Logic Synthesis of Pass-Gate Logic Circuits with Emerging Ambipolar Technologies

Logic Synthesis of Pass-Gate Logic Circuits with Emerging Ambipolar Technologies

Valerio Tenace, *Member, IEEE,* Andrea Calimera, *Member, IEEE,*
Enrico Macii, *Fellow, IEEE,* Massimo Poncino, *Fellow, IEEE,*

***Abstract*—Emerging devices and new ultra-scaled silicon transistors have shown disruptive electrical and functional properties that might bring digital hardware to the next level. The key issue today concerns their integration. Even though the classical complementary logic style is the most intuitive option, other strategies such as pass-transistors that were discarded in the past because they did not fit Silicon MOS-FETs logic should be reconsidered. Obviously, the assessment of such alternatives requires customized CAD tools and optimization engines. The objective of this work is to introduce a synthesis and optimization flow for pass-gate logic circuits mapped onto emerging ambipolar technologies. As main contributions we propose: (*i*) a novel EXNOR-based decomposition technique that fully exploits don't care conditions to generate compact logic function representations; (*ii*) a dedicated one-pass synthesis flow where optimization and technology mapping are concurrently run on a common data structure, the Reduced Ordered Pass-Diagram. Experimental results demonstrate that the proposed flow outperforms existing synthesis tools by achieving more compact circuit representations with 8.5$\times$ less devices and about 8$\times$ shallower structures (on average), while still yielding lower CPU times.**

I. Introduction

A. Research Context

With the end of the Moore's law and the emerging of new computing paradigms, the ICT industry is called to face a big challenge, i.e., to re-think hardware design from devices to systems. At the lower levels of the design stack, this effort translates into a search for technologies that can outperform Silicon CMOS. In recent years, there has been significant growth of new devices, such as Silicon Nanowires [1], Magnetic Tunnel Junctions [2], Domain-Wall Nanowires [3], Graphene Nanoribbons [4], and Graphene p-n Junction devices [5]. Apart from their astounding electro-mechanical properties, these devices implement new logic primitives that might enable the design of digital circuits with intriguing characteristics. At this preliminary stage it is hard to predict which one will reach massive production and therefore speculating on their large-scale integration is even more risky. We embrace the idea that, as happened for semiconductors, CAD tools will play a key role during the selection process [6]. It is therefore essential to enable the possibility of a fast design exploration for the assessment of different implementation strategies. This work points in this direction as it introduces a novel framework for the logic synthesis of *pass-gate logic circuits mapped onto ambipolar technologies*.

B. Motivations

Static CMOS has become a *de-facto* standard for VLSI circuits implementation due to its many desirable properties: resilience to transistor scaling and supply voltage lowering, large noise margins, low static power consumption. Unfortunately, as transistors approached sub-nanometer lengths, most of these benefits have progressively blurred [7], [8]. With many new emerging devices experimented by process engineers, CMOS may possibly lose its supremacy.

New devices, however, also bring along new implementation styles; some of those that were discarded in the past as not suited for CMOS have now become a viable alternative. This is the case of pass-gate logic families, like the well-known Pass-Transistor Logic (PTL) [9], which already proved their efficiency for nanoelectromechanical switches (NEMs) [10], carbon nanotubes (CNTs) [11], and graphene p-n junctions [12], [13]. A proper assessment of pass-gate logic circuits requires design tools able to cope with the complexity of modern designs. For pass-gate logic the state-of-the-art is still represented by models and algorithms that rely on Binary Decision Diagrams (BDDs) [15]. In spite of the many values of BDDs, BDD-based logic synthesis shows several limitations which lead to poor design quality. As we will show in this work, there are many margins of improvement.

C. Contributions

This work introduces a dedicated design flow for the logic synthesis and optimization of ultra-low power pass-gate logic circuits, an alternative logic style suitable for emerging ambipolar devices that naturally implement binate logic primitives, EXOR/EXNOR in particular. The main contributions are as follows:

1) generalize the idea of EXNOR-based pass-logic, originally proposed for graphene devices in [16], to other emerging ambipolar technologies;
2) improve the design flow by means of a new logic synthesis engine built upon *(i)* a new data-structure called *Reduced Ordered Pass-Diagram*, and *(ii)* an efficient optimizer that exploits ad-hoc algorithms for redundancy removal and optimal variable ordering.

The proposed one-pass logic synthesis flow enables an efficient EXOR/EXNOR decomposition that matches the circuit topology of the pass-gate logic. The obtained circuits achieve remarkable area savings with respect to those implemented with tree-based logic synthesis flows commonly adopted for PTL designs, e.g. [17].

It is worth emphasizing that the objective is not to demonstrate the superior performance of pass-gate logic style against CMOS, nor that of directly comparing emerging technologies against silicon. Other previous works served this purpose, both for silicon [14], and beyond-silicon devices [10], [11], [12],

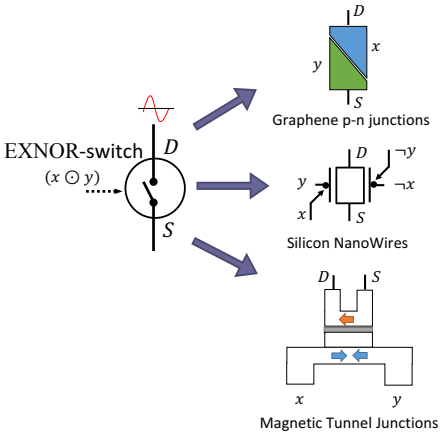

Fig. 1. Example of emerging devices and their logic equivalence with an EXNOR-switch node.

[13]. The goal of this work is to enable an efficient synthesis framework for the pass-gate logic style.

Since an exhaustive validation over all the possible ambipolar technologies is unfeasible, we restrict the choice to two relevant case studies, i.e., Silicon Nanowires [1] (SiNW) and graphene p-n junctions [5]. These represent the extreme corners of a wide spectrum of emerging devices that reflect the same logic abstraction. SiNWs, which are still built on conventional semiconductors, are the ultimate link between silicon and beyond-silicon circuits; Graphene p-n junctions, built on a futuristic 2-D flexible substrate, represent the cutting line. Both show a common property, that is, *ambipolarity*, thereby enabling an efficient implementation of binate logic primitives. As pictorially illustrated in Figure 1, a digital switch controlled through the built-in EXOR/EXNOR function is efficiently implemented; the same applies for other technologies, e.g., Magnetic Tunnel Junctions [2], Carbon NanoTubes [18], Graphene NanoRibbons [4].

II. Technology Review

A. Graphene p-n Junctions

The graphene p-n junction (Figure 2-a) has been introduced and discussed in several works, e.g. [19], [20]. It consists of a graphene sheet with two metal-to-graphene contacts, A and Z, which serve as signal input and output respectively, and a thick oxide layer that isolates the two back-gates, S and U, from graphene.

Exploiting the electrostatic doping, voltage potentials on terminals S and U work as a control knob to tune the Fermi energy (E_F) of the graphene sheet [21]; a positive voltage shifts down E_F in the valence band thereby leading to a p-type doping of the graphene region on top of the gate, whereas a negative voltage shifts E_F up in the conductance band leading to n-type doping. When opposite voltages are concurrently applied on S and U, i.e., $V(S)$=$\pm V$ and $V(U)$=$\mp V$, the device implements the p-n junction. As described in [21], under such configuration the transmission of carriers from the p-region towards the n-region follows the probability $T(\theta) = \cos^2(\theta)\mathrm{e}^{-\pi k_F D \sin^2(\theta)}$; θ is the angle between the electron's wave vector k_F and the normal of the junction (45° as imposed by the triangular shape of the back-gates, Figure 2-c), and D is the width of the metal gap between the back-gates, assumed to be $18nm$ [22]. It is worth noticing that $T(\theta)$=1 (i.e. 100% of carrier transmitted) when voltages applied at the back-gates are concordant, i.e., $V(S)$=$V(U)$ (n-n doping when $V(U)$=$-V$, p-p doping $V(U)$=V).

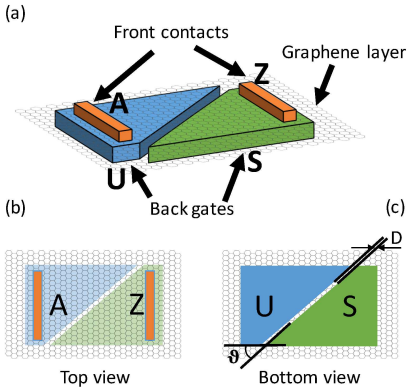

Fig. 2. Graphene p-n junction, (a) 3-D view, (b) top view, (c) bottom view.

Figure 3 depicts the electrical model of the p-n junction. The two resistors R_C connected to pins A and Z are the parasitic resistance of metal-to-graphene contacts. The resistor R_{AZ} represents the A-to-Z resistive path across the layer of graphene; its analytical expression is $R_{AZ} = \frac{R_0}{N_{ch}T(\theta)}$, where $R_0 = \frac{h}{4q^2}$ is the quantum resistance per propagating mode, N_{ch} is the number of excited propagating modes [21], and $T(\theta)$ is the transmission probability. As reported in [22], values of R_{AZ} ranges from $R_{ON} = 300\Omega$, (under n-n or p-p configuration), to $R_{OFF} = 10^7\Omega$ (p-n or n-p configuration). The model also integrates the coupling capacitance C_C between the two adjacent regions; the two lumped capacitors connected to the back-gates S and U, namely, C_{gS} and C_{gU}, consist of the series of the oxide capacitance and the quantum capacitance of the graphene sheet[1].

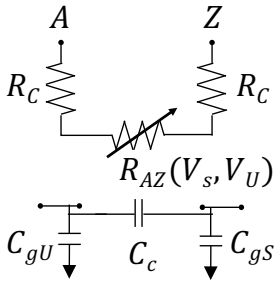

Fig. 3. Graphene p-n junction electrical model.

The electrical model of the p-n junction was implemented in Verilog-A and validated with accurate Spice simulations. Figure 4 shows the variation of the junction resistance as function of the voltage at the back-gates. The plot shows two curves: the first one (square markers) describes the resistance when the back-gate U is polarized through a positive voltage $+V_{dd}/2$; whereas the second one (circle markers) represents

[1]For an exhaustive discussion on the p-n junction and its electrical model, interested readers can refer to [22].

