Received December 6, 2015, accepted December 23, 2015, date of publication January 4, 2016,
date of current version January 15, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2514398

Design Flow and Characterization Methodology for Dual Mode Logic

VIACHESLAV YUZHANINOV, ITAMAR LEVI, AND ALEXANDER FISH
Emerging Nanoscale Integrated Circuits and Systems Labs, Faculty of Engineering, Bar-Ilan University, Ramat Gan 5290002, Israel
Corresponding author: V. Yuzhaninov (yuzhans@biu.ac.il)

ABSTRACT Recently, the dual mode logic (DML) family was introduced as a superior energy-delay alternative to CMOS. DML gates utilize two different modes of operation, dynamic and static, to selectively achieve either high-performance or low-energy operation. Custom designs of DML circuits have been shown to be very efficient. However, implementing DML circuits using the standard design flow and Electronic Design Automation (EDA) tools is very challenging, since DML gates operate in two different modes, each with its own characteristics and operating mechanisms. This paper shows, for the first time, that DML logic can be compatible with the standard design flow and optimized by various tools, such as synthesis and physical design. A DML cell library characterization methodology is also proposed to support the design flow. The methodology and flow were verified on a wide variety of benchmark designs with different gate counts and logic depths, and show that DML design is efficient under the standard design flow restrictions.

INDEX TERMS Standard design flow, alternative logic family, dynamic logic, dual mode logic (DML).

I. INTRODUCTION

Digital logic families have been extensively researched; since the late 70's a variety of static [1], [2] and dynamic [3]–[5] logic families have been introduced. Research efforts have mainly targeted tradeoff analyses between different logic styles in terms of design metrics such as energy, performance, area utilization, robustness and standard-flow compatibility. While performance is usually regarded as the gold standard for dynamic approaches (trading off the rest), static logic (especially CMOS) is slower, but more robust, easily applicable and compatible with standard flow and tools.

Standard static-logic compatible Computer Aided Design (CAD) flows have been researched for almost 30 years. Diverse algorithms and heuristics have been proposed and implemented for each step and tool of the design flow and many abstractions have been added to reduce complexity [6]–[8]. These CAD tools have reached a high level of integrity, yielding cross-verified quality results and short time-to-market. However, fundamental design challenges arise when the logic under the scope of the automation tools is dynamic-based [9]–[13].

The recently proposed Dual Mode Logic (DML) provides a hybrid (dynamic and static) functionality at the gate level, as reported in [14]–[17]. DML shows superior robustness, high noise immunity and low leakage in nano-scaled technologies. In addition, DML performs faster than CMOS if operated dynamically, and consumes less power in the static mode. DML is based on a static core gate, preferably CMOS, and has a single clocked transistor to enable its high performance dynamic mode. The complementary network (which usually does not present in standard dynamic logic) endows DML with superior robustness compared to the vulnerability of pure dynamic logic families. Furthermore, the complementary network enables a functional (but slower than CMOS) static mode of operation [16]. It is important to note that DML gate transistors are uniquely sized to achieve all the above [18].

In this paper we introduce a design flow for Dual Mode Logic for the first time that also overcomes a few of the well-known dynamic design challenges. This automation method is implemented on a dedicated DML standard cell library which was constructed and fully characterized for this study. The DML-Flow offers fully automation-compatible steps, innovative approaches and the use of standard tools, which results in a flexible design in terms of energy-delay tradeoffs by utilizing the inherently different operational modes of DML.

The contributions of this work consist of the:

i. Formulation of a DML standard-cell-library characterization methodology and an option to utilize it within a design-flow.
ii. Adaptation of the standard-design-flow (STDF) to DML, while utilizing well-known commercial Electronic Design Automation (EDA) tools to address the unique needs of DML.

The structure of this paper is as follows: Section II introduces Dual Mode Logic and presents the challenges involved in its adaptation to the automated ASIC design flow. Section III describes the digital design flow for DML, and Section IV covers the DML library characterization methodology. Section V summarizes the results of this DML standard library characterization and EDA flow. Section VI concludes the paper.

II. CHARACTERIZATION AND STANDARD DESIGN FLOW CHALLENGES

This section provides a self-contained overview of DML and presents some of the challenges that arise during DML library characterization and digital flow integration as discussed in Sections III and IV.

A. DML OVERVIEW

DML logic was proposed to allow controllable switching between static and dynamic modes of operation. As was presented in [14], this dual modularity offers greater performance *vs.* energy tradeoff flexibility. This flexibility can be tuned, as required, by system or input-driven control, or by designer considerations. The dynamic mode enables higher performance with moderate energy consumption, whereas the static mode has very low static and dynamic energy consumption with moderate performance. The two functional modes of DML have different energy-delay (E-D) optimal curves, whereas the ability to change modes on-the-fly makes it possible to attain the E-D targets of their union, providing extended optimization space [16], [19]. It has been shown in a number of publications [14]–[17] that DML is fully functional, robust and efficient at wide range of supply voltages.

This behavior is the result of the special structure and operation of a basic DML gate, as shown in Fig. 1. A static core gate is supplemented with an additional pre-charge (or pre-discharge) transistor M_1, which enables dynamic operation. A full swing (rail-to-rail or R2R) of the static logic core gate is highly valuable; which is why a CMOS gate was chosen. During static operation, the M_1 transistor is disabled and the DML gate operates like the static logic core gate.

Similar to other dynamic logic families, DML gates can be implemented with or without a footer (or header) transistor. Fig. 1 shows all the possible DML configurations: Type-A with its pre-charge operation and Type-B with its pre-discharge operation. Hereafter, only the footless topology is discussed, unless noted otherwise. Generally, footed gates are rarely used because of their negative impact on gate performance and area.

Operating the DML gate in the static mode is highly intuitive, in that the pre-charge transistors need to be disabled: the CLK_A port is set at HIGH and the CLK_B set at LOW. As a result, the gate retains the functionality of its static core gate, except for an extra negligible parasitic capacitance due to the additional transistor.

The DML full R2R property is highly desirable given its lower leakage power and superior robustness compared to

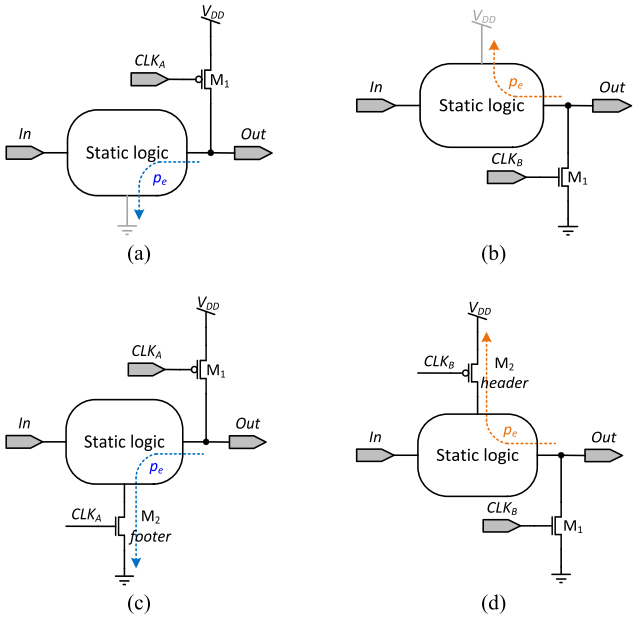


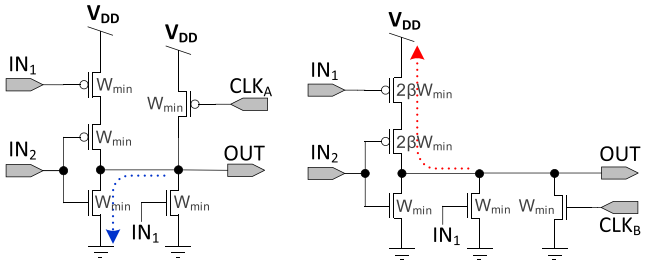
FIGURE 1. Basic DML gates topologies: (a) Type-A footless. (b) Type-B footless. (c) Type-A footed. (d) Type-B footed. p_e traces represent the evaluation paths for each topology.

standard dynamic logic. A DML dedicated transistor sizing methodology was introduced in [18] that dealt with CMOS-based gates. This unique sizing scheme results in a substantial performance gain during the dynamic mode relative to its CMOS counterpart. It presents moderate energy consumption during the static mode (*i.e.* the optimal dynamic mode sizing for performance meets the semi-optimal static mode sizing for energy minimization). Similarly, sizing optimization for high performance can be done on any other static logic family (typically by trading off energy).

