Multi-function logic synthesis of silicon and beyond-silicon ultra-low power pass-gates circuits

Multi-Function Logic Synthesis of Silicon and Beyond-Silicon Ultra-Low Power Pass-Gates Circuits

Valerio Tenace, Andrea Calimera, Enrico Macii, Massimo Poncino
Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy

Abstract

Pass-gates logic is known to be intrinsically more energy efficient than static CMOS. This feature attracted the research interest over the years and many working implementations have been demonstrated. Recent works, in particular, have shown that pass-gates logic is well suited for ultra-low power adiabatic circuits mapped on emerging technologies.

Despite the progress made, several design issues still prevent pass-gates logic circuits reaching large scale integration.

In this work we deal with the lack of synthesis tools and methodologies. We propose a multi-function decomposition engine that yields (*i*) an efficient abstract circuit modeling through a more compact data-structure, the Multi-Function Pass Diagram (MFPD) and (*ii*) an effective multi-gate area/delay-driven low-power synthesis&optimization flow.

Simulation results conducted on different technologies, i.e., silicon and graphene, demonstrate that logic circuits synthesized with the proposed tool are smaller in size and depth, hence less power consuming and faster than circuits obtained through conventional synthesis flows based on Binary Decision Diagrams.

1 Introduction

1.1 Motivation

Static CMOS has been taken as a reference style for mainstream VLSI circuits due to high noise immunity and high performance. However, other logic families have shown to be less power consuming and intrinsically more energy efficient. The *pass-gates logic*, a.k.a. *pass-transistor logic* (PTL), is the most representative one [1].

PTL circuits can implement logic functions with a lower transistor count, smaller parasitic capacitance and hence better performance [2]. Even modern CMOS libraries make use of PTL for some logic gates, e.g., flip-flops and multiplexers, because of the gain it offers. Moreover, PTL circuits offer an opportunity to work "adiabatically", namely, mimicking the adiabatic (i.e., without energy exchange) charging process [3]. For this reason as well, PTL might find space in the growing segment of ubiquitous computing, where always-on and ceaselessly connected ICs are in charge of processing huge amount of "slow" physical-data (e.g., biometric signals) with a very limited energy budget [4].

The interest in PTL and, more precisely, in adiabatic PTL, recently increased with the raising of emerging devices, such as nanoelectromechanical switches (NEMs), carbon nanotubes (CNTs), and graphene p-n junctions. For such devices, which naturally implement passive resistors, the PTL style enables the design of logic circuits with improved energy efficiency if compared to CMOS [5, 6, 7].
It might finally be the time to keep pace with PTL, but the roadmap to close the gap between PTL and CMOS still misses an important stage, that is, the implementation of logic synthesis algorithms.

1.2 Limitations of Existing Tools

While logic synthesis evolved following the growth of semi-custom CMOS libraries, synthesis for PTL have been improved only marginally. This is why, even today, PTL remains underutilized [2]. It's not a coincidence that most of the previous works do focus on circuits for very specific arithmetic functions [8, 9] or handcrafted basic Boolean logic gates [1, 10]. Indeed, when the target design turns into random logic, standard multi-level synthesis engines can't exploit the structural properties of PTL. That brings to sub-optimal implementations that typically require *ad-hoc* actions at the post-synthesis stage.
This problem is not new to the research community and several solutions have been introduced in the last years. Most of them, if not all, are closely related to the concept of Binary Decision Diagrams (BDDs) or some of its variant [11, 2, 12]. There are two main reasons behind the use of BDDs. First, there exists a one-to-one matching between the BDD representation of the logic function and the final circuit implementation; this enables the concept of *one-pass logic synthesis* [13] where logic optimization and technology mapping are carried out concurrently on the same data structure thereby saving CPU and memory usage. Second, BDDs [14] are a very mature data-structure with lots of available optimization algorithms for redundancy removal.
Despite the efficiency of BDDs as data-structure is unquestionable, BDD-based synthesis tools show many limitations. First, the tree-like structure of BDDs reflects into a "pyramidal" circuit topology with long depth, and hence large propagation delays. Second, state-of-the-art decomposition methods for BDDs construction all operate using a pre-fixed variable-order (VO), namely, the order used for variable expansion is fixed during the entire decomposition procedure, no matter what the logic function is. Since VO affects the vertex-set cardinality of BDDs, a wrong VO might result into dramatic area increase of the resulting circuit. Third, decomposition methods are constrained to a "single-function" decomposition. Such a function, here referred as $g(\mathbf{X})$, differs depending on the type of BDD variant in use, e.g., MUX for standard BDDs [14], XOR for Biconditional-BDDs [12]. Logic circuits dominated by $g(\mathbf{X})$, e.g., XOR-rich arithmetic circuits, take advantage of this characteristic, others, like random logic circuits, may suffer from sub-optimal minimization. While the first two issues have been addressed in [15] with the introduction of the *Pass Diagram* (PD) data-structure and the non-fixed VO decomposition, this work elaborates on the third issue, i.e., how to overcome "single-function" decomposition.