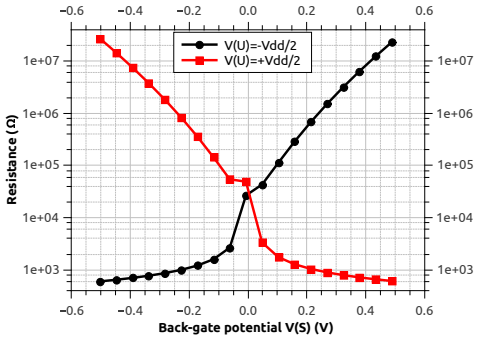

Fig. 4. P-N junction resistance variation vs. back-gate potential

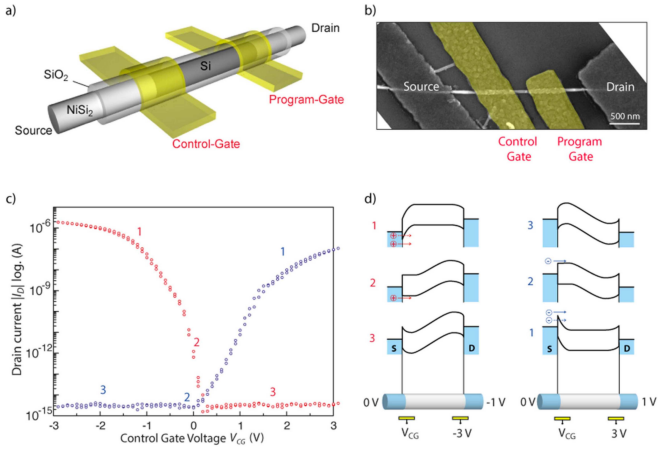

Fig. 5. Reconfigurable SiNW transistor. Schematic view (a), lithographic images (b), transfer characteristics for p-FET, red marker, and n-FET, blue marker (c) and band diagrams (d). Image credits due to [23].

the resistance when the back-gate U is polarized through a negative voltage $-V_{dd}/2$. Let us consider the positively-polarized curve (square markers) when S is driven by a voltage ranging from $-V_{dd}/2$ to $+V_{dd}/2$; the junction resistance is high at $V(S)$=$-V_{dd}/2$, while it gets smaller when $V(S)$ approaches $+V_{dd}/2$, few hundred ohms. It is therefore clear that the p-n junction is ON (low resistance) when the two back-gates have same polarity.

B. Silicon NanoWires

Reconfigurable SiNWs FET transistors (RFET), schematically represented in Figure 5-a and depicted in Figure 5-b, represent another class of dual-gate polarity controlled devices. The two gates are used to control charge carrier injection over Schottky barriers. The first one, called program gate (PG), is in charge of suppressing the injection of one type of charge carrier; the control gate (CG) modulates the injection of the other type of charge carrier [23].

The transfer characteristic (Figure 5-c) and the band diagrams (Figure 5-d) show that the p-type region is achieved by driving the programmable gate with $V_{PG} = -3V$ and $V_{DS} = -1V$. With such configuration the obtained barrier at the drain side prevents the injection of electrons (states 1 to 3) [23]. As the V_{CG} potential increases, the bending reduces until it reaches a flat-band (state 2). When V_{GC} is high enough, the energy barrier prevents injection of holes. Tuning the polarity of V_{DS} and V_{PG} prevents the injection of holes, thus configuring the RFET to an n-type region.

RFETs can be implemented with low band-gap semiconductor materials. Small threshold voltages and high drive currents, in both p- and n-type configurations, leads to enhanced device characteristics: (i) lower transfer curve in the Schottky region with respect to conventional FETs; (ii) low OFF currents; (iii) high ON-OFF ratio due to drive current limited by tunneling through the Schottky barrier [23], [24].

III. EXNOR-Based Pass-Gate Logic

A. Pass-EXNOR Gate (PXG)

As depicted in Figure 6, a PXG is a transmission gate enhanced with a built-in EXNOR switching function. It has two *logic pins* which are fed by the input logic signals (x and y) and two *transmission pins* that work as source (S) and drain (D) of an evaluation signal, e.g., a ramp or a sinusoidal signal. The switching function evaluates the Boolean EXNOR between the logic inputs, i.e. $x \bar{\oplus} y$; when TRUE, the PXG is ON (low resistance path from S to D); when FALSE, the PXG is OFF (high-impedance path from S to D). Different embodiments of the PXG are possible depending on the technology. Figure 6 reports those for graphene p-n junctions and SiNW RFETs, the target of this work. The SiNW implementation is borrowed by previous works [18], [17], which make use of two parallel devices in order to account for the threshold-voltage effect. This is not required for graphene p-n junctions, as they behave as pure passive resistors. To notice that a PXG shows higher expressive power if compared to CMOS gates (one device for graphene and two for SiNW with respect to twelve of a CMOS gate).

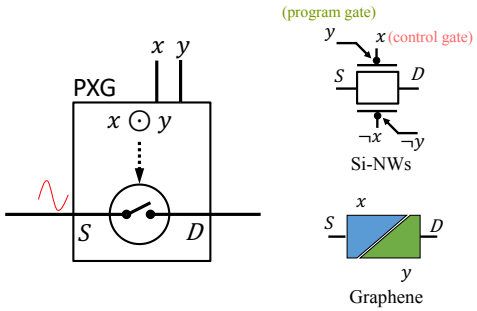

Fig. 6. PXG: abstract view and circuit implementation with SiNWs and Graphene.

The waveforms collected through the electrical Spice simulation of a graphene PXG are shown in Figure 7. The input pulse injected into S passes through the PXG and reaches D whenever x=y.

B. Unate Boolean Functions using PXGs

The EXNOR operator (hereafter indicated as $\bar{\oplus}$ or $\odot$) is not *functionally complete per sé*, and other unate logic primitives are needed to describe all possible logic functions. Figure 8 shows the network topologies that implement the *AND/OR* primitives, the logical *identity* and the *complement*.

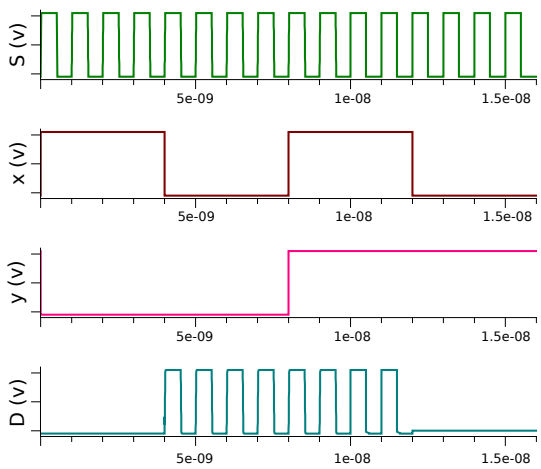

Fig. 7. Functionality of the Pass-EXNOR logic gate.

Series connections of PXGs implement the logic AND (symbol $\wedge$). In the example reported in Figure 8 (topmost left), the input pulse propagates to the output *iff* both PXGs are ON, namely, when $(x_1 \bar{\oplus} x_2) \wedge (x_3 \bar{\oplus} x_4)$. Parallel connections of PXGs, instead, implement the logic OR ($\vee$). In the example reported in Figure 8 (right), the input pulse propagates to the output when *at least one* of the two parallel PXGs is ON, namely, when $(x_1 \bar{\oplus} x_2) \vee (x_3 \bar{\oplus} x_4)$. Finally, connecting one of the control gate to '1' or '0' implements the logical *identity* or the *complement*. In the examples reported in Figure 8 (bottom left), the input pulse propagates toward the output when $x_1 = 1$ if the back-gate is fed with '1', when $x_1 = 0$ if the back-gate is fed with '0'.

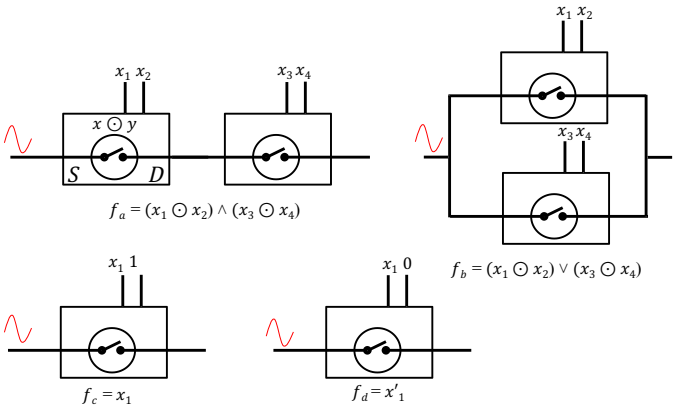

Fig. 8. Unate Boolean function using Pass-EXNOR gates.

C. Pass-XNOR Logic (PXL)

A PXL circuit [13] consists of a network of PXGs, Figure 9. Series connections of PXGs form a logic path; logic paths are connected in parallel from the root of the circuit (fed by a clocked-power signal) to the sink (the main output). The clocked-power signal works as an evaluation signal that eventually reaches the output when a logic path is ON; in this case the logic function is evaluated as TRUE, i.e., 1-logic. When all the logic paths are OFF, the propagation is inhibited and the logic function is evaluated as FALSE, i.e., 0-logic[2].

[2]More details on the integration of PXL networks can be found in [13]

An efficient mapping of generic Boolean functions onto PXL networks requires *ad-hoc* tools for abstract modeling, proper EXNOR decomposition, and efficient optimization. These issues are discussed in the next section.

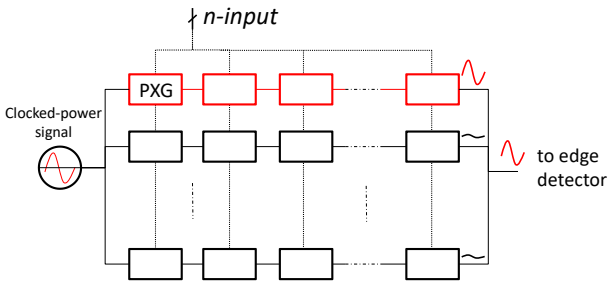

Fig. 9. Abstract view of a PXL network.

D. Quasi-Adiabatic Computing through PXL

The dynamic power consumption of a PXL circuit is the sum of two main contributions: (i) P_{in}, which is the power consumed by primary inputs when charging/discharging the input capacitance of the PXGs' control gate; (ii) P_{eval}, which is the power consumed due to the propagation of the evaluation signal through the PXGs. The former term (P_{in}) is similar to the input power consumed by CMOS gates; the latter (P_{eval}) follows the adiabatic charging principle.