As shown in [18] (in particular for CMOS based gates) the most efficient DML gates are typically the ones with a pre-charge (or pre-discharge) transistor connected in parallel to a group of serially stacked transistors, which are minimally sized (whether pull-up or pull-down). Therefore, the evaluation network is usually dominated by parallel paths, which contribute to a very fast evaluation period (small evaluation path resistance and reduced output capacitance). In general, the designer is not obliged to use these guidelines and the pre-charge transistor can be placed in parallel to a parallel paths network, but this will result in relatively slow DML gates (compared to the opposite type). Hence, in order to fully exploit the DML advantages, specific gates are better utilized in certain types. Fig. 2 illustrates this principle, where a DML Type-A NOR_2 gate is very fast in comparison to a DML Type-B NOR_2 gate. The reader is referred to [18] for more detailed information.

B. STANDARD DESIGN FLOW CHALLENGES WITH RESPECT TO DML

This sub-section reviews the STDF challenges with respect to DML structure and standard design tool capabilities.



(a) Efficient Type-A NOR_2 gate (b) Less efficient Type-B NOR_2 gate

FIGURE 2. DML NOR_2 (a) Type-A (b) Type-B.

Automated ASIC design-flows have been governed since their early days by the CMOS digital logic-family for a variety of reasons, including superior robustness and noise margins, rail-to-rail logic levels, one-way directivity and low leakage. Nowadays, the main virtue of static CMOS is its *correct-by-design* compatibility with EDA tools, which have evolved greatly over the years since changes in the logic family can lead to unexpected and deteriorated results from EDA tools. When using CMOS, only a small set of design rules need to be applied for successful EDA integration. This simplifies the automation tool complexity. Needless to say, the majority of standard EDA tools nowadays are also completely oriented to static CMOS designs. The ASIC industry adheres to a well-defined and tested STDF throughout the entire design cycle, starting with the product specification definition up to submission of production files to the foundry. A typical flow fragment of the design phase (verification excluded) is shown in Fig. 3 [8]:

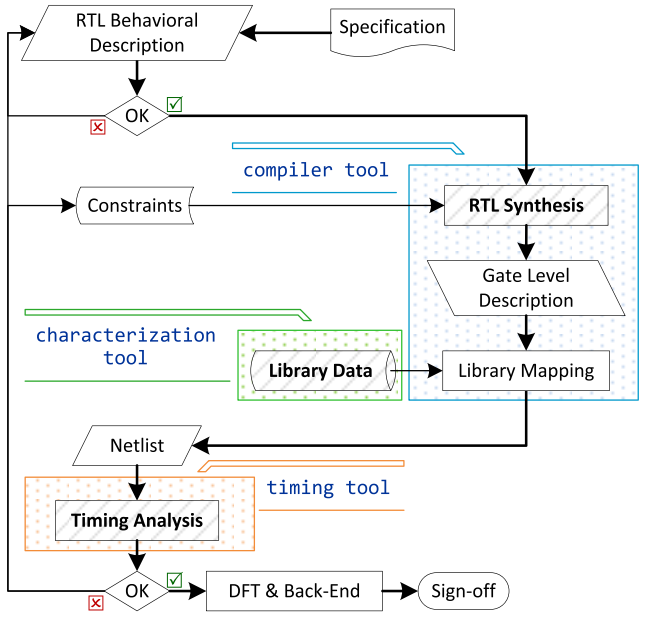


FIGURE 3. Simplified ASIC Standard Design Flow.

The key STDF stages are:

i. *RTL Synthesis* – conversion of RTL to generic gates and registers, while optimizing the logic efficiency and then mapping it to a real library of characterized cells by revamping it for best design metrics.

ii. *Standard Cell Library* – a library of real, laid-out and characterized logic gates for the use of the synthesizer mapping process. These include geometric, timing and power metric data.

iii. *Static Timing Analysis (STA)* – Evaluation of all timing paths within the logic networks of the design and monitoring any timing constraint violations.

For simplicity, a DML gate can be depicted as a static CMOS gate that is able to operate dynamically when an active clock is applied to it. On the other hand, an inactive clock signal will degenerate the DML gate to its CMOS counterpart. This abstraction could be exploited as a starting point for a static STDF adaptation. However:

i. DML operates statically or dynamically, so the design flow must distinguish between these two substantially different cases.

ii. As introduced in sub-section II.C.1), the composition of dynamic logic networks is subject to the bipartite criterion; hence, not all statically valid designs are dynamically credible.

C. DYNAMIC OPERATION MODE DESIGN CHALLENGES

In this sub-section, we highlight the main challenges of designing in DML's dynamic mode (note that typically, standard dynamic logic challenges are a subset of these obstacles). We present the challenges as a foundation for Sections III and IV, in which the proposed approach for characterization and design flow (DF) integration tackles these challenges.

1) NON-UNATE BOOLEAN FUNCTIONS

According to [6], a function is unate in all its variables if and only if it is either monotonically increasing or monotonically decreasing for all of its variables.

Monotonicity in variable x_1 described as:

$$f_{inc}(0, x_2 \ldots, x_n) \leq f_{inc}(1, x_2 \ldots, x_n), \quad \forall (x_2, \ldots, x_n)$$
$$f_{dec}(0, x_2 \ldots, x_n) \geq f_{dec}(1, x_2 \ldots, x_n), \quad \forall (x_2, \ldots, x_n)$$

Thus, the unateness and monotonicity terms are interchangeable throughout this article.

The non-unate functions are of great importance for dynamic logic based designs because implementing them can imply an area increase and the addition of logically redundant gates. In what follows, we clarify this point: dynamic logic takes advantage of absent or degraded complementary evaluation networks [3]–[5], and thus propagates faster than **C**MOS (**C**omplementary MOS) logic. All dynamic logic styles must apply a proper cascading policy of evaluation networks to ensure correct data propagation [1]. Fig. 4(a) illustrates an example of improper cascading of dynamic gates and the subsequent corruption of the propagating data. The example shows that right after the pre-charge (start of evaluation) of serially connected nMOS evaluation

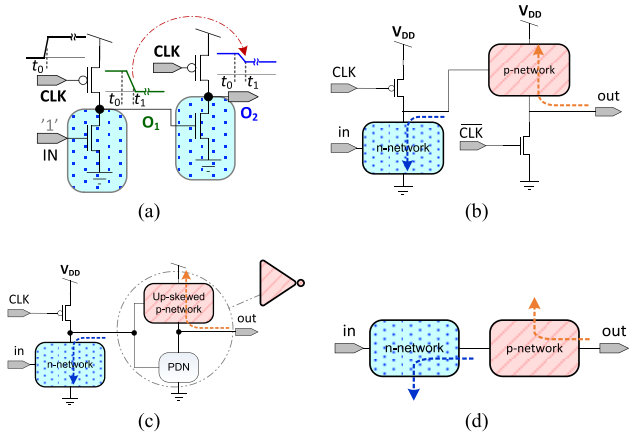


FIGURE 4. Color classification of dynamic logic cells. (a) Improper dynamic cascading link. (b) np-CMOS (NORA) cascading link. (c) n-Domino cascading link. (d) Generic dynamic cascading link.

network based dynamic gates, the second gate will start discharging regardless of the actual output to be evaluated at the output of the first gate.

This concept can be generalized even further. If a logic gate with a favored nMOS evaluation network is referred to as an **A***mber* vertex (Type-**A** in DML terminology), and a gate with a pMOS evaluation network is referred to as a **B***lue* vertex (Type-**B** in DML terminology), then a correct dynamic logic cascading exists if every vertex has complementary colored predecessor and successor vertices. This concept has its analogy in graph theory and is known as a two-colored graph or *bipartite* [20]. Fig. 4 depicts the color classification of the (b) np-CMOS, (c) n-Domino logic families and (d) generic dynamic style.