1.3 Contribution

The objective of this work is to improve the quality of pass-gates logic synthesis by means of concurrent "multi-function" decomposition. We generalize/extend the concept of the PD data structure and pass-XNOR logic (PXL) [15] to *multi-function pass-diagrams* (MFPDs) and *multi-gate pass logic* (MGPL) respectively. The MGPL circuits show a power-delay-product (PDP) which is 3 orders of magnitude smaller than that of standard PTL circuits synthesized using BDDs.
We introduce an automatic tool, called *Kanon*, which serves as a low-power synthesis and optimization engine. It consists of two main stages: (*i*) *multi-function decomposition* for a user-defined Boolean operators library (e.g., AND, OR, XOR and their complement), and (*ii*) *redundancy removal* by means of new minimization rules. Moving from single- to multi-function decomposition can be conceptually seen as the shift from two-level to multi-level synthesis in CMOS.
In order to quantify the efficiency of (*i*) the proposed multi-function decomposition, (*ii*) the MFPD model and (*iii*) the resulting MGPL style, we apply our tool to a sub-set of generic benchmarks mapped onto three different technologies, i.e., Silicon MOS transistors, Ambipolar Silicon Nanowires and Graphene p-n Junctions. While the use of generic benchmarks avoids biased results due to the presence of circuits dominated by a specific function, the use of different technologies demonstrates that the proposed technology-independent solution well fits both silicon and beyond-silicon devices. Indeed, each technology comes with its own preferential primitive, i.e., the one with higher expressive power; having the chance to exploit different Boolean operators improves the quality of the resulting circuits. Experimental results show that MFPDs obtained with our tool are smaller in size and lower in depth if compared to state-of-the-art abstract models based on BDDs (93% smaller, 95% shallower) and multi-level Boolean networks used in commercial tools (77% shallower). This reflects into MGPL circuits that are more energy efficient, smaller in area and faster in delay, regardless of the adopted technology.

2 Building MFPDs

2.1 Multi-Function Decomposition

The key step of any logic synthesis algorithm is the decomposition of a logic function through the logic primitives made available by the technology in use. Since most techniques are fine-tuned for multi-level logic representations, we propose an *ad-hoc* decomposition that is fully compliant with the requirements of pass-gates logic.
The multi-function decomposition proposed in this work relies on the basic assumption that any Boolean equation given in the form of *sum-of-products* (SOPs), or *product-of-sums* (POSs), can be decomposed by means of a user-defined set of logic connectives $\mathcal{G} = \{g : B^2 \rightarrow B\}$. Let's take a function $f(S)$ with support-set $S = \{x_1, x_2, x_3\}$ described by the

following SOP:

$$f(S) = (x_1 \wedge \neg x_4) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3) \tag{1}$$

Using the distributive property and the identity rule, $f(S)$ can be easily expanded as a sequence of cubes with cardinality of two literals:

$$\begin{aligned} f(S) = (x_1 \wedge \neg x_4) \vee (\neg x_1 \wedge \neg x_2) \\ \wedge (x_3 \wedge 1) \vee (x_1 \wedge x_2) \wedge (x_3 \wedge 1) \end{aligned} \tag{2}$$