During the evaluation-phase, the PXL network reduces to an equivalent lumped resistor R_{eq} [25], [26]. The R_{eq} is calculated as series/parallel connections of the in-to-out resistance of the PXGs; each of them can assume the value R_{ON} or R_{OFF} depending on the input pattern. Without loss of generality, let's consider as an evaluation signal an ideal ramp having a rise/fall-time T_{rf}. The power consumed across R_{eq} can be calculated as $P_{eval}(t) = R_{eq} i^2_{C_l}(t)$, with i_{C_l} the current injected into C_l, the output load capacitance driven by the PXL network. Spice-level simulations can be used to estimate R_{eq} and $i_{C_l}(t)$ under different input patterns.

The energy dissipated across R_{eq} is thereby obtained by integrating the evaluation power P_{eval} over T_{rf}:

$$E = \int_0^{T_{rf}} R_{eq} i^2_{C_l}(t) dt = \int_0^{T_{rf}} R_{eq} \frac{C_l^2 V_{dd}^2}{T_{fr}^2} dt = \frac{R_{eq} C_l}{T_{rf}} C_l V_{dd}^2 \quad (1)$$

The larger the T_{rf}, the smaller the energy; T_{rf}=$2R_{eq}C_l$ is the break-even point at which the consumed energy equals that obtained by using an ideal step as the evaluation signal. With $T_{rf} \rightarrow \infty$ the consumed energy approaches zero, that is the adiabatic charging.

While a detailed discussion on the energy savings brought by adiabatic computing on PXL can be found in [12], [13], what is paramount to note here is that adiabaticity is a peculiar property of the pass-gate logic topology. Indeed, in a PXL circuit the clocked-power is forced to flow through resistive paths (S-to-D paths across PXGs) never bumping into metal-oxide gate connections; the same is not for CMOS or other static logic families. As it will be discussed in Section IV, the logic synthesis of PXL networks has to meet specific constraints to preserve this feature.

E. Signal integrity in PXL circuits

Since PXL circuits are resistive nets with negative gain, complex topologies might suffer from signal integrity. For instance, deep chains of cascading gates may induce a non-negligible voltage degradation; similarly, a PXG with a too high fan-out won't be able to propagate the evaluation signal. To provide a clearer picture on these effects, we describe a parametric analysis conducted on a graphene-based PXL circuit. For what concerns voltage degradation, we set up the following simulation:

- the evaluation signal (i.e. the clocked power) is a square wave with a period of 1ns and amplitude of 1.1V;
- the evaluation signal is fed as input to a CMOS buffer which serves as the driver cell; the buffer, taken from a commercial 45nm technology library, is minimum-sized;
- the output load capacitance of the PXG chain is set to 0.6fF, which is the equivalent input capacitance of a minimum sized CMOS buffer gate (the same kind of buffer used as driver);
- all the PXGs in the chain are configured as transparent gates, i.e., both input signals at the back-gates have the same polarity (refer to the highlighted path in Figure 9 for a pictorial representation);
- the output signal integrity is measured for different depths of the PXG chain.

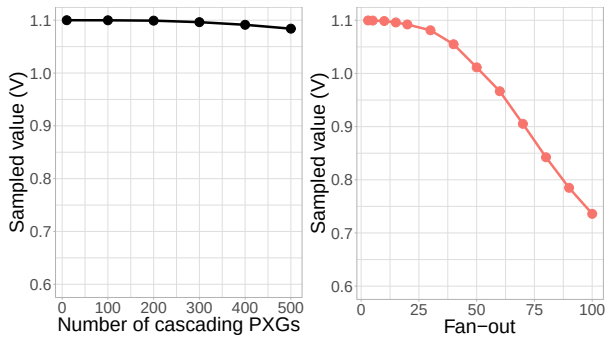

Fig. 10. Signal integrity analysis. Cascading gates (left), and fan-out (right) effects in PXL circuits.

Figure 10 (left plot) reports the obtained results. Numbers suggest that even very deep chains (e.g. 500) guarantee an effective propagation of the evaluation signal.
For what concerns the assessment of fan-out capabilities, the topology of the simulated PXL circuit is the following:

- a single transparent PXG driven by a minimum size 45nm CMOS buffer, we refer to the PXG as the root PXG;
- the root PXG drives other parallel branches, each one made up of a single PXG (the number of branches B defines the fan-out).
- B ranges from 2 up to 100.

Two different experiments were conducted.
Analysis 1: only one branch over B allows the propagation of the evaluation signal. Collected results show that the sampled output signal is about 1.099V for all configurations, namely signal integrity is preserved.
Analysis 2: all the branches B are active at the same time. This represents a worst-case scenario that arises whenever a branch of a PXL circuit is shared among different output cones. Obtained results are reported in Figure 10 (right plot). Signal integrity is retained up to fan-out 50 (degradation $< 10\% \cdot V_{dd}$); a larger fan-out (e.g. 100 active gates) introduces a non negligible signal degradation (33%). To overcome this issue, a simple and effective solution could be introducing an upper limit on the maximum number of fan-out PXGs; whenever such a limit is reached, branches are merged in different clusters. Thereby, signal integrity would be preserved with a minimum impact on circuit size. It is worth to notice that the same countermeasure is also adopted for BDD-based synthesis of PTL circuits [14].

IV. Previous Work on EX(N)OR-based Expansion

During the last decades, research on optimal logic functions representation and simplification using binate operators, EXOR in particular, has been extensively investigated as an answer to the growing complexity of logic circuits. Nevertheless, such techniques didn't find much room in commercial applications. The reason is that EXOR CMOS gates show a number of transistors much larger than AND-OR gates. Things are changing with the growth of emerging ambipolar devices which may revive those methods.
In 1990, a pioneering work by Sasao et al. [27] introduced an algorithm for the expansion of logic functions using EXOR-AND products, i.e., Exclusive-Sum of Products (ESOPs). The same was later improved by [28] which introduced a more complex method that makes use of EXOR-AND-OR products, i.e., Exclusive-Sum of two Sum-of-Products. As reported in [28], area savings with respect to classical Sum-Of-Product (SOP) representations are in the order of 40%.
Unfortunately, such methods (and their extensions) do not fit PXL circuits, since all of them look for a possible insertion of EXOR operators between SOP clauses. However, a PXL network is in the form of *Sum-of-Products of Exclusive-Sums*, where the inner part, i.e., the first primitive used during decomposition, is the EXNOR, and not the AND-OR like in [27], [28]. In other words, in a PXL circuit the constraint is that every EXNOR should work on the primary inputs. Preserving such topological characteristic is paramount to (i) exploit the expressive power of the PXGs and (ii) keep adiabaticity alive.
As an example, let's consider the Boolean function $f{=}x_1x_2\bar{\oplus}x_3x_4$, which is expressed in the form of Exclusive-Sum of Products (ESOP) [29]. The corresponding PXL implementation would be the one reported in Fig. 11.

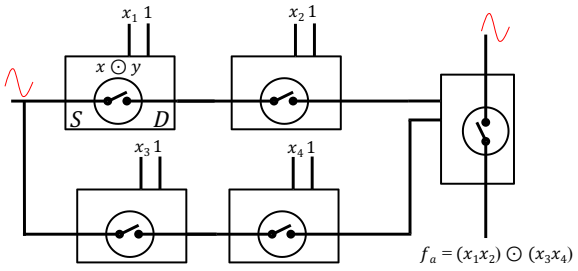

Fig. 11. Example of *intra-layer* connections

As one can observe, the expressive power of PXGs is underutilized as all of them have one logic input stacked at a fixed value. Moreover, the circuit shows a break on the resistive path across the PXGs; this topology prevents the adiabatic charging of the output capacitance. As a consequence, power consumption increases. Moreover, fabricating intra-layer connections on monolithic graphene sheets may result dramatically complex, hence, too costly.

Only recently, the work by Bernasconi et al. [30] demonstrated a Boolean decomposition in the form of *Sum-of-Products of Exclusive-Sums* that works as follows. Given a generic function $f(x_1, x_2, ..., x_N)$, where $S = (x_1, x_2, ..., x_N)$ is the support-set of cardinality N, f can be expanded according to the $(x_i - p)$ paradigm as shown below:

$$f(x_1, x_2, ..., x_N) = (x_1 \bar{\oplus} x_2) f|_{x_1 = x_2} \vee (\bar{x_1} \bar{\oplus} x_2) f|_{x_1 \neq x_2} \quad (2)$$

Unlike a Shannon's decomposition that considers the expansion with respect to a single variable x_i forced to be '0' or '1', here the co-factoring is obtained imposing the equality between two variables x_i and x_j, i.e. $x_i = x_j$ or $x_1 \neq x_2$; each co-factor is then AND-ed with the EXOR/EXNOR of the two variables. Equation (2) can be recursively applied until the function reduces to one variable. In this case, the $(x_i - p)$ decomposition applies the equality $x_i = 1$ or $x_i = 0$, which is the Shannon's decomposition. As per the Shannon's expansion, also the $(x_i - p)$ expansion is ruled by a pre-fixed *global* variable sequence (G-VS) used during the co-factoring process.

The above theory inspired a new abstract model for logic representation, i.e., the Bi-conditional Binary Decision Diagrams (BBDD) [17], a tree-based structure that extends BDDs [15] by using nodes with two EXOR-ed control variables. The following example gives a synthetic description of BBDDs.