The output logic level of a non-unate Boolean function $f_{\text{nu}}(\vec{\mathbf{x}}) = f_{\text{nu}}(x_1, \ldots, x_i \ldots, x_n)$ will evaluate to logical '*0*' or '*1*' depending on the change of x_i and the status of the other inputs $x_{j\neq i}$ [21]. This implies a re-convergence of paths with unbalanced odd and even numbers of logic stages prior to the non-unate stage (gate of re-convergence), G_f. See Fig. 5, where $G_0, G_1, \ldots, G_n$ represent logic levels from the primary inputs to the output $f_{\text{nu}}(\vec{\mathbf{x}})$. A non-unate function can always be represented as in Fig. 5, where the difference of logic levels between two re-convergent branches must be odd.

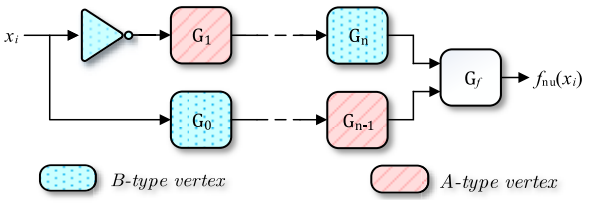


FIGURE 5. Typical network fraction of non-unate Boolean function.

As depicted in Fig. 5, a logic network of a non-unate Boolean function must have at least two re-convergent paths with an odd difference of logic-depths. This conflicts with the cascading policy detailed above, and none of the CMOS based dynamic logic styles is able to implement these functions as is. Unfortunately, among these directly inapplicable sets of non-unate Boolean functions are a few vital functions such as XOR or MUX. The only conventional generic method to deal with this restriction, unless a clock signal is also used to control the data flow, dictates duplication of the logic cones prior to the non-unate function node, thus transforming its logic cone into a *monotonic network* [6], [22].

Logic duplication is also closely related to the "trapped inverter" problem in Domino logic [13], where both polarities of the input signals are required simultaneously, while stand-alone inverters are not available. This method is assisted by recursive unate transformations (*bubble pushing*) before the duplication so as to push the trapped inversion stages down the logic path to the primary inputs (PI), thus preserving the unateness. The primary inputs of a particular dynamic logic domain are defined as its input ports adjacent to other logic domains. Fig. 6(a) shows a general logic structure of a non-unate function and Fig. 6(b) shows the same function after duplication and bubble pushing, where the $*$ symbol denotes the unate transformed logic, and Λ stands for a logical cone.

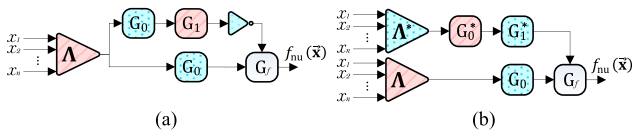


FIGURE 6. Logic cones duplication and bubble pushing. (a) Before. (b) After.

2) DYNAMIC OPERATION CHARACTERIZATION CHALLENGES

In this sub-section, we briefly present the main characterization challenges of a DML standard cell library (*.lib*).

a: CHARACTERIZATION OF ASYMMETRIC BEHAVIOR

As mentioned, DML is capable of operating the gates in the dynamic mode, which runs with synchronization to the clock signal. Hence, some adjustments to the standard static characterization are essential. In contrast to static CMOS logic, the evaluation of data throughout a dynamic logic network is asymmetric (*i.e.* always performs only one *high-to-low* or *low-to-high* transition); thus the assessment of propagation delay, input capacitance and dynamic power must be tailored to differentiate and capture only the specific transitions of logic cells (typically the evaluation path under every input transition and gate-topology).

b: INTER-DATA TIMING RELATIONS

Typically, the timing closure provided by static synthesis tools for combinational cells does not incorporate information on how to handle a synchronizing input (clock signal) to the gates. Dynamic-mode DML characterization requires a definition of new inter-data timing relationships to address this point, as will be elaborated in Section IV.

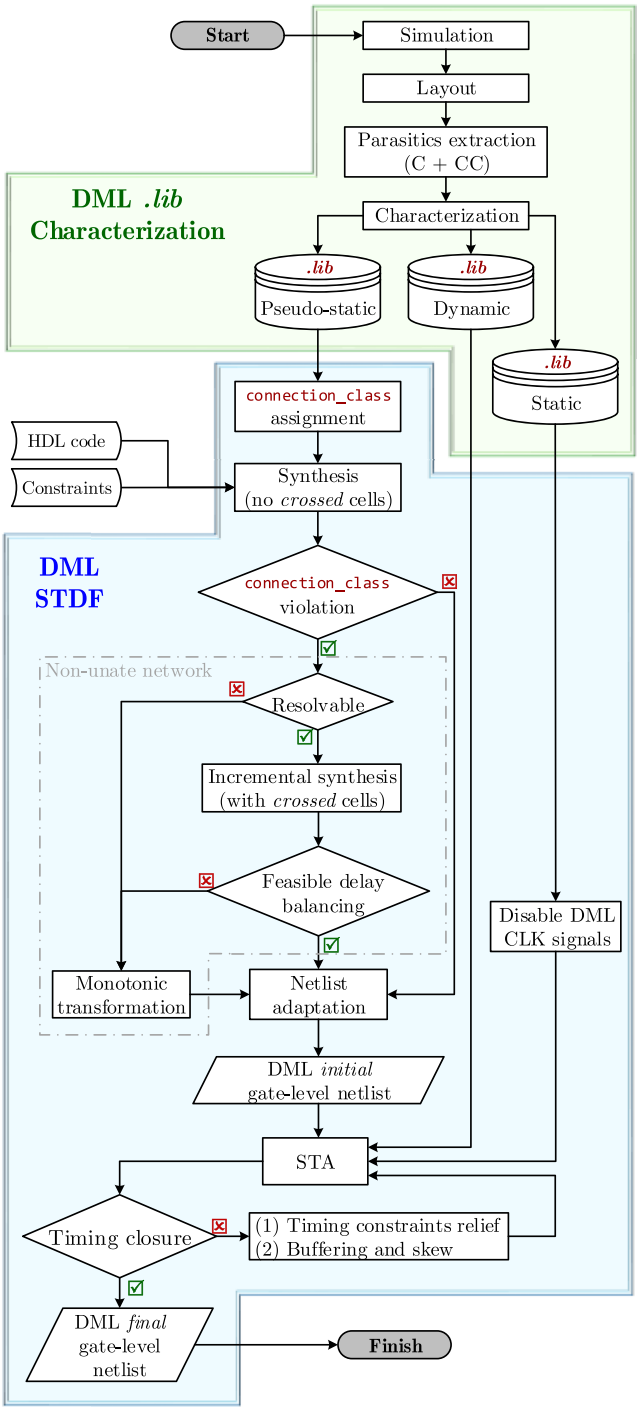


FIGURE 7. Proposed DML standard design flow chart.

III. PROPOSED DESIGN FLOW STEPS FOR DML

This section covers the proposed dynamic DML flow step by step, and then details and explains the rationale for each according to the design flow chart presented in Fig. 7, which summarizes the entire workflow, including partitioning into characterization and design flow. Sub-section III-A explains the rationale behind the use of a dummy *pseudo-static* library during synthesis and mapping. Sub-section III-B details the synthesis and mapping of the DF, including its associated complexities, whereas III-C and III-D conclude the DML DF with few netlist adaptations and STA, respectively.

A. PSEUDO-STATIC LIBRARY AND MULTI-LIBRARY REPRESENTATION

As explained in sub-section II-B, both of the DML's functional modes must be verified separately; thus each of the modes requires its own library (with same gates) describing the characteristics induced by the operating mechanism of the mode. A static library is almost identical to a standard CMOS library, whereas a dynamic library is more complex, as it also contains the timing relationships of synchronizing clock signal *vs.* the data pins.

As noted, the standard synthesis tools are static, but the dynamic mode of DML involves clock synchronization. Assertion of a clock signal, which is used to bring the dynamic logic to a pre-defined initial condition, is called a pre-charge (or pre-discharge) value. This preset phase is followed by a logic evaluation phase during which the clock is inactive and the logic state is determined only by data pins. Recall that an inactive clock degenerates the DML to its static form; therefore, a dynamic DML gate acts the same way as a static gate in terms of data propagation during the evaluation phase. The data start their propagation from a pre-defined state, and are recurrently pulled up and down by evaluation networks of the gates throughout the logic path (see Fig. 8).