Each product can then be rewritten using the Boolean connectives $g \in \mathcal{G}$ by means of duality. For instance, let's assume the availability of two connectives $\mathcal{G} = \{\{x \neg\vee\ y\}, \{x \neg\oplus y\}\}$, where the first one, the NOR ($\neg\vee$ symbol), has higher priority (i.e., is processed first). Then $f(S)$ is *NOR*-decomposed as shown in 3:

$$\begin{aligned} f(S) = (\neg x_1 \neg\vee x_4) \vee (x_1 \neg\vee x_2) \wedge (\neg x_3 \neg\vee \neg x_3) \\ \vee (\neg x_1 \neg\vee \neg x_2) \wedge (\neg x_3 \neg\vee \neg x_3) \end{aligned} \tag{3}$$

since $(\neg x_3 \neg\vee \neg x_3)$ is a common term, it can be factorized as shown in (4).

$$\begin{aligned} f(S) = (\neg x_1 \neg\vee \neg x_4) \\ \vee (\neg x_3 \neg\vee \neg x_3) \wedge [(x_1 \neg\vee x_2) \vee (\neg x_1 \neg\vee \neg x_2)] \end{aligned} \tag{4}$$

The second operator in $\mathcal{G}$, the XNOR ($\neg\oplus$ symbol), can now come into play. Indeed, the term $(x_1 \neg\vee x_2) \vee (\neg x_1 \neg\vee \neg x_2)$ can be represented as the XNOR between x_1 and x_2. We call this operation *Boolean substitution*. As a final result of the multi-function decomposition, the original function (1) is decomposed as described in (5).

$$f(S) = (\neg x_1 \neg\vee x_4) \vee (x_1 \neg\oplus x_2) \wedge (\neg x_3 \neg\vee \neg x_3) \tag{5}$$

It is easy to check that (5) is Boolean equivalent to (1) with 25% of literal savings.
The efficiency of the proposed multi-function decomposition is closely related to (i) the set of Boolean operators and (ii) their order in $\mathcal{G}$. Although several options do exist, we propose a technology-instructed strategy: available operators in $\mathcal{G}$ are sorted in terms of their *expressive power*[1] (EP) for the target technology, largest EP first. This contributes to making our tool more flexible and, therefore, orthogonal to different technologies.
As will be shown later in the text, different primitives are used during different stages of the multi-function decomposition. For the sake of clarity we define the first operator in $\mathcal{G}$, the one with the highest EP, as the *primary* primitive, the remaining ones as the *secondary* primitives.

[1]Ratio between the complexity of the logic operator and the number of devices needed to implement the corresponding logic gate

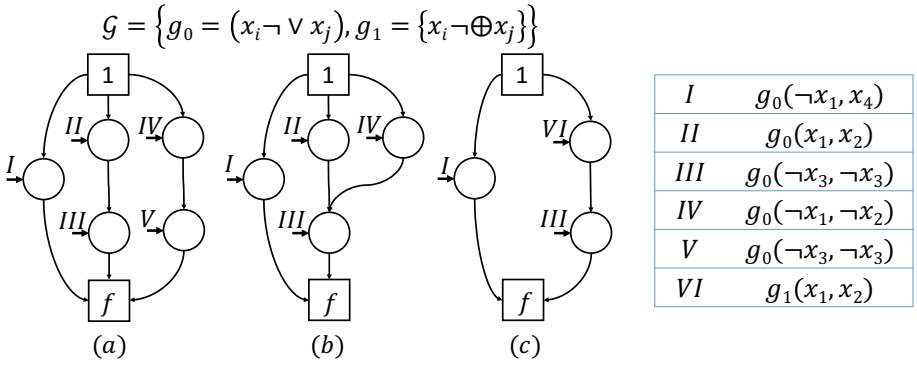

Figure 1: MFPD of function of Equation (3) before optimization (a), after merging of common sub-graphs (b), and after Boolean substitution (c).