Example IV.1. Given the Boolean function $f(S)$, with $S = (x_1, x_2, x_3)$, described as:

$$f(S) = (x_1 \wedge x_2) \vee (\bar{x_2} \wedge x_3) \vee (x_2 \wedge \bar{x_3}). \quad (3)$$

Assuming a global variable sequence G-VS=(x_1, x_2, x_3,1), the $(x_i - p)$ representation of f is given by (4).

$$\begin{aligned} f(S)^{(x_i - p)} = &(x_1 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} x_3) \wedge x_3 \\ &\vee (x_1 \bar{\oplus} x_2) \wedge (\bar{x_2} \bar{\oplus} x_3) \\ &\vee (\bar{x_1} \bar{\oplus} x_2) \wedge (\bar{x_2} \bar{\oplus} x_3) \end{aligned} \quad (4)$$

Figure 12 shows the BBDD representation of f after the $(x_i - p)$-expansion. The graph is composed of a root node through which the evaluation signal is injected into the network, two terminal nodes indicating the value assumed by the function, and four internal nodes that work as switches. More in detail, as depicted in the enlargement, each internal node evaluates the EXNOR-switch on the two primary inputs associated to the node: if true, i.e. signals have the same value, the left branch (solid line in the graph) is activated, the right branch (dotted line) is active otherwise. Like classical BDDs, the selected global variable sequence has a direct effect on the BBDD structure and its complexity. Indeed, each level is associated to two adjacent primary inputs in the G-VS

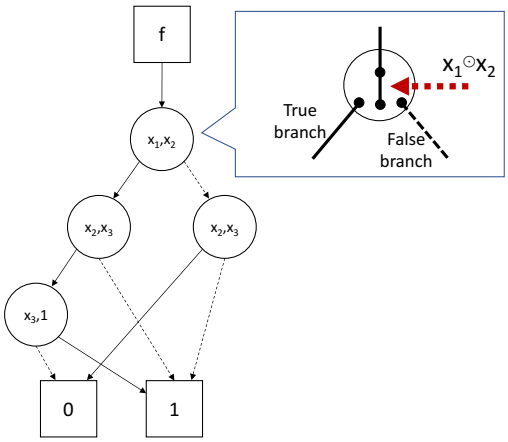

Fig. 12. Biconditional BDD representation of (4).

set, and different sequences do result into different branch organizations.

It is worth emphasizing that BBDDs could be adapted to represent any PXL circuit implemented as shown in Figure 9. In such a case, each branch ending on the logic '0' sink would be omitted thus to save one device in the resulting circuit, e.g. the false branch of node $(x_3, 1)$ in Figure 12. However, as it will be shown later in the simulation section, the new abstract model proposed in this work fits better with PXL circuits and it outperforms BBDDs when considering incompletely-specified Boolean functions.

V. The PXL Expansion

Let us consider the Boolean function $F(S)$ in Example IV.1. By following the Boolean decomposition principle, f can be rewritten as in (5), being $f_1(S_1) = x_1 \wedge x_2$, $f_2(S_2) = \bar{x_2} \wedge x_3$, $f_3(S_3) = x_2 \wedge \bar{x_3}$, and $S_1, S_2, S_3 \in S$ their support-set.

$$F(S) = f_1(S_1) \vee f_2(S_2) \vee f_3(S_3) \quad (5)$$

Each f_k function is a chain of products between primary inputs that can be manipulated with the Boolean rule described in (6), where n represents the number of literals in the support set S_k for each f_k. The result is an EXNOR-expanded function.

$$\begin{aligned} f_k(S_k) =& x_1 \wedge x_2 \wedge x_3 \cdots \wedge x_n \\ =& (x_1 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} x_3) \wedge \ldots (x_{n-1} \bar{\oplus} x_n) \wedge x_n \end{aligned} \quad (6)$$

It is thereby possible to convert each f_k as in (7).

$$\begin{aligned} f_1(S_1) &= (x_1 \bar{\oplus} x_2) \wedge x_2 \\ f_2(S_2) &= (\bar{x_2} \bar{\oplus} x_3) \wedge x_3 \\ f_3(S_3) &= (x_2 \bar{\oplus} \bar{x_3}) \wedge \bar{x_3} \end{aligned} \quad (7)$$

To be noted that each S_k in (7) retains the same subset of literals with respect to S. For instance, being $S_1 = \{x_1, x_2\}$, the sequence (x_1, x_2) represents the *local* variable sequence (L-VS) for decomposing f_1, where x_3 is discarded since it is not part of S_1. In other words, each product term described with (6) neglects, by construction, don't care variables. As a consequence, each f_k is always represented in the form of product-of-EXNOR over the minimum number of input literals. Also, the L-VS may change for each f_k.

The original function $F(S)$ is then obtained by substituting (7) in (5) thus to obtain:

$$F(S) = (x_1 \bar{\oplus} x_2) \wedge x_2 \vee (\bar{x_2} \bar{\oplus} x_3) \wedge x_3 \vee (\bar{x_2} \bar{\oplus} \bar{x_3}) \wedge \bar{x_3} \quad (8)$$

Equation (8) matches the PXL topology, namely, *Sum-of-Products of EXNOR.*

A more formal description of the PXL-expansion paradigm is given as follows: *A generic Boolean function $F(S)$ in the form of Sum-of-Products (SOP) is PXL-expanded by evaluating each single product term $f_p(x_{p1}, ..., x_{pN}, 1)$ independently; each f_p, defined over its support set $(x_{p1}, ..., x_{pN}) \in S$, is decomposed in the form of products-of-EXNOR by following a local variable sequence (L-VS$_{f_p}$) along which unspecified literals are dropped; $F(S)$ is finally reconstructed by OR-ing the products-of-EXNOR.*

VI. One-Pass PXL Synthesis

A PXL design can be synthesized using a *One-Pass Synthesis* (OPS) flow. In OPS both logic optimization and technology mapping are run over a common model that represents both the logic behavior and the physical structure of the circuit. This section describes an *ad-hoc* abstract model for PXL synthesis, the *Pass Diagram* (PD), along with the algorithms for its building and optimization. All routines are part of a software package written in the C++ programming language, where built-in data structures and standard libraries are exploited to achieve optimal performance.

A. Pass Diagrams (PDs)

Given a generic Boolean function $F(S)$, its corresponding PD is a rooted and directed acyclic graph defined as G=($\Phi \cup V \cup \Theta \cup A$). The internal vertexes $v_i \in V$ are labeled with a function $g(x, y)$, with $\{x, y\} \in S$ as the input literals and function g as the $EXNOR$. Each node has one outgoing edge $a \in A$ which identifies a logical AND between the predecessor and the successor node. Terminal nodes $\theta \in \Theta$ with out-degree 0 represent the output value assumed by F; whereas root nodes $\phi \in \Phi$ with in-degree 0 represent the input for the evaluation signal. Each root-to-sink path in a PD represents one product among EXNORs, i.e., the subfunctions f_k described in SectionV; parallel paths form the OR among subfunctions f_k. Figure 13-a depicts the PD of $F(S)$ in Equation 8. It is also worth noticing that the PD structure can be mapped directly in Pass-EXNOR logic by replacing each node with a dedicated PXG.

B. The PD build function

PDs are built using as starting point the implicant table of the function $F(S)$, also known as *PLA table* in the Espresso environment [31].

As an example, Table I collects the implicants of the logic function (3). The value of the input literals are listed in their respective columns, whereas the output of the logic function is reported in column F. For the sake of clarity, we also report the implicants of F(S) in 3 (first column).

Algorithm (1) reports the pseudo code of the *build* procedure. It takes as an input the PLA table T and the global variable

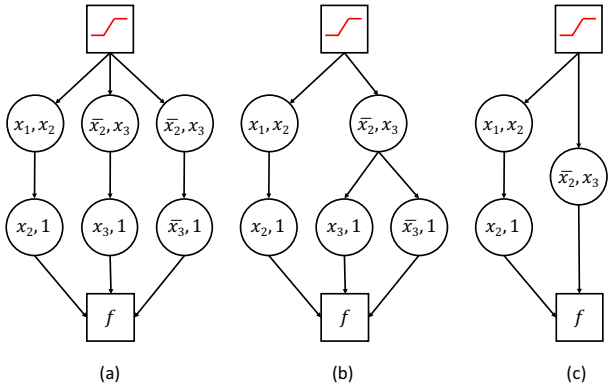

Fig. 13. Pass Diagram representation of the function in (8) before reduction rules (a), after Merge (b), and after Delete (c).

TABLE I
PLA table of (3)

	x_1	x_2	x_3	F
f_1	1	1	-	1
f_2	-	0	1	1
f_3	-	1	0	1

sequence G-VS. The latter, unless otherwise specified, is assumed to be the order of the input variables as they appear in T, left to right. At the end of the building process, the algorithm returns the final PD.

The algorithm generates the PD structure branch-wise, thus processes the table T row-by-row. For each row, it first generates the corresponding *local* variable sequence *L-VS*. Differently from the *G-VS* introduced earlier, the *L-VS* represents the sequence of primary inputs involved in the PXL expansion of each branch f_R of the PD structure. For instance, using Table I as T and a default global variable sequence G-VS= $\{x_1, x_2, x_3, 1\}$, the local variable sequence *L-VS* of the first row f_1 returned by `GenerateLVS` is *L-VS*= $\{x_1, x_2, 1\}$ as x_3 is a don't care.

Within each row, the algorithm does a comparison among pairs of adjacent literals in *L-VS*; if polarity is the same, i.e. 1-1 or 0-0, the operation between the two is an EXNOR, otherwise the first variable gets complemented. Resorting to the previous example, the first pair in f_1 is $\{x_1, x_2\}$ which results into the EXNOR between x_1 and x_2; the second pair is $\{x_2, 1\}$, hence, the EXNOR between x_2 and 1. For f_2, the first pair is $\{\bar{x_2}, x_3\}$ which results into the EXNOR between $\bar{x_2}$ and x_3.

Each EXNOR term forms a new node that is appended to the current path f_R. Once all literals in the *L-VS* are processed, the branch f_R is added to the PD. Figure 13-a shows the resulting PD for the function (8), which is composed of six nodes (six PXGs).

For a PLA table with N implicants and M literals, the complexity of the proposed *build* function is $O(N \cdot M)$.

C. Reduced Pass Diagrams

As proposed in the past for BDDs, several reduction rules based on structural matching can also be applied to PDs. What follows is the description of two of them tailored on the PD structure: *Merge* and *Delete*.