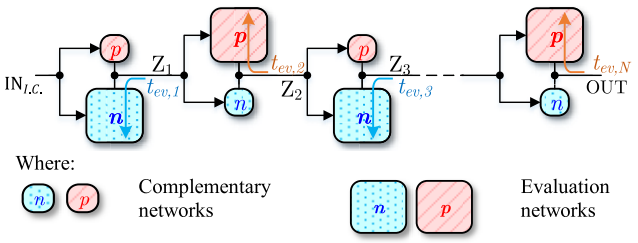


FIGURE 8. DML data propagation during the dynamic evaluation phase.

The arrival evaluation delay at the destination logic OUT node is bounded by the summation value of all transitions on the path (see the left hand term of the next equation):

$$t_{ev,OUT} \leq \sum_{i}^{N_n} t_{ev,i}^{\{n\}} + \sum_{j}^{N_p} t_{ev,j}^{\{p\}} \Bigg|_{\left\{ \begin{array}{l} t_{ev}^{\{n\}} = t_{ev}^{\{p\}} = t_{ev} \\ N_n + N_p = N \end{array} \right\}} = \sum_{i}^{N} t_{ev,i}$$

t_{ev} – evaluation delay
N – logic depth of data propagation path
n, p indices – denote PDN and PUN respectively

Standard synthesis and timing tools are incapable of computing this evaluation delay. An intuitive way we propose here to manipulate the automatic tool is to duplicate the evaluation network delay (pull up or down transition) values to the complementary transition of the same gate, which will

make the tool take on the correct value regardless of transition direction (as though the gate were static), thus simplifying the timing analysis. For example: a Type-A gate always evaluates from high-to-low; therefore the characterized (simulated) high-to-low delay will be copied to the low-to-high delay tables despite the fact that this transition is not possible in dynamic operation. Hence, this type of library characterization is entitled *pseudo-static*. This form of characterization can be treated as an enforced symmetric $\left(t_{ev}^{\{n\}} = t_{ev}^{\{p\}}\right)$ dummy *pseudo-static .lib* library of DML cells (as shown in the previous equation). This library is used for dynamic synthesis while using a *static* tool. This yields an initial candidate design which is post processed later on by two additional libraries for complete timing checks of the static and dynamic modes. The details on the construction of these libraries can be found in sub-section III-B.

B. PSEUDO-STATIC SYNTHESIS AND LIBRARY MAPPING

As introduced above, dynamic logic networks must apply a set of rules to ensure their functionality. This set of *connection* constraints is applied by reading the pseudo-static *.lib* content and setting the Synopsys LIBERTY's `connection_class` attribute [23] on each of the library cell's data pins. The attribute is considered a design rule and should be followed by the synthesis tool during mapping, otherwise `connection_class` violations are reported. The mapping process can be enforced to apply these connections. The connection rules of ordinary DML cells defined according to Fig. 9 ensure correct cascading.

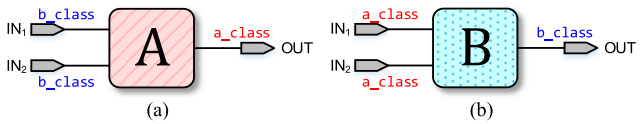


FIGURE 9. DML regular footless cell connection classes. (a) Type-A cell. (b) Type-B cell.

It should be noted that the LIBERTY `connection_class` attribute support is neglected by the Synopsys Design Compiler, which was initially utilized for voltage island design. However, it is part of the LIBERTY gate-level library modeling industry standard, which is compatible with all EDA mapping tools if they choose to use it. The `connection_class` is one way to abide by the connection restrictions. However, clearly, designers can implement it differently. Each pin of each library cell is explicitly attributed to a single valid `connection_class` before the mapping, since it is a hard task for the tool to abide by all the constraints simultaneously if they were loosely specified. For this purpose, several dummy cells are defined to separate different valid connection scenarios, whereas all the connection classes are defined distinctly on the cell level.

All primary inputs should be imposed with footed cells; otherwise, functionality might be lost. This distinction of PIs is made by the assignment of a dedicated PI `connection_class`. All the input pins of cells that are authorized to link to these primary inputs are assigned to the PI class. Cells that are valid to link to primary inputs are all footed or semi-footed. *Semi-footed* refers to a cell that has a serial stack of transistors in the evaluation path; if so, then one of its inputs might also directly interact with the primary inputs (as long as the other input can cut off the evaluation network at the beginning of evaluation phase).

As noted in II.C.1, not all Boolean functions can be realized with a *bipartite* network; thus the workflow should be capable of generating correct designs for a) monotonic and b) non-unate functions as well.

1) MONOTONIC (UNATE) NETWORK MAPPING

In the case of a monotonic logic network, the mapping process is trivial since no color conflicts are observed, and results in a clean report of `connection_class` violations and valid logic network structures.

2) NON-UNATE NETWORKS MAPPING

Not all logic networks can be classified as monotonic; therefore color conflicts are inevitable in the case of a non-unate Boolean function realization. There are two strategies to cope with non-monotonic networks: making it monotonic [13] and a multi-phased clocking scheme [13], [24]. The latter method is not evaluated in this work, since it complicates the design and its clock trees considerably. The monotonic transformation of the logic network is considered only an alternative solution if the following attempts (in order) fail to construct a feasible and valid design.

The report of `connection_class` violations is used to locate the color conflicting nodes of the non-unate logic network. This includes violating the `pin_name` of the conflicting gates and their `cell_name`, which is parsed and serves for the automatic resolution by replacement procedure, where possible.

Before demonstrating the removal procedure of `connection_class` violations, we introduce the cell-naming nomenclature. The output class of the cell is denoted by a capital A or B letter prefix. The class of each IN_i input of the cell is denoted by lowercase *a* (***a***mber) or *b* (***b***lue) letters reflected by appending the `cell_name` with suffix letters by their order of appearance. For example, a Type-A $NAND_2$ cell is abbreviated `A_ND2_bb`, and a Type-A crossed $NAND_2$ cell written `A_ND2_ab` describes IN_1 that should be driven by a Type-A cell, whereas IN_2 should be driven by a Type-B cell. The input connectivity suffix of ordinary DML cells can be omitted, as all the inputs have the opposite connection class of the cell's output. In contrast, *crossed* cells have a combination of input classes, so more information is required.

Fig. 10 illustrates an example of a `connection_class` violation removal procedure, where the conflicting $NAND_2$ cell is replaced by its *crossed* dummy version.

Resolvable conflicting sites are repaired by replacing the conflicting cells with their dummy *crossed* cell clones. On the

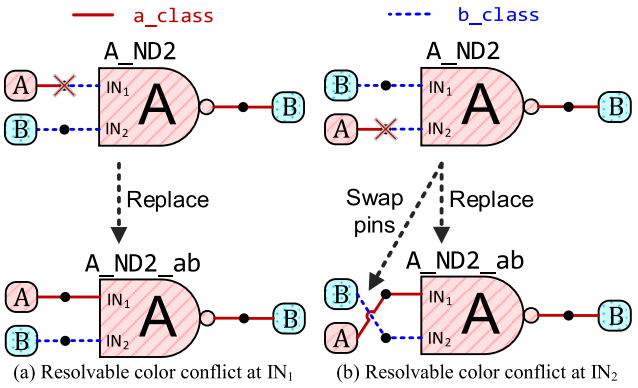


FIGURE 10. Resolvable color conflicts by *crossed* cells. (a) Resolvable color conflict at IN_1. (b) Resolvable color conflict at IN_2.

other hand, unresolvable color conflicts require a monotonic transformation.

Resolvable color conflicts are resolved by reinitiating an additional incremental synthesis run, whereas the entire logic network other than `connection_class` violators is frozen for changes (the `dont_touch` attribute is set). This way, the logic structure remains intact except for insertion of *crossed* dummy cells and delay balancing buffers to meet data-to-data timing constraints. The operation of the *crossed* cells is described in the next paragraph.

a: CROSSED (STACKED) GATES

Some logic gates have the ability to absorb two conflicting colors if a particular input condition is met. The basic idea behind these *crossed* cells takes advantage of a serial transistor stack inside the evaluation path. This serial stack allows for a mixture of two signals with opposite pre-charged states, as one of them cuts off the evaluation path and prevents the gate's false evaluation. Unfortunately, this solution depends on the data-to-data timing interdependence of the gate's inputs. To see this case, refer to Fig. 11, which illustrates the case of a Type-A (**A**mber) $NAND_2$ gate.