2.2 Multi-Function Pass Diagrams (MFPDs)

Representation of logic circuits during optimization needs an abstract model. We introduce the MFPD, a simple, yet efficient abstract model for one-pass synthesis of pass-gates logic circuits.

Given a generic multi-input/single-output Boolean function f with support-set $S = \{x_1, ..., x_N\}$, its MFPD (Figure 1) representation is a polarized, directed acyclic graph defined as $G = (\Phi \cup V \cup \Theta \cup A)$. The set of internal nodes $v \in V$ are labeled as $g(x, y)$, with $g \in \mathcal{G}$ a two-input primitive Boolean connective and $x, y \in S$. Each internal node v has one outgoing edge $a \in A$ representing the logical conjunction (AND) with the successor node. The terminal node with *indegree* 0 represents the root of the MFPD, where the function starts to be evaluated; the terminal node with *outdegree* 0 is the leaf of the MFPD, the output of the function f. Multiple output functions are represented by many MFPDs as the number of outputs. As an example, Figure 1-(a) shows the MFPD structure for the function (3) with $g_0 = (x_i\neg \vee x_j)$ and $g_1 = (x_i\neg \oplus x_j)$.

The strength of MFPDs is the capability of supporting multi-function decomposition. This degree of freedom comes at the cost of canonicity, that is, MFPDs do not have an unique representation of Boolean formulae. However, relaxing the canonicity constraint is a well-accepted concept in the EDA community; for instance, And-Inverter Graphs (AIGs) integrated into commercial multi-level logic synthesis tools are non-canonical representations, but nonetheless they are likely used because more compact and manageable.

2.3 Algorithms

2.3.1 Building MFPD

Algorithm 1 shows the pseudo-code of the *Build* routine we implemented for multi-function decomposition and MFPD construction.

The main input parameters are (i) a tabular description T of the Boolean function and (ii) the primary connective (the first operator in the list of primitives $\mathcal{G}$). Table T can be a

Algorithm 1: MFPD build

```
   Input: PLA Table T, Primary connective g_0 ∈ G
   Output: Multi-Function PD MFPD
 1 MFPD = ∅
 2 foreach row R ∈ T do
 3 |   CUBES_R = ∅
 4 |   DontCareSet = DetermineDCS(R)
 5 |   foreach primary input PI ∈ R do
 6 |   |   if PI ∉ DontCareSet then
 7 |   |   |   CUBES_R.append(PI)
 8 |   |   end
 9 |   end
10 |   foreach v_{i,k} ∈ CUBES_R do
11 |   |   NewNode ← SetPolarity(v_{i,k}, g)
12 |   |   PT_R.append(NewNode)
13 |   end
14 |   MFPD.append(PT_R)
15 end
```

non-minimized *implicant* table (i.e., not prime) and can be obtained through any Verilog compiler, e.g., *ABC* [16]; obtaining T is not a computational intensive task. We refer to T as the *PLA* table. As an example, Table 1 shows the PLA table for the Boolean function (2); the character '-' identifies a dont'care.

Table 1: PLA table of function (2)

$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	x_4	$\mathbf{f}$
1	-	-	0	1
0	0	1	-	1
1	1	1	-	1

The MFPD is generated branch-wise, that is, for each row of the PLA table, i.e., for each product term of the function, nodes are appended in series by iterating the following sequence of operations:

Cube sequence generation (line 3-9) – variables not belonging to the don't-care set are included in the cube list $CUBES_R$ in order of appearance; those belonging to the dont-care set are dropped. For odd sequences, the last variable is paired with '1' logic so as to maintain Boolean equivalence. For instance, considering Table 1, for the first row $CUBES_1 = \{(x_1, \neg x_4)\}$, for the second row $CUBES_2 = \{(\neg x_1, \neg x_2), (x_3, 1)\}$, for the third row $CUBES_3 = \{(x_1, x_2), (x_3, 1)\}$.