Algorithm 1: Pass Diagram build function

Input: PLA Table T, Global Variable Sequence *G-VS*
Output: Pass Diagram PD

```
PD ← ∅
foreach row R ∈ T do
    f_R ← ∅
    L − VS_R = GenerateLVS(R, G-VS)
    foreach pair of primary inputs (v_i, v_k) ∈ L − VS_R do
        if polarity(v_i) == polarity(v_{i+1}) then
            newGate ← EXNOR(v_i, v_k)
        else
            newGate ← EXNOR(v̄_i, v_k)
        end
        f_R ← AddGate(newGate)
    end
    PD ← AddPath(f_R)
end
```

Algorithm 2: Pass Diagram reduction algorithm

Input: Pass Diagram PD
Output: Reduced Pass Diagram RPD

```
foreach branch B ∈ PD do
    foreach branch G ∈ PD, where G ≠ B do
        if CheckSimilarGates(B,G) ≠ 0 then
            Q ← MergeGates(B, G)
            if CommonNodesAreTautology(Q) then
                Q ← DeleteCommonNodes(Q)
            end
            PD ← UpdateNetwork(Q)
        end
    end
end
RPD = PD
```

Rule #1 - Merge: *If a node $v_n \in V$ in a path f_i has the same parent nodes of a node $v_p \in V$ in a path f_j, the common nodes of f_i and f_j can be merged.*
For the PD shown in Figure 13-a, nodes $\{x_3, 1\}$ and $\{\bar{x_3}, 1\}$ on the two rightmost paths have same parent $\{\bar{x_2}, x_3\}$; the latter can be merged thus to achieve the reduced PD shown in Figure 13-b. This topological transformation results into the Boolean simplification described in (9).

$$\begin{aligned} f_2^{PXL} \vee f_3^{PXL} =& ((\bar{x_2} \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} 1)) \vee ((\bar{x_2} \bar{\oplus} x_3) \wedge (\bar{x_3} \bar{\oplus} 1)) \\ =& (\bar{x_2} \bar{\oplus} x_3) \wedge ((x_3 \bar{\oplus} 1) \vee (\bar{x_3} \bar{\oplus} 1)) \end{aligned} \tag{9}$$

This operation resembles an algebraic factorization and can be applied on multiple nodes/branches simultaneously.
Rule #2 - Delete: *Two or more nodes $(v_n, \ldots, v_p) \in V$ with the same parent node can be deleted if, and only if, they represent a tautology*.
This operation is often enabled by Rule #1. For instance, after the merge operation on the PD in Figure 13-b, it results that $(x_3 \bar{\oplus} 1) \vee (\bar{x_3} \bar{\oplus} 1)$ is a tautology; the PD is thereby simplified as depicted in Figure 13-c. To notice that the sequence of the two reduction rules brings to 50% cardinality reduction from the original PD.
Algorithm 2 gives the pseudo-code of the optimization loop that applies *Merge* and *Delete* for redundancy removal; it takes as an input the PD generated from the *build* function and returns a reduced PD structure (RPD).
For each branch of the original PD, the algorithm searches for other branches that share one, or more, common nodes. As soon as such common nodes are identified, selected paths B and G are merged according to Rule #1 (*Merge*). Thereafter, a structural tautology check is run over non-merged nodes belonging to the two branches B and G; if positive, the nodes gets pruned according to Rule #2 (*Delete*) and the structure updated. The final graph is then returned to the calling function.

D. Reduced and Ordered Pass Diagrams

Let us consider a Boolean function $g(x_1, x_2, x_3)$ whose implicant table is reported in Table II.

TABLE II
PLA TABLE OF (10).

	x_1	x_2	x_3	F
f_1	0	-	0	1
f_2	1	1	1	1
f_3	1	-	1	1

TABLE III
SORTED PLA TABLE OF (10).

	x_1	x_3	x_2	F
f_1	0	0	-	1
f_2	1	1	1	1
f_3	1	1	-	1

The *build* function returns the Boolean expression described in (10).

$$\begin{aligned} g = & (x_1 \bar{\oplus} x_3) \wedge (\bar{x_3} \bar{\oplus} 1) \\ & \vee (x_1 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} 1) \\ & \vee (x_1 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} 1) \end{aligned} \tag{10}$$

The reduced version after *Merge* and *Delete* (between the first and the last product terms) is described in (11).

$$\begin{aligned} g = & (x_1 \bar{\oplus} x_3) \\ & \vee (x_1 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} 1) \end{aligned} \tag{11}$$

Let's now start from a different, yet equivalent, table where columns x_2 and x_3 are swapped, Table III. The *build* function made run over the new table returns a different PD, hence, a different Boolean representation which is described in (12).

$$\begin{aligned} g = & (x_1 \bar{\oplus} x_3) \wedge (\bar{x_3} \bar{\oplus} 1) \\ & \vee (x_1 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} 1) \\ & \vee (x_1 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} 1) \end{aligned} \tag{12}$$

The *Merge* and *Delete* operations (applied on the first and last product terms) generates the expression in 13

$$\begin{aligned} g = & (x_1 \bar{\oplus} x_3) \\ & \vee (x_1 \bar{\oplus} x_3) \wedge (x_3 \bar{\oplus} x_2) \wedge (x_2 \bar{\oplus} 1) \end{aligned} \tag{13}$$

which can be further minimized by a second round of optimization using the same *Merge* and *Delete* (on the two rightmost terms). The resulting Boolean representation is given in (14)

$$g = (x_1 \bar{\oplus} x_3) \tag{14}$$

A comparison between (11) and (14) suggests the existence of an optimal variable sequence that minimizes the vertex-set cardinality of the PD (similar to BDDs). Finding such an optimum would require an exhaustive exploration of the search space. In order to address this issue, we introduce two heuristics that can be eventually chained in a two-stage flow.

Predictive sorting

An accurate analysis on PLA tables reveals the following rule of thumb: given two columns, the larger the number

Algorithm 3: Predictive sorting algorithm

Input: PLA Table T
Output: Global Variable Sequence G-VS
G-VS = $\emptyset$
foreach *column* $x_k \in T$ **do**
  $\#DC = 0$
  $S = 0$
  foreach *row* $r \in x_k$ **do**
    if r *is don't care* **then**
      $\#DC = \#DC + 1$
    end
  end
  $S_{x_k} = \frac{\#DC}{\#rows}$
end
G-VS = SortBySparsity(S)

of adjacent literals with a specified value, either 0 or 1, the higher the probability to find common nodes among parallel logic paths; common nodes open to *Merge* optimization, hence, more savings. In other words, the larger the number of EXNOR operations over the same literal pairs, the larger the probability to find nodes with a common parent. Ideally, all rows should be processed with the same *G-VS*; however, since don't care literals alter the *L-VS* (they are dropped during expansion), their presence is the main source of irregularity among logic paths.

The basic idea behind predictive sorting is to arrange the implicant table keeping in mind the observation on the position of don't cares. We thereby define a metric used for column sorting. That's the **column sparsity** (S_{x_k}), defined as follows: *Given a $N \times M$ implicant table, defined x_k as the k-th input literal, $S_{x_k} = \frac{\#DC_{x_k}}{N}$, with $\#DC_{x_k}$ the number of don't cares in the k-th column.*

Intuitively, to increase the effectiveness of a sorting algorithm, columns with similar sparsities should be clustered. In other words, low-sparsity columns, e.g., columns with $S_{x_k} \approx 0$, should be separated by high-sparsity columns ($S_{x_k} \approx 1$). The proposed sorting, called *Predictive sorting*, arranges columns with ascending sparsity order. For instance, in Table II, columns x_1 and x_3 have the same sparsity index $S_{x_1} = S_{x_3} = 0$, whereas $S_{x_2} = 2/3$; the default *G-VS*$= \{x_1, x_2, x_3\}$ becomes $\{x_1, x_3, x_2\}$, which, as shown in (14), gets to a smaller PD.

Algorithm 3 reports the pseudo-code of the implemented routine; it takes the PLA table T as an input and returns the S_{x_k}-ordered table.

It is also worth emphasizing that *Predictive sorting* does not guarantee an optimal solution, but it could be rather used as a lightweight strategy when CPU-time is the priority.

Genetic sorting

The *Predictive sorting* can't always get improvements with respect to the original *G-VS*. Indeed, the *Merge* and *Delete* rules are subject not only to the sparsity of the primary inputs, but also to the actual polarity of the input literals. Finding the optimal *G-VS* is therefore a complex task that can be processed using some meta-heuristic, e.g., genetic algorithms (GAs) [32], a well known class of optimization techniques emulating the rules of natural selection.

We propose the use of a genetic search to find the global variable sequence that minimizes the vertex-set cardinality of a PD generated with the proposed *build* procedure. The GA is defined through the following parameters:

- *genes*: the atomic object that describes an individual, in our case each input literal is a gene;
- *individuals*: possible solutions to the problem, in our case a given G-VS is an individual;
- *generation*: the set of individuals of a given population.

A GA repeatedly modifies a population of individual solutions. At each step it selects the best individuals from the current population and uses them as parents to generate children belonging to the next generation, i.e., an improved population. From generation to generation, individuals evolve to preserve the kind, namely, the algorithm evolves toward an optimal solution. An efficient GA requires proper evolution rules that bring to convergence; we rely on the following three criteria:

- *Crossover*: two selected individuals are split at a random gene and then crossed with each other. Figure 14-a depicts an abstract crossover operation between two individuals. It is important to underline that individuals with repeating genes are not allowed in our solution. Therefore, crossover is applied *iff* obtained individuals do not contain repeated items. Due to such limitation, crossover results to be the least effective rule for generating best individuals.
- *Mutation*: alter the value of a gene of the best individuals. In our formulation, a gene is altered by swapping two randomly selected genes, as depicted in Figure 14-b.
- *Inversion*: line the individual's gene footprint in reverse order (refer to Figure 14-c). This criteria is randomly applied to all genes of an individual, or just to a subset of those.

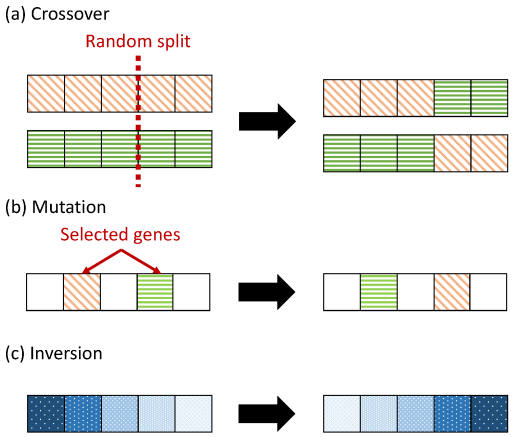

Fig. 14. Pictorial representation of crossover (a), mutation (b), and inversion (c) mechanisms adopted in the proposed genetic sorting algorithm. Different colors/textures represent genes that belong to the same individual, as in (a), or different genes of the same individual, as in (b) and (c).

Best individuals, those used as parents, are recognized by means of a min-cost function we described as $min(|D|)$, with D as the vertex-set cardinality of the resulting PD.