Fig. 11 (b) and (c) waveforms show two cases of a cell's correct and false response, respectively. The correct response

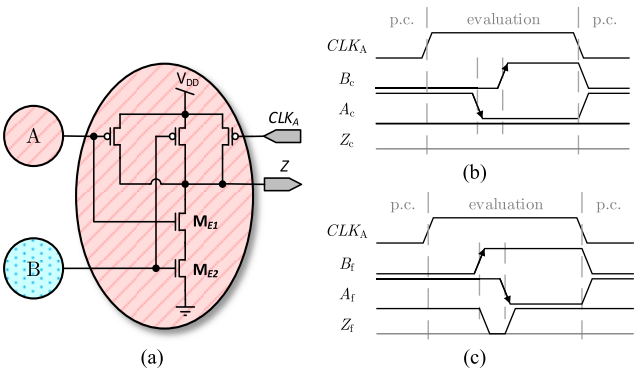


FIGURE 11. $NAND_2$ crossed cell example. (a) Crossed Type-A $NAND_2$ cell. (b) Correct evaluation response. (b) Correct evaluation response.

does not include a transient glitch of output node Z unlike in the failed response, since the input signals do not have a HIGH level overlap. A failed response of the Z output node is observed when both the A and B signals are HIGH and trigger an erroneous evaluation. This input data timing criterion imposed by the mapping tool with respect to the data-to-data timing constraints is defined in the pseudo-static *.lib*. In the case of a Type-A crossed cell, the earliest rise $\min\{t_{rise}(B)\}$ of the Type-B input should occur later than the latest fall $\max\{t_{fall}(A)\}$ of the Type-A input. If this condition is fulfilled, the *crossed* cell behaves as expected.

Note that a failed response is transient and necessarily converges to its correct value after the arrival of both inputs. However, this incurs a false pull-down and then a pull-up evaluation, which propagates through the rest of the logic nodes and forces them to evaluate the data via their degraded complementary networks, resulting in a much slower response. This behavior is unacceptable, as it deviates from the modeled timing frame of the entire design while consuming unnecessary power.

b: DELAY BALANCING

The delay-balancing step assesses the feasibility of meeting the *crossed* cell input data-arrival timing constraint at a reasonable cost (detailed at the end of the paragraph). For instance, the automatic insertion of a number of buffers to meet crossed cell inter-data timing might be far more efficient than duplication of the entire preceding logic cone. If this solution is classified as infeasible, the netlist is recovered in its initial state (no *crossed* cells or delay balancing). In this work, a criterion of a maximum 10% gate count increase was used. Note that this arbitrary threshold parameter is up to the designer, based on the design specifications.

c: MONOTONIC TRANSFORMATION

Impractical *crossed* cell color conflict removal leads to the somewhat costly solution of a monotonic transformation of part of the logic network. In contrast to the widespread dynamic Domino logic design style [3], there is no need for recursive bubble pushing due to trapped inversion stages [13], since the proposed DML flow is based on discrete inverting logic stages. Therefore, the monotonic transformation only consists of preceding logic cone duplication at conflicting nodes. Logic duplication is done with a `monotonic_transform` script that parses the gate-level netlist and replicates the sub-networks prior to the splitting points of the `connection_class` violated cells and re-colors (swap of types) them complementarily (as introduced by Fig. 6(b)). The duplication script has the ability to minimize logic redundancy by constantly updating and re-using already cloned sub-networks, hence saving energy and area.

C. POST-SYNTHESIS NETLIST ADAPTATION

At this point, the pseudo-static gate-level netlist is clear of connection rule violations and is ready to be associated with

real DML libraries. This is easily done by several procedural steps:

i. CLK_A and CLK_B signals are added to the global module port list.

ii. CLK_A and CLK_B signals are added to the port list of each DML gate locally, and globally propagated to the higher hierarchy via the port list of the module.

This way the full structural DML netlist is ready to be analyzed for timing with a standard STA tool.

D. STATIC TIMING ANALYSIS (STA)

As detailed in sub-section V-A, the dynamic *.lib* contains various clock related timing constraints, which should be verified for proper dynamic functionality. The clock affinity is absent in standard STA tools; hence the clock port of each DML cell is defined as a dummy data-port. These inter-data timing relationships are reminiscent of the conventional *setup* and *hold* constraints during the characterization of sequential elements. These timing constraints are satisfied just like standard timing constraints by relaxing the operation frequency and delay corrections which can be fixed with proactive buffering or Place & Route tools. The STA of a static library has no timing issues by default as long as the operating frequency is met.

IV. PROPOSED CHARACTERIZATION OF DML LIBRARIES

This section focuses on the construction of the multiple standard cell DML libraries required for the proposed DF, as summarized in Fig. 7.

The implemented cascading topology of the DML network was chosen to be similar to np-CMOS [5] (see Fig. 4(b)), which utilizes both types (colors) of gates, providing more optimization space for the synthesis tool. For simplicity and a proof-of-concept, only a small, but universal set of logic gates was constructed. This set was comprised of $NAND_2$, NOR_2 gates and inverters of both the A and B types; each had several flavors as detailed below.

Beyond color-coding, the DML cells were divided into three additional sub-categories:

i. *Footless* cells – ordinary logic cells, extensively applied unless a particular condition is encountered.

ii. *Footed* or *semi-footed* cells – logic cells that have a clock controllable evaluation path. These cells are required for interfacing other non-DML logic domains. Semi-footed cells have one controllable evaluation path and another ordinary footless evaluation branch; this structure provides an additional degree of delay optimization during the mapping process.

iii. *Dummy* cells – auxiliary cells, logically identical to other footless cells and used to force the mapping tool to abide with inter-cell connection rules (see Type-A $NAND_2$ example in Fig. 10).

Table 1 lists the entire set of implemented cells characterized throughout the flow.

TABLE 1. Implemented DML cells with connection classes.

Cell name	IN_1 class	IN_2 class	OUT class	*Notes*
A_INV	Type-B		Type-A	
A_INV_f	PI (Primary Input)		Type-A	*Footed*
A_ND2	Type-B	Type-B	Type-A	
A_ND2_ab	Type-A	Type-B	Type-A	*Crossed*
A_ND2_pb	PI	Type-B	Type-A	*Stacked*
A_ND2_f	PI	PI	Type-A	*Footed*
A_NR2	Type-B	Type-B	Type-A	
A_NR2_sf	PI	Type-B	Type-A	*Semi-footed*
A_NR2_f	PI	PI	Type-A	*Footed*
B_INV	Type-A		Type-B	
B_INV_f	PI		Type-B	*Footed*
B_ND2	Type-A	Type-A	Type-B	
B_ND2_sf	PI	Type-A	Type-B	*Semi-footed*
B_ND2_f	PI	PI	Type-B	*Footed*
B_NR2	Type-A	Type-A	Type-B	
B_NR2_ab	Type-A	Type-B	Type-B	*Crossed*
B_NR2_pa	PI	Type-A	Type-B	*Stacked*
B_NR2_f	PI	PI	Type-B	*Footed*

`<type>_<cellname>_<connectivity_suffix>`
`<type>` – notes the type (color) of the cell, which implies pre-charge value.
`<cell_name>` – abbreviation of logic function and a corresponding fan-in.
`<connectivity_suffix>` – outlines the inputs connectivity notation:
- `f` – footed cell
- `sf` – semi-footed cell
- `ab` – IN_1 has Type-A source, while IN_2 has Type-B source.
- `pa`,`pb` – IN_1 has PI source, while IN_2 has Type-A or Type-B source.

As introduced in sub-section III-A, the DML flow requires three different libraries of characterized cells:

i. Pseudo-static *.lib* – auxiliary library used for the construction of valid logic networks.

ii. Dynamic and static *.lib* – real libraries, which characterize the design metrics of DML cells in different operational modes.

The entire set of DML gates was simulated, laid out, extracted for parasitic elements and represented in the form of a SPICE netlist for further characterization. The following sub-sections go into the details of the characterization process and reference its content to the proposed design automation flow. Note that characterization of only one type (A) of DML gates is described, since the other type is characterized along the same lines as the general procedure.

A. PSEUDO-STATIC LIBRARY

The idea behind a pseudo-static library of standard cells is to imitate the dynamic behavior of DML cells without clock synchronizing. In addition, it is also subject to enforcement of connection rules for the generation of dynamically compatible logic networks.