Node generation (line 10-14) – for each pair of cubes stored in $CUBES_R$, the polarity of the variables are fixed according to the *primary* Boolean connective g and the resulting nodes are appended on the current branch. Let us consider $CUBES_2$ which contains two cubes, $(\neg x_1, \neg x_2)$ and $(x_3, 1)$; with g the NOR operator (like the example in Section 2.1), variables are complemented (by De-Morgan) as (x_1, x_2) and $(\neg x_3, \neg x_3)$ respectively.

Given a table T with N implicants and M literals, the proposed *build* routine has a complexity of $O(N \cdot M)$

Algorithm 2: MFPD optimization algorithm

```
   Input: MFPD, Secondary connectives G = (g_1, ... g_m) ∈ 𝒢
   Output: Optimized Multi-Function PD OMFPD
 1 OMFPD = ∅
 2 foreach path P ∈ MFPD do
 3 |   C_M ← ∅
 4 |   C_E ← ∅
 5 |   foreach path Q ∈ MFPD, with Q ≠ P do
 6 |   |   if SameSupport(P, Q) then
 7 |   |   |   if CheckBoolSub(P, Q, G) then
 8 |   |   |   |   C_E.append(Q, g_k ∈ G)
 9 |   |   |   end
10 |   |   else
11 |   |   |   if SharedNodes(P, Q) then
12 |   |   |   |   C_M.append(Q)
13 |   |   |   end
14 |   |   end
15 |   end
16 |   if |C_E| > 0 then
17 |   |   M ← ApplyBoolSub(P, C_E, G)
18 |   else if |C_M| > 0 then
19 |   |   M ← MergeIsomorphic(P, C_M)
20 |   OMFPD.append(M)
21 end
```

2.3.2 Optimization

Algorithm 2 describes the pseudo-code of the optimization stage for redundancy removal. It implements two different optimization techniques: *(i)* node elimination by *Boolean substitution*; *(ii)* merging of isomorphic sub-graphs. While the latter is reminiscent of standard reduction rules from BDDs [14], the former one is an *ad-hoc* strategy for MFPDs. Its purpose is to find suitable equivalent logic connectives, among the list of *secondary* connectives in $\mathcal{G}$, that can be eventually substituted in order to enable node elimination and reduce the MFPD cardinality; as illustrated in the examples of Section 2.1. Please note that *secondary connectives* are selected with a greedy approach, that is, the first one that satisfies the Boolean equivalence is instantiated in the network.

Input parameters of Algorithm 2 are the MFPD obtained through the *MFPD Build* routine, and the list of secondary connectives $G \in \mathcal{G}$.

Candidate selection (line 3-15) – Each root-to-leaf path P of the MFPD is compared with any other path Q ($P \neq Q$). If (line 6) P and Q share the same support set (i.e., nodes in P and Q are driven by the same literals) the algorithm checks (line 7) whether it is possible to perform a *Boolean substitution*, namely, it checks whether some of the operators associated with the nodes in Q can be substituted with some other operator $g_k \in G$ s.t. Boolean equivalence is satisfied. If so, P and Q share a common node expressed by means of g_k, that allows to merge P and Q in a single path. Therefore, Q is stored in the candidates list C_E together with the connective g_k that enables its elimination. If P and Q do not have common support set (line 10), the algorithm checks whether a path Q shares at least one node with P; if so, Q is a potential candidate for node merging and it is temporarily stored in the list of candidates C_M.

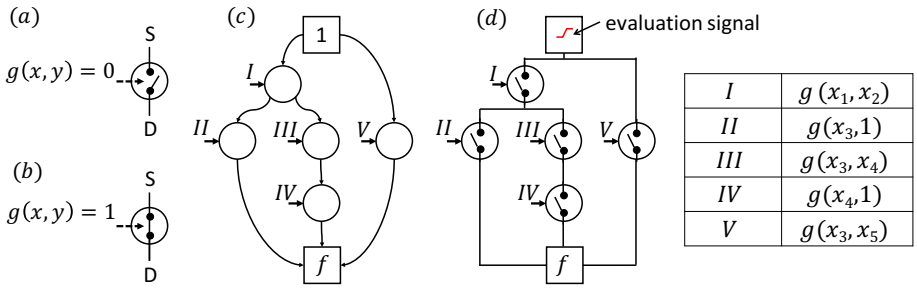

Figure 2: MGLP circuit example, where $f = (x_1\neg\vee x_2)\wedge[(x_3\neg\vee 1)\vee((x_3\neg\vee x_4)\wedge(x_4\neg\vee 1))]\vee(x_3\neg\vee x_5)$