The pseudo code of the proposed GA is reported in Algorithm 4.

Additional details of the GA algorithms are given below. First generation: As a first step, a new population is generated;

Algorithm 4: Genetic sorting algorithm

Input: PLA Table T, Original Global Variable Sequence G-VS, Generations MAX_GEN, Individuals I_MAX, Survival rate Sr
Output: Optimal Global Variable Sequence OG-VS

```
G = InitGeneration(G-VS)
OG-VS = ∅
repeat
    Fitness = ∅
    foreach individual I ∈ G do
        Fitness[I] = Build(T, I)
    end
    BestIndividuals ← SelectBest(Fitness, Sr)
    G = NextGeneration(BestIndividuals)
    OG-VS = BestIndividuals[0]
until until generation < MAX_GEN;
```

given the maximum number of allowed individuals (I_MAX) the first generation is obtained by randomly scrambling the original *G-VS*. At the beginning, the G-VS can be assumed to be the variable order as it appears in the original PLA table description or, alternatively, the one that comes after the *Predictive sorting*.

Fitness function & best individuals: The PD structure for each individual of a given generation G is obtained with the `build` function (please refer to Algorithm 1). The fitness index, which describes the distance between the current individual and the best solution achieved so far, is represented by the total number of nodes of the PD. The `SelectBest` sub-routine takes those individuals with a fitness value close to the best one; the parameter Sr, also called *survival rate*, is assumed to be an upper bound to the fitness index, e.g., if $Sr = 0.1$ then only the best 10% individuals will be selected.

Local best & next generation: The best individual is selected to represent the optimal solution found in the current generation. The next generation is then obtained by applying crossover, mutation and inversion transformation till reaching I_MAX individuals.

The process stops when the maximum number of generations MAX_GEN is reached; here's where the optimal result *OG-VS* is returned.

VII. Simulation Results

A. Experimental Setup

In order to validate the proposed OPS flow and quantify the savings with respect to state-of-art synthesis methods, this section presents a comparison among three different design implementation frameworks:

PXL + Pass Diagram: The method presented in this paper. Each benchmark, originally described using the Espresso PLA format, is processed with the proposed OPS flow; the latter returns a Spice netlist where each node of the reduced & ordered PD is mapped as shown in Figure 15-(a).

Tree-of-MUXs (T-MUX) + BBDD: A T-MUX circuit can be seen as a pass-gate circuit with a tree-structure implemented by MUXes; as for the PXL, an evaluation signal flows through the PXG paths eventually reaching the output. Due to its topology, a T-MUX circuit is well modeled by a BBDD. Indeed, each node in the BBDD represents a MUX driven by an EXOR operator between two primary inputs. Such nodes are mapped with two p-n junctions or with four Si-NWs, as described

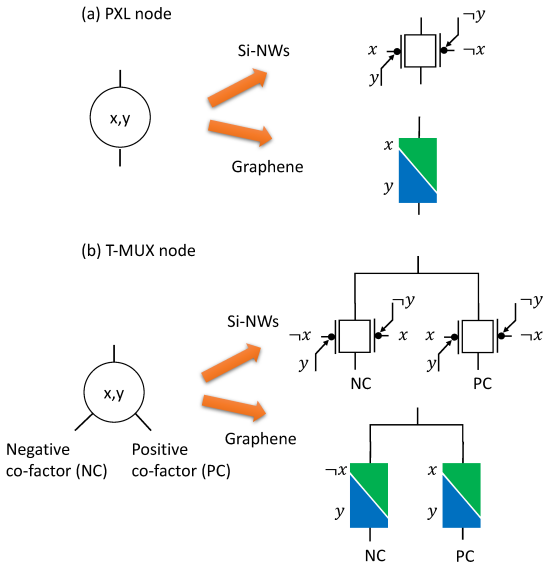

Fig. 15. Technology mapping for PXL and T-MUX implementations.

in Figure 15-(b). The adopted synthesis flow consists of a first stage where each benchmark is processed with the ABC tool [33]; the resulting Verilog description is then processed using the BBDD-package [34] in order to obtain a reduced & ordered BBDD structure with a single leaf node. Finally, a dedicated TCL script parses the BBDD and generates the Spice netlist. This flow represents the state-of-art for Pass-Transistor-Logic (PTL) circuits.

Multi-level (ML) + AIG: a standard multi-level design flow for CMOS gate libraries. The And-Inverter Graph (AIG) is the abstract model through which both open-source logic optimizers *ABC* and commercial synthesis tools operate. It is worth emphasizing that AIG-based flows cannot be used for pass-gate logic circuits; indeed, the output is a multi-level generic netlist for standard gates. Each standard cells is then "virtually" implemented with both graphene p-n junctions and SiNWs RFETs. We introduce AIGs solely as a benchmark to quantify how close to commercial CMOS tools the proposed synthesis flow is. Indeed, as discussed in the first section of the paper, implementing CMOS-like gates results to be quite inefficient.

Other packages for logic synthesis tailored on SiNW technologies are available. Most notably, the *MIGthy* synthesis tool relies on majority-inverter graphs (MIGs) mapping and optimizations strategies [35]. However, we decided to not include MIGthy circuits in our experimental comparison for two reasons: (i) the MIG data-structure is derived from classical AIG circuit representations, therefore they can be considered as a sub-class of ML circuits; (ii) each node in a MIG is represented by a 3-input majority gate, where inputs connections are not primary inputs but intermediate results computed in some previous logic levels. As discussed in Section V, methods for multi-level logic decomposition do not apply to PXL circuits.

The experiments have been run on a set of open-source benchmarks from the LGSynth91 suite [36]. Within each circuit, in-to-out and register-to-register combinational logic cones are

TABLE IV
SYNTHESIS RESULTS OBTAINED WITH THE PROPOSED TECHNIQUE. HIGHLIGHTED NUMBERS REPRESENT SELECTED RESULTS.

	PI	PO	PT	Build & Opt.		Predictive		Genetic	
				w/o opt.	w/ opt	Nodes	Savings (%)	Nodes	Savings (%)
misex1	8	7	32	122	68	**64**	5.88	56	17.65
o64	130	1	65	130	130	130	-	**130**	-
misex2	25	18	29	188	152	**151**	0.66	123	19.08
s298	17	20	70	250	187	≥187	-	**177**	5.35
s510	25	13	112	485	370	**274**	25.95	244	34.05
s820	23	24	127	854	666	**575**	13.66	545	18.17
s400	24	27	167	913	673	≥673	-	**574**	14.71
s1488_split	14	25	283	1634	1171	**1018**	13.07	883	24.59
s1494	14	25	283	1634	1191	**1046**	12.17	942	20.91
s953	45	52	235	1750	1190	≥1190	-	**1021**	14.20
apex3	54	50	642	4602	3352	**3112**	7.16	2582	22.97
table5	17	15	554	6372	4989	≥4989	-	**4070**	18.42
k2	45	45	936	7114	4053	≥4053	-	**3939**	2.81
apex1	45	45	940	7124	4138	≥4138	-	**4054**	2.03
apex5	117	88	1221	7202	5568	≥5568	-	**4852**	12.86
s713	54	42	912	7540	5677	≥5677	-	**4917**	13.39
s1196	32	32	1120	9603	7805	**7298**	6.50	6560	15.95
too_large	38	3	1069	14782	11540	≥11540	-	**9736**	15.63
seq	41	35	1462	17839	14352	**13761**	4.12	11519	19.74
bigkey	486	421	6151	34885	25229	**24316**	3.62	22633	10.29
s13207.1	700	790	10992	103381	86134	**83740**	2.78	80005	7.12
Total				**228404**	**178635**	**173500**	**2.87**	**159562**	**10.68**

extracted and synthesized. The netlists obtained through the three implementation frameworks are then built with minimum and equally sized PXGs; this gives us the opportunity to use the number of devices (graphene p-n junctions or SiNW transistors) as a metric for area comparison. Both PDs and BBDDs models involved in the logic synthesis are reduced and ordered using the techniques illustrated in Section VI-C for PXL, or the optimizations provided in the official BBDD package.

B. On the efficiency of PXL synthesis

Table IV reports an overview of the benchmarks; columns **PI** and **PO** collect the total number of primary inputs and primary outputs, whereas column **P** reports the total number of implicants, i.e., the rows of the PLA table.

The first analysis concerns the efficiency of the OPS flow for PXL circuits. For the sake of clarity, Table IV shows the vertex-set cardinality of the PD at each stage of the synthesis flow. The column **Build & Opt.** refers to Algorithm 1 (column **w/o opt**) and Algorithm 2 (column **w/ opt**); column **Predictive sorting** shows the results after Algorithm 3; column **Genetic sorting** after Algorithm 4. Reported savings are computed with respect to optimized PXL circuits (column **w/ opt**). Total savings for each optimization step represent the effectiveness of each algorithm. Numbers in bold highlight the final outcome of the optimization process. Indeed, the GA might be bypassed in order to save CPU time using a straightforward policy: *if Predictive sorting brings to some optimization, then annotate the netlist; run Genetic sorting otherwise.* Other elaborate policies might result more efficient.

As can be seen from column **Build & Opt.**, the optimization routines (*Merge* and *Delete*) give substantial savings: 21.7% on the total number of gates. Only the benchmark `o64` gets no improvement. It represents a specific class of circuits for which the proposed rules show weaker; indeed, its PLA table consists of a diagonal of 1s (one entry per row). This singular distribution prevents any reduction (both *Merge* and *Delete*).

Concerning the CPU usage, Figure 16 shows a comparison between execution times for the synthesis of PXL (circle) and T-MUX (triangular) circuits. It is important to note that a fair comparison between the two would require applying the same sorting technique for both implementations, which is unpractical as BBDDs do not have such a feature. For that reason, only the build&reduce steps are accounted here. Numbers are normalized with respect to the fastest benchmark, `misex1` (2.4 ms). The generation of PXL algorithms requires 18× less time (on average); significant achievements are obtained with the largest circuit, namely the `s13207.1`: 22× CPU-time reduction. This proves the scalability of the process. Indeed the execution time of the proposed algorithms does not scale with circuit complexity, but it is rather affected by the amount of don't care states; the higher the number of don't cares in the circuit description, the faster the computational time. This characteristic is peculiar of the proposed method and does not apply to other synthesis strategies based on decision diagrams.