1) DESIGN METRIC CHARACTERISTICS

In order to resemble dynamic behavior, cells within the library were characterized solely for relevant evaluation transitions; *i.e.*, input capacitance, intrinsic propagation delay and power consumption assessed during high-to-low (Type-A) or low-to-high (Type-B) evaluation transitions. Complementary dummy timing arcs artificially duplicated these characterized metrics:

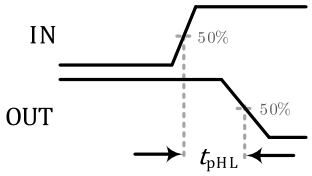


FIGURE 12. Intrinsic high-to-low evaluation delay definition.

1. Intrinsic high-to-low evaluation delay, measured relative to the data IN port Fig. 12.

$$i_{ds,n} = C_{load}\frac{dv}{dt} \Rightarrow t_{ev}^{\{n\}} = \int_0^{t_{pHL}} dt = C_{load}\int_{V_{DD}}^{\frac{V_{DD}}{2}} \frac{dv_{out}}{i_{ds,n}(v_{out})}$$

2. Data input capacitance: $C_{IN} = \frac{\int_0^T i_{IN}(t)dt}{V_{DD}}$
3. Power consumption: $P_{total} = P_{leak} + P_{dyn}$, where
   a. $P_{dyn} = \frac{V_{DD}}{T}\int_0^T i_{DD,dyn}(t)\,dt$
   b. $P_{leak} = V_{DD}{\cdot}I_{DD,leak}$
4. Area footprint was taken from the layout.

Part of these standard cell characteristics were simulated as a function of a parametric two or one-dimensional grid of the IN node transition slope and the capacitive load of the OUT node. These simulations were arranged in tables that can be linearly interpolated or extrapolated by the tool.

2) DATA-TO-DATA TIMING CONSTRAINTS

As introduced in sub-section II-C.2 data-to-data timing constraints are required to generate valid dynamic non-monotonic logic networks. These constraints are used to apply a skew relationship between data pins. The definition of inter-data timing constraints is once again shown with a familiar example of a crossed $NAND_2$ cell. As depicted in Fig. 13, a skew must separate the arrival of two conflicting types to prevent a race of the Z node. This constraint is defined by the `timing_type` group of `non_seq_*` parameters of the Synopsys LIBERTY library modelling standard format [23], [25].

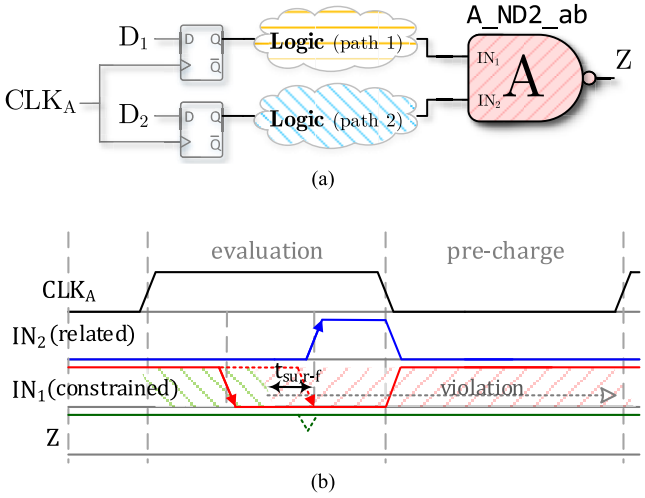


FIGURE 13. Definition example of data-to-data timing constraint. (a) Data-to-data timing constraint data path example. (b) Data-to-data timing constraint waveforms example.

In order to define the skew timing constraint between the rise (evaluation) of the Type-B IN_1 input and the fall (evaluation) of the Type-A IN_2 input, the `non_seq_setup_rising` timing constraint was defined relative to the IN_1 pin, whereas the IN_2 pin governed by the `fall_constraint` parameter values was denoted as $t_{su,r-f}$ in Fig. 13.

These timing parameters were simulated and derived with the characterization tool according to appropriate criteria such as OUT node voltage drop, excessive current drawn, *etc.*

B. DYNAMIC LIBRARY

The design characteristics described in pseudo-static *.lib* are somewhat different from the dynamic *.lib* because of the distinction between the data evaluation and pre-charge phases that have completely different goals.

1) DESIGN METRIC CHARACTERISTICS

For the evaluation phase period in the clock cycle, the characterization is identical to the pseudo-static *.lib*. On the other hand, during the pre-charge phase there are some changes in the design metric assessments:

1. The intrinsic pre-charge delay is measured relative to the CLK edge and is irrelevant in terms of data propagation, but only for propagation of the pre-charged state Fig. 14:

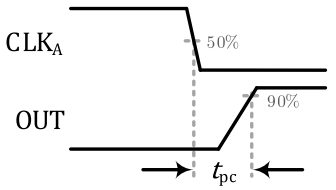


FIGURE 14. Intrinsic pre-charge delay definition.

2. The data input capacitance C_{IN} is also irrelevant, as its discharge does not affect data propagation.
3. The CLK port input capacitance C_{CLK} is averaged to assess its switching: $C_{CLK} = \frac{\int_0^T i_{CLK}(t)dt}{V_{DD}}$
4. Power consumption: $P_{av} = P_{leak} + P_{dyn}$, where
   a. $P_{dyn} = \frac{V_{DD}}{T}\int_0^T i_{DD,dyn}(t)\,dt$, where all the current drawn is due to a transient short circuit caused by non-imminent propagation of the pre-charge state.
   b. $P_{leak} = V_{DD}{\cdot}I_{DD,leak}$, where no stable short circuit condition is assumed.

2) TIMING CONSTRAINTS

The main focus of the dynamic DML library characterization is to identify its cells' timing constraints which are defined to guide the tools for proper dynamic functionality. One group of timing constraints has already been covered in the pseudo-static library sub-section; namely the data-to-data constraints.

Its characterization is identical. This group aims to enforce the skew relationship between constrained data signals.

The other group of constraints can be classified as data-to-clock, but is only an abstraction, since DML clock pins are categorized as data pins as well. This group has various timing constraints that are reminiscent of the classic *setup* and *hold* constraints of sequential components. All of the following timing constraint parameters are characterized to be integrated into the Synopsys LIBERTY library modelling standard format [23].

Setup/Hold parameters – Timing parameters preventing data signal transitions within safety margins before/after edges of CLK_A, which are intended to isolate the evaluation and pre-charge phases and avoid data corruption.

C. STATIC LIBRARY

The standard CMOS-like library characterizes the design metrics of DML cells when gate clock signals are disabled.

V. RESULTS

This section summarizes the results of the proposed DML characterization methodology and its design flow.

A. CHARACTERIZATION

To estimate the quality of the results on a *.lib* cell level, some fundamental design metrics of all the library cells were compared to their CMOS counterparts with same technology node.

1) PERFORMANCE

The evaluation delay data of the dynamic DML library was compared to the CMOS propagation delays on top of a two dimensional grid of transition slope *vs.* capacitive load vectors. The Fig. 15, Fig. 16 and Fig. 17 speed-up results consolidate the early theoretical assessment of DML's dynamic performance superiority.

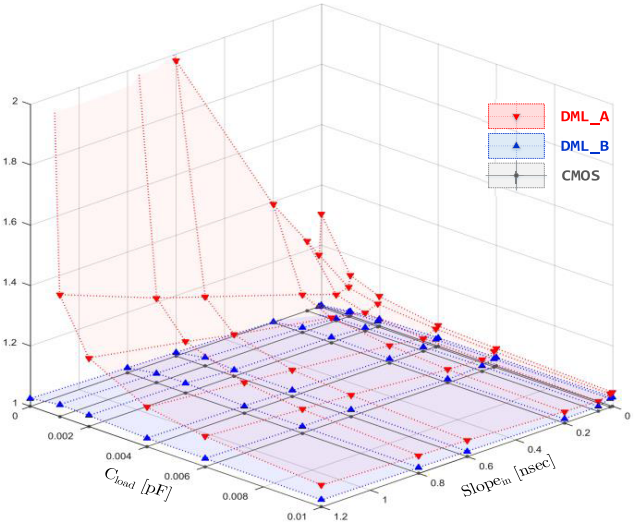


FIGURE 15. Speed-up of dynamic DML inverter vs. CMOS.

