circuits with severely unbalanced drive strengths are susceptible to slope degradation [62]. Consequently, to ensure the efficacy and reliability of circuits, it is imperative to employ proper cell sizing and signal buffering.

Given the substantial enhancement in risk mitigation during the design of larger systems and the comparable or lower overhead with regard to QDI, this DI-like approach is a highly compelling prospect for the future. It has the potential to replace both automated SI and QDI designs on a general basis, offering benefits such as operation temperature independence and reduced effort formal verification.

References

[1] J. D. Cressler and H. A. Mantooth, *Extreme environment electronics*. Boca Raton: CRC Press, Taylor & Francis Group, 2013. DOI: 10.1201/b13001.

[2] D. E. Muller, "Theory of asynchronous circuits," University of Illinois at Urbana-Champaign. Dept. of Computer Science, Internal Report, 1955.

[3] J. Sparsø and S. Furber, Eds., *Principles of asynchronous circuit design: a systems perspective*. Boston, MA: Springer US, 2001. DOI: 10.1007/978-1-4757-3385-3.

[4] R. Keller, "Towards a Theory of Universal Speed-Independent Modules," *IEEE Transactions on Computers*, 1974. DOI: 10.1109/T-C.1974.223773.

[5] R. M. Incandela *et al.*, "Characterization and Compact Modeling of Nanometer CMOS Transistors at Deep-Cryogenic Temperatures," *IEEE Journal Electron Devices Society*, 2018. DOI: 10.1109/JEDS.2018.2821763.

[6] D. Palmer and R. Heckman, "Extreme Temperature Range Microelectronics," *IEEE Transactions on Components, Hybrids, Manufacturing Technol.*, 1978. DOI: 10.1109/TCHMT.1978.1135312.

[7] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," in *Beauty Is Our Business*, W. H. J. Feijen *et al.*, Eds., New York, NY: Springer New York, 1990. DOI: 10.1007/978-1-4612-4476-9_35.

[8] R. Manohar and Y. Moses, "The Eventual C-Element Theorem for Delay-Insensitive Asynchronous Circuits," in *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, San Diego, CA: IEEE, 2017. DOI: 10.1109/ASYNC.2017.15.

[9] J. C. Sims and H. J. Gray, "Design criteria for autosynchronous circuits," in *Papers and discussions presented at the December 3-5, 1958, eastern joint computer conference: Modern computers: objectives, designs, applications on XX - AIEE-ACM-IRE '58 (Eastern)*, Philadelphia, Pennsylvania: ACM Press, 1958. DOI: 10.1145/1458043.1458065.

[10] K. Stevens *et al.*, "Relative timing [asynchronous design]," *IEEE Transactions on Very Large Scale Integr. (VLSI) Systems*, 2003. DOI: 10.1109/TVLSI.2002.801606.

[11] R. Manohar and Y. Moses, "Asynchronous Signalling Processes," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Hirosaki, Japan: IEEE, 2019. DOI: 10.1109/ASYNC.2019.00018.

[12] J. T. Udding, "A formal model for defining and classifying delay-insensitive circuits and systems," *Distributed Computing*, 1986. DOI: 10.1007/BF01660032.

[13] J. C. Ebergen, *Translating programs into delay-insensitive circuits* (CWI tract). Amsterdam: Centrum voor Wiskunde en Informatica, 1989.

[14] C. J. Myers, "Computer-aided synthesis and verification of gate-level timed circuits," PhD Thesis, Stanford University, Stanford, CA, USA, 1996.

[15] P. Patra and D. Fussell, "Efficient building blocks for delay insensitive circuits," in *Proceedings of 1994 IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, UT, USA: IEEE Comput. Soc. Press, 1994. DOI: 10.1109/ASYNC.1994.656312.

[16] J. Lee *et al.*, "Universal delay-insensitive circuits with bidirectional and buffering lines," *IEEE Transactions on Computers*, 2004. DOI: 10.1109/TC.2004.51.

[17] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, 1986. DOI: 10.1007/BF01660034.

[18] A. J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits," *UT Year Programming Institute on Concurr. Programming*, 1989.

[19] C. Van Berkel and R. Saeijs, "Compilations of communicating processes into delay-insensitive circuits," in *Proceedings 1988 IEEE International Conference on Computer Design: VLSI*, Rye Brook, NY, USA: IEEE Comput. Soc. Press, 1988. DOI: 10.1109/ICCD.1988.25682.

[20] C. Jesshope *et al.*, "Compilation of process algebra expressions into delay-insensitive circuits," *IEE Proceedings E (Computers Digit. Tech.*, 1993. DOI: 10.1049/ip-e.1993.0038.

[21] I. E. Sutherland, "Micropipelines," *Communications ACM*, 1989. DOI: 10.1145/63526.63532.

[22] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration*, 1993. DOI: 10.1016/0167-9260(93)90035-B.

[23] K. van Berkel, "Beware the isochronic fork," *Integration*, 1992. DOI: 10.1016/0167-9260(92)90001-F.