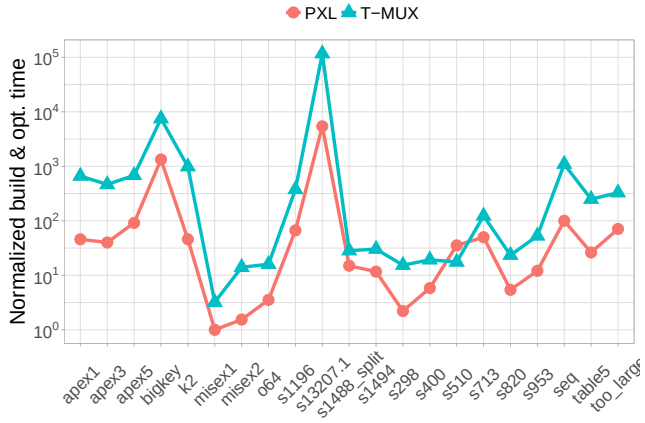

Fig. 16. Normalized CPU time required for building & optimize.

As described earlier, the effectiveness of the sorting algorithms have been quantified and selected with the following rationale: in the first place, Predictive sorting is applied; if it is able to reduce the cardinality of the PD then the network is annotated. Otherwise, the Genetic sorting algorithm is applied. The motivation lies in the fact that it is high likely that Predictive sorting will be able to perform a reasonable amount of reduction with an effortless computation, thus drastically reducing execution times. Experimental results show that predictive sorting succeeds for 11 benchmarks over 21 (highlighted numbers in column **Predictive**), with an average improvement of about 2.87% (average over the bold numbers in column **Predictive** of table IV). However, there are benchmarks for which the Predictive sorting fails, e.g. `table5` and `apex1`. This is when the Genetic sorting comes into play. Parameters like population dimensions and maximum generations are empirically defined in order to achieve a reasonable trade-off between savings and execution time. For benchmarks with less than ten thousand PXGs (refer to column **PD** in Table V) the use of a population of 40 individuals with the maximum number of generation set to 50 has proven to be effective; for larger benchmarks, best results are obtained with 20 individuals that evolve over a

maximum of 20 generations. Collected numbers clearly show that the proposed GA substantially reduces each circuit with a total gate count saving of about 11%. The only exception is `o64` which, as previously described, cannot be optimized by construction, whereas the maximum yield is for the `s510` benchmark (about 34%). Clearly, such level of optimization comes at a CPU time cost. Figure 17 reports the execution times required to perform both Predictive (square mark) and Genetic (circle mark) sorting per benchmark. Over the entire set of experiments, the heuristic approach performs best, achieving a 255× speedup. However, if we neglect largest circuits, i.e., `bigkey` and `s13207.1`, the ratio falls to a 50× factor. If, on the one hand, those results demonstrate that Predictive sorting is an affordable solution to PD reduction, on the other hand they suggest that there is room for an improvement of the Genetic algorithm. Intuitively, larger circuit descriptions require higher computational efforts when applying reduction rules. Therefore, a possible enhancement lies in the exploitation of massive parallel computations where each pool of processes explores a limited portion of the solution space.

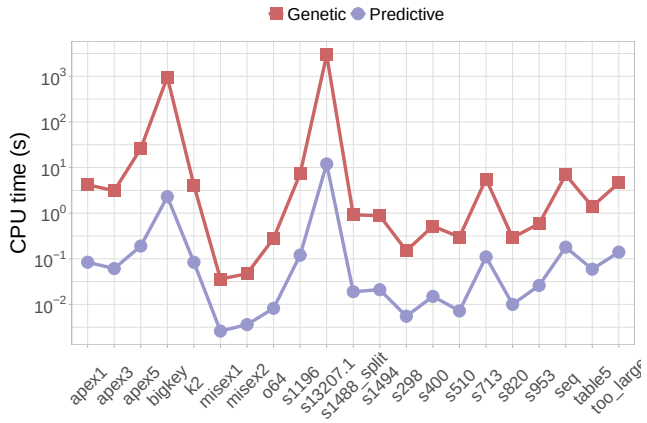

Fig. 17. CPU time: Predictive sorting vs. Genetic sorting.

C. Comparison to SoA synthesis flows

Table V shows a comparison between PXL, T-MUX, and ML circuits mapped onto graphene PXGs and Si-NWs RFETs; PXL circuits are those obtained with the sorting rationale described in Section VII-B.

The PXL implementations show 8.5× less devices with respect to the T-MUX counterparts. The best results are achieved by circuits whose implicant table contains a larger number of don't cares. Column S reports the percentage of don't cares over all the PLA entries; circuits with $S \to 1$ show the largest savings (e.g., `bigkey` with $S = 0.988$ leads to 17× less devices with respect to T-MUX), while those with $S \to 0$ results in unchanged or worse performance (e.g., `too_large` with $S = 0.636$ requires about 5× more devices than its T-MUX implementation). Those results are due to the higher expressive power of the PXL-expansion which allows to efficiently represent incompletely-specified Boolean functions with fewer EXNOR gates, especially those with a high number of don't cares. On the other hand, ML circuits result to be more efficient than both PXL and T-MUX due to the possibility of cascading common sub-expressions. Indeed, higher differences are recorded with benchmarks having more that one thousand cells, whereas for smaller circuits PXL is capable to match, if not improve, the ML outcomes, e.g.,the `s510` design. However, the proposed PXL+PD synthesis gets close to ML much more than T-MUX+BBDD does (please refer to row **Total** in Table V).

TABLE V
SYNTHESIS RESULTS COMPARISON BETWEEN PXL, T-MUX, AND ML DESIGN FLOWS. REPORTED NUMBERS REFER TO THE TOTAL AMOUNT OF DEVICES NEEDED FOR EACH BENCHMARK.

	S	Graphene			Si-NWs Transistors		
		PD	T-MUX	ML	PD	T-MUX	ML
misex1	0.523	64	126	122	128	252	122
o64	0.984	130	644	196	260	1288	196
misex2	0.74	151	350	154	302	700	154
s298	0.789	177	596	238	354	1192	238
s510	0.826	274	1250	468	548	2500	468
s820	0.707	575	1266	522	1150	2532	522
s400	0.772	574	556	270	1148	1112	270
s1488_split	0.587	1018	1204	1130	2036	2408	1130
s1494	0.587	1046	1204	1130	2092	2408	1130
s953	0.834	1021	2222	872	2042	4444	872
apex3	0.867	3112	11166	2976	6224	22332	2976
table5	0.323	4070	1866	2640	8140	3732	2640
k2	0.831	3939	9664	3822	7878	19328	3822
apex1	0.831	4054	10696	3862	8108	21392	3862
apex5	0.949	4852	40620	1842	9704	81240	1842
s713	0.846	4917	6364	752	9834	12728	752
s1196	0.732	7298	6664	2014	14596	13328	2014
too_large	0.636	9736	1950	606	19472	3900	606
seq	0.702	13761	12380	3710	27522	24760	3710
bigkey	0.988	24316	420874	12096	48632	841748	12096
s13207.1	0.986	83740	912446	8024	167480	1824892	8024
Total		**168825**	**1444108**	**47440**	**337650**	**2888216**	**47440**

Figure 18 shows the maximum circuit depth for the obtained networks. PXL networks are, on average, about 8× shallower than T-MUX networks; this translates into lower propagation delays and thus improved performance. This characteristic is correlated to the structure of the two implementations. Indeed, it is well-known that depth of tree-like structures grows linearly with the number of input variables, whereas PXL structures use the minimum number of PXG gates for each branch, thus dropping unnecessary inputs. Although ML circuits are clearly shallower than T-MUX ones, PXL networks outperform multi-level circuits with a 23% smaller depth. This underlines once again that there are margins to improve the logic synthesis when dealing with other classes of circuits, e.g., the pass-gate logic, for which depth is a key design metric.

As discussed in Section III-E, the maximum fan-out represents an important aspect for the reliability of PXL networks. The synthesized circuits have a maximum fan-out of 6.87 (average over all the benchmarks), where the `s13207.1` reaches the highest value, 19. Given the signal integrity analysis shown in Figure 10, we can state that all the graphene-based PXL circuits operate within the safe region (fan-out < 50).

D. Performance analysis

Although the effectiveness of the adiabatic charging principle has been demonstrated in other previous works [37], [13], in this section we briefly discuss the power gains brought by the proposed synthesis algorithm. Due to the lack of a free and

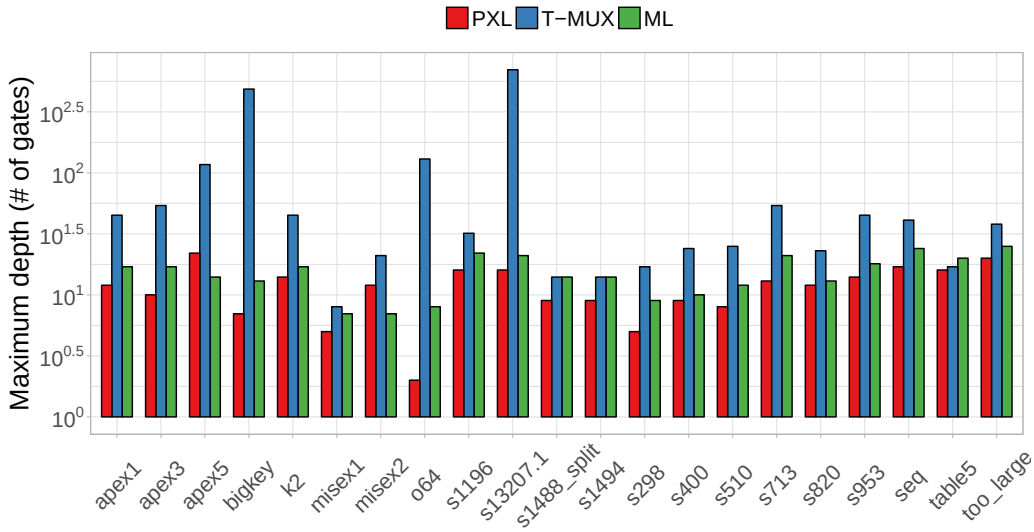

Fig. 18. Maximum depth.

open-source SiNW device model, we focus our investigation on PXL circuits composed of graphene-based PXGs. The

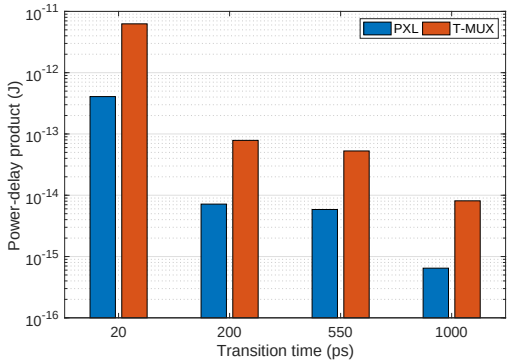

Fig. 19. Power-delay product of considered benchmarks.

plot of Figure 19 shows the power-delay product (PDP) for PXL (darker bars) and T-MUX circuits (lighter bars); numbers refer to the average over all the benchmarks reported in TableV (column **Graphene**). The PDP is measured through Spice-level simulations using the Verilog-A model introduced in [26]. The clocked-power is a ramp signal with a varying rise/fall transition time T_r, as to simulate different working conditions (refer to Section III for additional details).