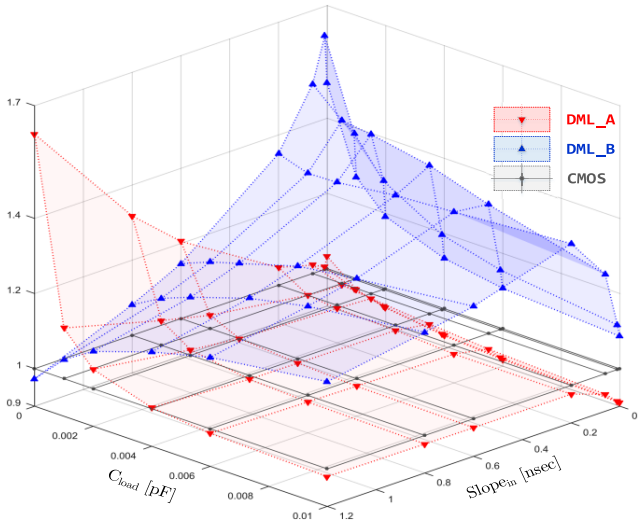


FIGURE 16. Speed-up of dynamic DML $NAND_2$ vs. CMOS.

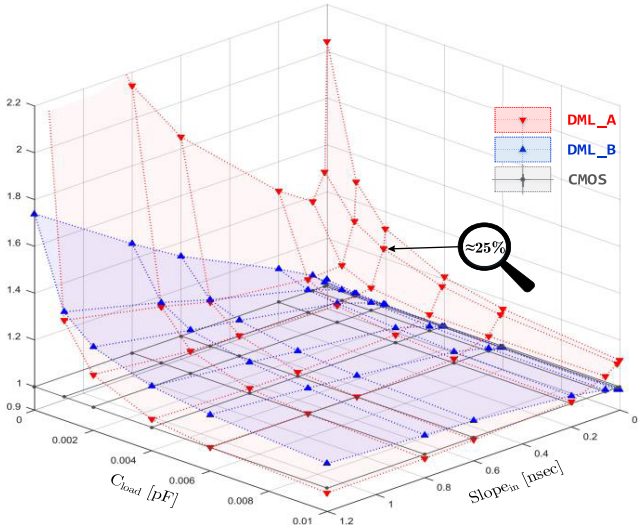


FIGURE 17. Speed-up of dynamic DML NOR_2 vs. CMOS.

All the surface plots show a similar dynamic speed-up pattern, where the most efficient type of DML cell (Fig. 15-16) displays a significant performance boost, whereas the least efficient type of cell exhibits at worst the same speed as CMOS counterpart. For example, a NOR_2 Type-A cell presents about a 25% performance gain for a nominal capacitance of 2 [fF] and a rise time of 10 [psec].

2) AREA AND LEAKAGE

Area and leakage are usually closely related, as can be seen in Fig. 18. It should be noted that the average leakage during the dynamic mode of DML exhibited similar behavior, but was assessed somewhat differently, since the pre-charge combination has a more dominant weight over the rest, due to its half cycle duration.

3) EQUIVALENT INPUT CAPACITANCE

Recall that switching energy is linearly related to the equivalent capacitance $\left(E_{sw} \propto \alpha C_{eq} V_{DD}^2\right)$, which is dominated by

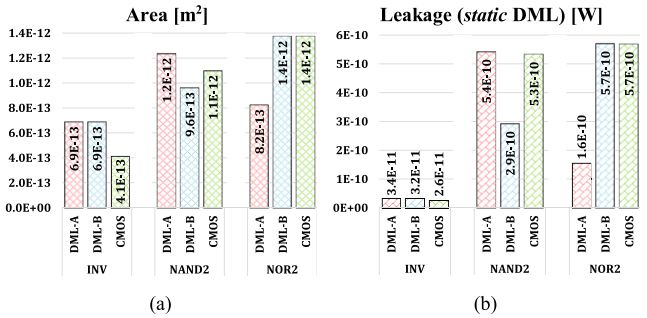


FIGURE 18. Area and Leakage comparison of cells. (a) Area of cells. (b) Leakage of cells.

the input DATA and CLK capacitances of the cells. Fig. 19 shows that DML cells are more efficient in terms of data switching, but continuous ($\alpha = 1$) clock toggling tips the scales in favor of CMOS.

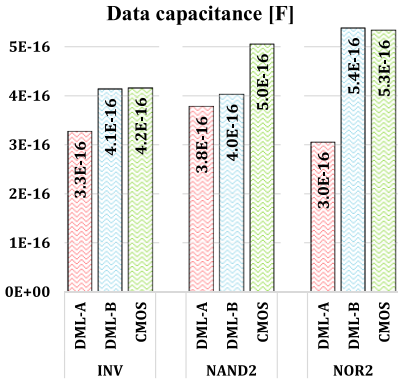


FIGURE 19. Input capacitance and switching energy of cells.

B. DESIGN FLOW

The evaluation of the DML automated design flow results was based on the synthesis of various combinational RTL benchmarks. These benchmarks included typical logic constructs such multiplexers, decoders, comparators, adders, complex Boolean logic functions with varying fan-in, etc. Two concurrent synthesis processes were executed on the basis of DML *vs.* CMOS equivalent standard libraries. In order to prevent a black box uncertainty regarding the commercial CMOS library, it was independently laid out and re-characterized with similar conventions. Both libraries included identical minimal logic sets: $NAND_2$, NOR_2 and an inverter. The results can be divided into two groups: those that include and exclude non-unate logic. Table 2 and Table 3 summarize the average design metrics of the evaluated unate and non-unate benchmarks for several gate count ranges.

Unate logic benchmarks presented better performing designs in terms of dynamic performance and slight power savings compared to the CMOS counterparts while operated statically. For example, the combinational barrel shifter presented in Table 4 dynamically sped up by 10%, with a power shift of about 20% in the static mode, whereas its gate count

TABLE 2. Average metrics of unate designs vs. CMOS, segmented by gate count ranges.

Gate count range (sample space)	0-50 (20)	50-100 (15)	100-250 (10)	250-500 (5)
Area expansion	+12.5%	+6.9%	+5.8%	+3.1%
Gate count	+4.4%	+1.3%	+5.0%	-0.8%
Power shift (*stat.*)	-17.4%	-20.1%	-22.1%	-28.9%
Power shift (*dyn.*)	+315%	+323%	+299%	+340%
Speedup (*dyn.*)	+7.8%	+9.9%	+11.2%	+10.4%
Slowdown (*stat.*)	-32.7%	-31.1%	-37.7%	-41.5%

TABLE 3. Average metrics of non-unate designs vs. CMOS, segmented by gate count ranges.

Gate count range (sample space)	0-50 (20)	50-100 (15)	100-250 (10)	250-500 (5)
Area expansion	+20.2%	+41.2%	+36.4%	+47.8%
Gate count	+4.3%	+11.8%	+15.2%	+21.8%
Power shift (*stat.*)	-3.20%	+3.40%	-0.70%	+5.20%
Power shift (*dyn.*)	+324%	+302%	+315%	+351%
Speedup (*dyn.*)	+7.4%	+8.4%	+16.1%	+12.2%
Slowdown (*stat.*)	-36.0%	-40.3%	-32.9%	-38.3%

TABLE 4. Design metrics comparison vs. CMOS of unate designs.

	Barrel shifter 16 bits	AND 128 bit	4to16 decoder	Avg. of 50 BMs
Area expansion	4.7%	1.9%	9.8%	8.5%
Gate count	3.4%	-2.3%	21.4%	3.1%
Power shift (*stat.*)	-20.4%	-24.7%	-6.0%	-20.3%
Power shift (*dyn.*)	243.0%	224.8%	305.7%	316%
Speedup (*dyn.*)	10.2%	9.4%	9.6%	9.4%
Slowdown (*stat.*)	-29.7%	-39.1%	-32.7%	-34.1%

TABLE 5. Design metrics comparison vs. CMOS of non-unate designs.

	4 bit magnitude comparator	4 bit CLA adder	16 bit dual priority decoder	Avg. of 50 BMs
Area expansion	49.9%	50.6%	37.9%	32.5%
Gate count	20.8%	15.1%	28.7%	10.5%
Power shift (*stat.*)	-7.2%	-11.4%	-0.9%	-3.6%
Power shift (*dyn.*)	301%	282%	327%	318%
Speedup (*dyn.*)	11.5%	12.0%	3.8%	9.9%
Slowdown (*stat.*)	-59.4%	-25.0%	-37.9%	-36.9%

and area expanded only about 4%. In addition, 50 generic combinational logic designs (with no particular functionality) were simulated and showed an average dynamic speedup of about 9%, a static power saving in the region of 20%, a similar gate count and a minor area expansion of 8%.