[24] R. Manohar and Y. Moses, "Analyzing Isochronic Forks with Potential Causality," in *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, Mountain View, CA, USA: IEEE, 2015. DOI: 10.1109/ASYNC.2015.19.

[25] R. Manohar and A. J. Martin, "Quasi-delay-insensitive circuits are Turing-complete," Department of Computer Science California Institute of Technology, Pasadena, CA, USA, Tech. Rep., 1995. DOI: 10.21236/ada444284.

[26] N. Sretasereekul and T. Nanya, "Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits," in *Proceedings of the 2001 conference on Asia South Pacific design automation - ASP-DAC '01*, Yokohama, Japan: ACM Press, 2001. DOI: 10.1145/370155.370450.

[27] H. Park *et al.*, "Modular Timing Constraints for Delay-Insensitive Systems," *Journal Computer Science Technol.*, 2016. DOI: 10.1007/s11390-016-1613-y.

[28] H. Saito *et al.*, "What is the cost of delay insensitivity?" In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, San Jose, CA, USA: IEEE, 1999. DOI: 10.1109/ICCAD.1999.810668.

[29] K. Meekins *et al.*, "Delay insensitive NCL reconfigurable logic," in *Proceedings, IEEE Aerospace Conference*, Big Sky, MT, USA: IEEE, 2002. DOI: 10.1109/AERO.2002.1036908.

[30] F. Rosenberger *et al.*, "Q-modules: Internally clocked delay-insensitive modules," *IEEE Transactions on Computers*, 1988. DOI: 10.1109/12.2252.

[31] S. Dasgupta *et al.*, "Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits," *Electronic Notes Theoretical Computer Science*, 2006. DOI: 10.1016/j.entcs.2005.05.037.

[32] E. Brunvand and R. Sproull, "Translating concurrent programs into delay-insensitive circuits," in *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, Santa Clara, CA, USA: IEEE Comput. Soc. Press, 1989. DOI: 10.1109/ICCAD.1989.76949.

[33] D. R. Ghica and A. Smith, "Geometry of Synthesis II: From Games to Delay-Insensitive Circuits," *Electronic Notes Theoretical Computer Science*, 2010. DOI: 10.1016/j.entcs.2010.08.018.

[34] A. Moradi *et al.*, "Dual-rail transition logic: A logic style for counteracting power analysis attacks," *Computers & Electrical Engineering*, 2009. DOI: 10.1016/j.compeleceng.2008.06.004.

[35] S. C. Smith *et al.*, "Speedup of Delay-Insensitive Digital Systems Using NULL Cycle Reduction," in *Proceedings of the 2001 International Workshop on Logic and Synthesis (IWLS'01)*, Granlibakken, CA, USA, 2001.

[36] A. D. Bailey *et al.*, "Ultra-low power delay-insensitive circuit design," in *2008 51st Midwest Symposium on Circuits and Systems*, Knoxville, TN, USA: IEEE, 2008. DOI: 10.1109/MWSCAS.2008.4616846.

[37] S. Smith *et al.*, "Delay-insensitive gate-level pipelining," *Integration*, 2001. DOI: 10.1016/S0167-9260(01)00013-X.

[38] R. Mariani *et al.*, "On the realisation of delay-insensitive asynchronous circuits with CMOS ternary logic," in *Proceedings Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Eindhoven, Netherlands: IEEE Comput. Soc. Press, 1997. DOI: 10.1109/ASYNC.1997.587152.

[39] M. Nyström, "A Surprisingly Delay-Insensitive Circuit," Personal communication - ASYNC fresh idea, Tech. Rep., 2014.

[40] A. Mokhov *et al.*, "On Dual-Rail Control Logic for Enhanced Circuit Robustness," in *2012 12th International Conference on Application of Concurrency to System Design*, Hamburg, Germany: IEEE, 2012. DOI: 10.1109/ACSD.2012.17.

[41] A. Martin *et al.*, "The design of an asynchronous MIPS R3000 microprocessor," in *Proceedings Seventeenth Conference on Advanced Research in VLSI*, Ann Arbor, MI, USA: IEEE Comput. Soc, 1997. DOI: 10.1109/ARVLSI.1997.634853.

[42] H. Jacobson *et al.*, "High-level asynchronous system design using the ACK framework," in *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000) (Cat. No. PR00586)*, Eilat, Israel: IEEE Comput. Soc, 2000. DOI: 10.1109/ASYNC.2000.836975.

[43] J. Cortadella *et al.*, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on information Systems*, 1997.

[44] D. Edwards and A. Bardsley, "Balsa: An Asynchronous Hardware Synthesis Language," *The Computer Journal*, 2002. DOI: 10.1093/comjnl/45.1.12.

[45] P. A. Beerel *et al.*, "Proteus: An ASIC Flow for GHz Asynchronous Designs," *IEEE Design & Test Computers*, 2011. DOI: 10.1109/MDT.2011.114.

[46] R. B. Reese *et al.*, "Uncle - An RTL Approach to Asynchronous Design," in *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems*, Kgs. Lyngby, Denmark: IEEE, 2012. DOI: 10.1109/ASYNC.2012.14.