On average, PXL circuits have about 12× lower PDP with respect to T-MUX counterparts (best case is 15.3× at $T_r = 20ps$). In terms of power, PXL circuits show 1.3× lower consumption that T-MUX circuits; the largest savings are observed for benchmark `s298` (7.5×), whereas `too_large` and `s953` have shown smaller savings (74.5% and 56.8% respectively). For what concerns the propagation delays, PXL circuits are 2.18× faster than T-MUX circuits; that's another advantage of the synthesis algorithm which produces more regular and less deep topologies (please refer to the bar chart reported in Figure 18).

VIII. Conclusions

In this paper we described a novel synthesis and optimization flow for pass-logic devices tailored on emerging device technologies, graphene p-n junctions and SiNWs in particular. Experimental results demonstrate that the proposed methodology is capable to outperform state-of-art synthesis tools when dealing with incompletely-specified Boolean functions. As a final remark, this work enables the logic synthesis of a new class of complex devices built with cutting-edge emerging technologies.

References

[1] M. De Marchi, D. Sacchetto, S. Frache, J. Zhang, P.-E. Gaillardon, Y. Leblebici, and G. De Micheli, "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire FETs," in *Electron Devices Meeting (IEDM), 2012 IEEE International*. IEEE, 2012, pp. 8–4.

[2] D. M. Bromberg, D. H. Morris, L. Pileggi, and J.-G. Zhu, "Novel STT-MTJ device enabling all-metallic logic circuits," *IEEE transactions on Magnetics*, vol. 48, no. 11, pp. 3215–3218, 2012.

[3] S. S. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[4] M. Y. Han, B. Özyilmaz, Y. Zhang, and P. Kim, "Energy band-gap engineering of graphene nanoribbons," *Physical review letters*, vol. 98, no. 20, p. 206805, 2007.

[5] H.-Y. Chiu, V. Perebeinos, Y.-M. Lin, and P. Avouris, "Controllable pn junction formation in monolayer graphene using electrostatic substrate engineering," *Nano letters*, vol. 10, no. 11, pp. 4634–4639, 2010.

[6] L. Amar, P. E. Gaillardon, S. Mitra, and G. D. Micheli, "New Logic Synthesis as Nanotechnology Enabler," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2168–2195, Nov 2015.

[7] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM journal of research and development*, vol. 50, no. 4.5, pp. 433–449, 2006.

[8] S. E. Thompson and S. Parthasarathy, "Moore's law: the future of Si microelectronics," *Materials today*, vol. 9, no. 6, pp. 20–25, 2006.

[9] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," *IEEE journal of solid-state circuits*, vol. 32, no. 7, pp. 1079–1090, 1997.

[10] S. Houri, G. Billiot, M. Belleville, A. Valentian, and H. Fanet, "Limits of CMOS Technology and Interest of NEMS Relays for Adiabatic Logic Applications," *IEEE Transactions on Circuits and Systems I*, vol. 62, no. 6, pp. 1546–1554, 2015.

[11] L. Ding, Z. Zhang, S. Liang, T. Pei, S. Wang, Y. Li, W. Zhou, J. Liu, and L.-M. Peng, "CMOS-based carbon nanotube pass-transistor logic integrated circuits," *Nature communications*, vol. 3, p. 677, 2012.

[12] S. Miryala, A. Calimera, E. Macii, and M. Poncino, "Ultra Low-Power Computation via Graphene-Based Adiabatic Logic Gates," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 365–371.
[13] V. Tenace, A. Calimera, E. Macii, and M. Poncino, "Quasi-Adiabatic Logic Arrays for Silicon and Beyond-Silicon Energy-Efficient ICs," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 12, pp. 1111–1115, 2016.
[14] R. S. Shelar and S. S. Sapatnekar, "BDD decomposition for delay oriented pass transistor logic synthesis," *VLSI Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 957–970, 2005.
[15] S. B. Akers, "Binary decision diagrams," *Computers, IEEE Transactions on*, vol. 100, no. 6, pp. 509–516, 1978.
[16] V. Tenace, A. Calimera, E. Macii, and M. Poncino, "One-pass logic synthesis for graphene-based Pass-XNOR logic circuits," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.
[17] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Biconditional BDD: a novel canonical BDD for logic synthesis targeting XOR-rich circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1014–1017.
[18] M. H. Ben-Jamaa, K. Mohanram, and G. De Micheli, "An efficient gate library for ambipolar CNTFET logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 242–255, 2011.
[19] B. Huard, J. Sulpizio, N. Stander, K. Todd, B. Yang, and D. Goldhaber-Gordon, "Transport measurements across a tunable potential barrier in graphene," *Physical Review Letters*, vol. 98, no. 23, p. 236803, 2007.
[20] C.-Y. Sung and J. U. Lee, "The ultimate switch," *Spectrum, IEEE*, vol. 49, no. 2, pp. 32–59, 2012.
[21] V. V. Cheianov and V. I. Falko, "Selective transmission of Dirac electrons and ballistic magnetoresistance of n-p junctions in graphene," *Physical Review B*, vol. 74, no. 4, p. 041403, 2006.
[22] S. Tanachutiwat, J. Ung Lee, W. Wang, and C. Y. Sung, "Reconfigurable multi-function logic based on graphene pn junctions," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 883–888.
[23] T. Mikolajick, A. Heinzig, J. Trommer, T. Baldauf, and W. Weber, "The RFETa reconfigurable nanowire transistor and its application to novel electronic circuits and systems," *Semiconductor Science and Technology*, vol. 32, no. 4, p. 043001, 2017.
[24] W. Weber, A. Heinzig, J. Trommer, D. Martin, M. Grube, and T. Mikolajick, "Reconfigurable nanowire electronics–A review," *Solid-State Electronics*, vol. 102, pp. 12–24, 2014.
[25] S. Miryala, A. Calimera, E. Macii, and M. Poncino, "Ultra low-power computation via graphene-based adiabatic logic gates," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 365–371.
[26] S. Miryala, M. Montazeri, A. Calimera, E. Macii, and M. Poncino, "A Verilog-A model for reconfigurable logic gates based on graphene pn-junctions," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 877–880.
[27] T. Sasao and P. Besslich, "On the Complexity of Mod-2l Sum PLA's," *Computers, IEEE Transactions on*, vol. 39, no. 2, pp. 262–266, 1990.
[28] E. Dubrova, D. Miller, and J. Muzio, "Upper bound on number of products in AND-OR-XOR expansion of logic functions," *Electronics Letters*, vol. 31, no. 7, pp. 541–542, 1995.
[29] T. Sasao, "EXMIN2: a simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued-input two-valued-output functions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, no. 5, pp. 621–632, 1993.
[30] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "On decomposing Boolean functions via extended cofactoring," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 1464–1469.
[31] "Format for physical description of Programmable Logic Arrays," http://www.cs.columbia.edu/čs6861/sis/pla.txt, 2015.
[32] M. Melanie, "An introduction to genetic algorithms."
[33] B. L. Synthesis and V. Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/ alanmi/abc/, 2014.
[34] L. Amarú *et al.*, "An Efficient Manipulation Package for Biconditional Binary Decision Diagrams," in *DATE'14*, 2014, pp. 296:1–296:6.
[35] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.
[36] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.
[37] S. Miryala, V. Tenace, A. Calimera, E. Macii, and M. Poncino, "Ultra-low power circuits using graphene p–n junctions and adiabatic computing," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 962–972, 2015.

Valerio Tenace Valerio Tenace received his B.Sc. in Electronic Engineering and M.Sc. in Computer Engineering in 2009 and 2012 respectively. He is currently Postdoctoral Research Fellow in Computer and Control Engineering at the Politecnico di Torino. His research interests include CAD solutions and algorithms for logic synthesis for CMOS and beyond-CMOS technologies, thermal-aware and variability-aware design techniques, reliability analysis and optimization algorithms and methods.

Andrea Calimera Andrea Calimera (S'08, M'11) is an Associate Professor of Computer Engineering at Politecnico di Torino. He received the Ph.D. degree in Computer Engineering from the same institution. His main research interests focus on design automation of digital circuits, with particular emphasis on methods and CAD tools for power optimization and variability compensation in nanometric CMOS and beyond-CMOS ICs.

Enrico Macii Enrico Macii (M'92, SM'01, F'06) is a full professor of computer engineering at the Politecnico di Torino. He received the PhD degree in computer engineering from the Politecnico di Torino in 1995. From 1991 to 1997 he was also an adjunct faculty at the University of Colorado, Boulder. Since 2007, he is the vice rector for the Research and Technology Transfer, Politecnico di Torino, and since 2012 also the rectors delegate for the International Affairs. He was the National FP7 ICT Delegate from 2011 to 2013, and one of the Italian Members of the Public Authorities Board of the ENIAC and ARTEMIS Joint Undertakings from 2009 to 2013. His research interests are in the design of electronic digital circuits and systems, with particular emphasis on low-power consumption aspects. In the field above he has authored more than 450 scientific publications.

Massimo Poncino Massimo Poncino (M'97, S'12) is currently a Full Professor of Computer Engineering with the Politecnico di Torino, Torino, Italy. His current research interests include several aspects of design automation of digital systems, with a particular emphasis on the modeling and optimization of low-power systems. He received a PhD in computer engineering and a DrEng degree in electrical engineering from the Politecnico di Torino. He is a Senior Member of the IEEE.