On the other hand, non-unate benchmarks usually only had a dynamic performance gain, while lagging behind in terms of area and presenting similar power consumption in static mode (due to timing constraints, which enforce delay insertion or logic duplication). For example, the dual priority decoder presented in Table 5 had same static mode power consumption as CMOS due to a logic redundancy of about 21%, but still had a dynamic speedup of about 4%.

Fig. 20 presents the E-D plane of the typical DML benchmarks relative to the static CMOS and graphically highlights the minor performance gains (linear x-axis) of the dynamic

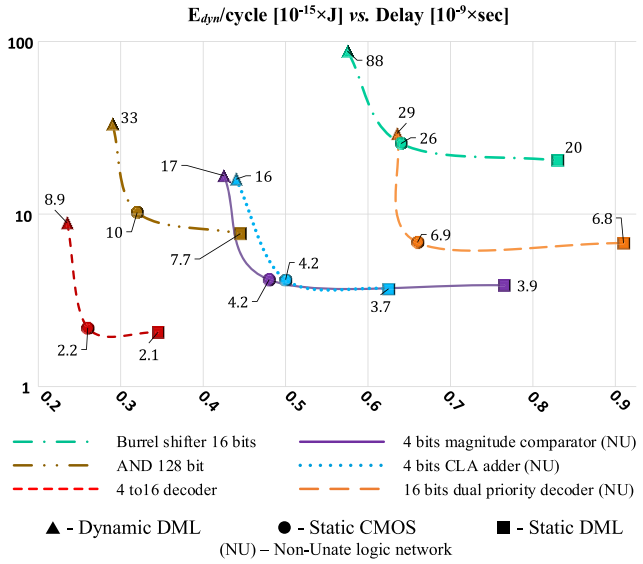


FIGURE 20. **E-D plane representation of typical benchmark designs.**

mode, while consuming much more power (logarithmic y-axis). The static mode exhibited much less drastic behavior, and minor power savings at the expense of a moderate slowdown. The striking power consumption increase of dynamically operated DML was associated with a continuous refreshing of the entire logic networks at CLK speeds. Thus, the dynamic performance boost of DML designs should be carefully optimized to prevent persistent loads.

VI. CONCLUSION

This work presents a novel DML design-flow supported by a sophisticated and unique DML characterization methodology of both DML modes. Despite the numerous challenges of dynamic logic characterization and design, the automated flow generates timing compliant netlists and exhibits improved results in terms of design metrics, as compared to CMOS. Custom DML have been shown to be very efficient. The results here indicate that DML design flow also enables the exploitation of DML advantages while providing a reasonably simple characterization and design flow. It is noteworthy that by allowing voltage scaling on top of DML mode controllability, an extended E-D range can be achieved. The design of DML in conjunction with Dynamic Voltage and Frequency Scaling (DVFS) can be performed in exactly the same way as with standard CMOS. This means that the DML library should be characterized for a desired number of voltages and the physical implementation flow should be the same.

REFERENCES

[1] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, vol. 2. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.
[2] A. Morgenshtein, I. Shwartz, and A. Fish, "Gate diffusion input (GDI) logic in standard CMOS nanoscale process," in *Proc. IEEE 26th Conv. Elect. Electron. Eng. Israel (IEEEI)*, Nov. 2010, pp. 1–5.
[3] R. H. Krambeck, C. M. Lee, and H.-F. S. Law, "High-speed compact circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. 17, no. 3, pp. 614–619, Jun. 1982.
[4] T. Williams, "Dynamic logic: Clocked and asynchronous," in *Proc. IEEE Int. Solid State Circuits Conf. Tuts.*, Feb. 1996, pp. 1–24.
[5] N. F. Goncalves and H. J. De Man, "NORA: A racefree dynamic CMOS technique for pipelined logic structures," *IEEE J. Solid-State Circuits*, vol. 18, no. 3, pp. 261–266, Jun. 1983.
[6] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Heidelberg, Germany: Springer, 2006.
[7] M.-B. Lin, *Introduction to VLSI Systems: A Logic, Circuit, and System Perspective*. Boca Raton, FL, USA: CRC Press, 2011.
[8] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Boston, MA, USA: Pearson Education, 2011.
[9] A. Pal and A. Mukherjee, "Synthesis of two-level dynamic CMOS circuits," in *Proc. IEEE Comput. Soc. Workshop VLSI*, Apr. 1999, pp. 82–92.
[10] G. Yee and C. Sechen, "Dynamic logic synthesis," in *Proc. IEEE Custom Integr. Circuits Conf.*, May 1997, pp. 345–348.
[11] M. Zhao and S. S. Sapatnekar, "Technology mapping for domino logic," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 1998, pp. 248–251.
[12] T. J. Thorp, G. S. Yee, and C. M. Sechen, "Design and synthesis of dynamic circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 141–149, Feb. 2003.
[13] R. Hossain, *High Performance ASIC Design*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[14] I. Levi, O. Bass, A. Kaizerman, A. Belenky, and A. Fish, "High speed dual mode logic carry look ahead adder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 3037–3040.
[15] I. Levi, A. Kaizerman, and A. Fish, "Low voltage dual mode logic: Model analysis and parameter extraction," *Microelectron. J.*, vol. 44, no. 6, pp. 553–560, 2013.
[16] I. Levi and A. Fish, "Dual mode logic—Design for energy efficiency and high performance," *IEEE Access*, vol. 1, pp. 258–265, May 2013.
[17] I. Levi, A. Albeck, A. Fish, and S. Wimer, "A low energy and high performance DM^2 adder," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 11, pp. 3175–3183, Nov. 2014.
[18] I. Levi, A. Belenky, and A. Fish, "Logical effort for CMOS-based dual mode logic gates," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1042–1053, May 2014.
[19] J. Rabaey, *Low Power Design Essentials*. New York, NY, USA: Springer, 2009.
[20] G. Chartrand, *Introduction to Graph Theory*. New York, NY, USA: McGraw-Hill, 2006.
[21] Y. Crama and P. L. Hammer, *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2011.
[22] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Coping with the variability of combinational logic delays," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Processors (ICCD)*, Oct. 2004, pp. 505–508.
[23] *Liberty User Guides and Reference Manual*, Synopsys, Mountain View, CA, USA, 2007.
[24] D. Harris and M. A. Horowitz, "Skew-tolerant domino circuits," *IEEE J. Solid-State Circuits*, vol. 32, no. 11, pp. 1702–1711, Nov. 1997.
[25] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs*. New York, NY, USA: Springer, 2009.

VIACHESLAV YUZHANINOV received the B.Sc. degree in electrical engineering from Ben-Gurion University, Israel, in 2012, and the M.Sc. degree in electrical engineering from Bar-Ilan University, Israel, in 2015. He was a Research Assistant with the Low Power Circuits and Systems Laboratory, VLSI Systems Center, Ben-Gurion University, from 2011 to 2012. Since 2014, he has served as a Design Engineer of the SoC team with the Emerging Nano-Scaled Intergraded Circuits and Systems Labs, Bar-Ilan University. His research interests include efficient logic families and digital design automation.

ITAMAR LEVI received the B.Sc. and M.Sc. degrees in electrical and computer engineering as a part of a direct excellence student track from Ben-Gurion University, in 2012 and 2013, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with Bar-Ilan University. His current research interests are digital circuit design, hardware security, and cryptography.

ALEXANDER FISH received the Ph.D. (*summa cum laude*) degree in electro-optics from Ben-Gurion University, in 2006. He is currently an Associate Professor with the Faculty of Engineering, Bar-Ilan University, and the Head of the Nano-Electronics Track. In addition, he established the new Emerging Nano-Scaled Integrated Circuits and Systems Labs. He has authored over 100 scientific papers in journals and conferences. His research interests include the development of secured hardware, ultralow power embedded memory arrays, and high speed and energy efficient design techniques. He is a member of the Sensory, VLSI Systems and Applications and Bio-Medical Systems Technical Committees of the IEEE Circuits and Systems Society. He serves as the Editor-in-Chief of the *MDPI Journal of Low Power Electronics and Applications* and an Associate Editor of the IEEE Sensors, the IEEE Access, *Microelectronics* (Elseiver), and *Integration, the VLSI Journal*.