[47] A. Yakovlev *et al.*, "Advances in Asynchronous Logic: From Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD Tools," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, Grenoble, France: IEEE Conference Publications, 2013. DOI: 10.7873/DATE.2013.346.

[48] S. Ataei *et al.*, "An Open-Source EDA Flow for Asynchronous Logic," *IEEE Design & Test*, 2021. DOI: 10.1109/MDAT.2021.3051334.

[49] C. Nielsen *et al.*, "Yak: An Asynchronous Bundled Data Pipeline Description Language," in *2023 28th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Beijing, China: IEEE, 2023. DOI: 10.1109/ASYNC58294.2023.10239566.

[50] H. Wu *et al.*, "A Design Flow for Click-Based Asynchronous Circuits Design With Conventional EDA Tools," *IEEE Transactions on Computer-Aided Design Integrated Circuits Systems*, 2021. DOI: 10.1109/TCAD.2020.3038337.

[51] A. Godard *et al.*, "Standard EDA tools based asynchronous design flow," in *IP-SoC Conference 2024*, Grenoble, France, 2024.

[52] M. Nyström *et al.*, *Async-toolkit*, Github, 2025.

[53] M. Singh and S. M. Nowick, "MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines," *IEEE Transactions on Very Large Scale Integr. (VLSI) Systems*, 2007. DOI: 10.1109/TVLSI.2007.898732.

[54] Y. Yang *et al.*, "Dali: A gridded cell placement flow," in *Proceedings of the 39th International Conference on Computer-Aided Design*, Virtual Event USA: ACM, 2020. DOI: 10.1145/3400302.3415689.

[55] K. Srinivasan and R. Manohar, "Maelstrom: A Logic Synthesis Technique for Asynchronous Circuits," *IEEE Transactions on Computer-Aided Design Integrated Circuits Systems*, 2026. DOI: 10.1109/TCAD.2025.3572364.

[56] C. Kelly *et al.*, "SNAP: A Sensor-Network Asynchronous Processor," in *Ninth International Symposium on Asynchronous Circuits and Systems, 2003. Proceedings.*, Vancouver, BC, Canada: IEEE Comput. Soc, 2003. DOI: 10.1109/ASYNC.2003.1199163.

[57] C. T. O. Otero *et al.*, "ULSNAP: An ultra-low power event-driven microcontroller for sensor network nodes," in *Fifteenth International Symposium on Quality Electronic Design*, Santa Clara, CA, USA: IEEE, 2014. DOI: 10.1109/ISQED.2014.6783391.

[58] A. Neckar *et al.*, "Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model," *Proceedings IEEE*, 2019. DOI: 10.1109/JPROC.2018.2881432.

[59] O. Richter *et al.*, "Speck: A Smart event-based Vision Sensor with a low latency 327K Neuron Convolutional Neuronal Network Processing Pipeline," in *28th IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, Beijing, China: accepted and presented, 2023. DOI: 10.48550/ARXIV.2304.06793.

[60] T. Jagielski *et al.*, "Integrating Asynchronous Circuits into the Caravel Testing Harness," 2024.

[61] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*, D. Hutchison *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-14295-6_5.

[62] F. Ouchet *et al.*, "Delay Insensitivity Does Not Mean Slope Insensitivity!" In *2010 IEEE Symposium on Asynchronous Circuits and Systems*, Grenoble, France: IEEE, 2010. DOI: 10.1109/ASYNC.2010.27.

APPENDIX A
CHP FOR PROCESSES USED IN TABLE I AND II

The cycle time of below examples denotes the time between succesive output data tokens being produced. The CHP for the iterative GCD computation program:

$$*[\ X?x; Y?y; *[x > y \longrightarrow x := x - y \ [] \ y > x \longrightarrow y := y - x]; O!x]$$

The CHP for the 2n-th Fibonacci number generation program:

$$*[N?n, x := 0, y := 1; *[n > 0 \longrightarrow x := x + y; \ y := x + y; n := n - 1]; O!x]$$

The CHP for the Bresenham's Line Drawing Algorithm ($x_0 \leq x_1$):

$$\begin{array}{l} *[X_0?x_0, X_1?x_1, Y_0?y_0, Y_1?y_1; dy := y_1 - y_0, \\ \quad dx := x_1 - x_0; D := 2(dy) - dx; \\ \quad *[x_0 \leq x_1 \longrightarrow P_x!x_0, P_y!y_0; \\ \qquad [D > 0 \longrightarrow y_0 := y_0 + 1, D := D - 2(dx) \\ \qquad [] else \longrightarrow skip]; D := D + 2(dy), x_0 := x_0 + 1] \ ] \end{array}$$

The CHP for the 4-way local split:

$$*[L?x; C?c; \ [c = 0 \longrightarrow R_1!x \ [] c = 1 \longrightarrow R_2!x \ [] c = 2 \longrightarrow R_3!x \ [] \ else \longrightarrow R_4!x]]$$