Merge and Eliminate (line 16-20) – once candidates have been selected, the algorithm first evaluates whether there exists at least one candidate for node elimination by *Boolean substitution* ($|C_E| > 0$). If so, the common node between P and C_E is replaced with the new connective g_k, and redundand paths in C_E are removed (*ApplyBoolSub* function). If not and the list C_M is not empty, then common nodes between C_M and P are evaluated for merging through the *MergeIsomorphic* function.
Figure 1-(b) and 1-(c) show the results of the optimization procedures described above applied on the MFPD obtained through the build function (Figure 1-(a)).
For what concerns complexity, since each path P is compared with any other path Q, the total number of loops is $\frac{N\cdot(N-1)}{2}$, with N the number of paths in the starting MFPD. The complexity of the optimization routine is $O(N^2)$. Notice that all other sub-routines have a $O(1)$ complexity (operations are completed in constant time) except for functions *SameSupport* and *SharedNodes* which show a complexity of $O(M)$, with M the number of nodes in the path Q.

3 Multi-Gate Pass Logic

The *Multi-Gate Pass Logic* (MGPL) style can be seen as a generalization of the existing PXL style. The physical primitives of a MGPL network are the *pass-gates* (PGs), that, from a functional point of view, can be seen as *function*-controlled switches. They consist of two *logic-terminals* fed by the input logic signals (x and y in Figure 2-(a)), and two *transmission terminals*, one playing as the source of an evaluation signal and the other as the collector (S and D in Figure 2-(a)). The control function is a two-input Boolean operator $g(x, y)$ between the x and y logic inputs; when $g(x, y) = 1$ the PG is ON (low-resistance), Figure 2-(a), when $g(x, y) = 0$ the PG is turned OFF (high-impedance), Figure 2-(b). PGs with different control functions can be designed depending on the technology in use. An MGPL circuit (Figure 2-(d)) has a 1-to-1 mapping with its MFPD (Figure 2-(c)) It consists of logic paths connected in parallel between a clocked-power supply (the root) and the main output (the leaf). Each path consists of a cascade of independent PGs driven by primary inputs. When activated (all PGs turned-ON), a logic path creates a low-resistive gateway through which the clocked-power signal can flow from the root to the leaf. Under this

condition the circuit's output is evaluated as 1-logic[2]. Logic paths are in mutual exclusion by construction, that is, for a given input pattern one and only one path can be eventually activated. In case none path is activated the circuit's output is evaluated as 0-logic[3].
As for other dynamic logic families, MGPL circuits work in two phases. In the *configuration* phase the input signals are evaluated by the PGs and the resistive paths of the network are set up. In the *evaluation* phase the clocked power signal is pre-charged and propagated through the network[4].
It is worth emphasizing that although the MGPL resembles the PTL structure, the difference is substantial. In PTL circuits, transistors are used as switches that deviate the current flow to different paths; on the contrary PGs are used as switches to open/close a logic path. This is reflected by the model used to represent the circuit. Indeed, BDDs are not the most intuitive representation as PG gates do not implement any deviation of the signal. Second, while in PTL an output is always connected to a static power supply terminal, V_{dd} if '1' or Gnd if '0', output evaluation in MGPL logic is dynamic: current is flowing if '1', not flowing if '0'. Alternatively, one can see MGPL circuits as a half way between CMOS and PTL. As in CMOS series/parallel connections between gates are available, as in PTL, information is carried out by means of root-to-leaf current flow.

3.1 Pass-Gate Devices

New logic primitives introduced by emerging technologies represent a perfect fit to the structure of PGs. Figure 3 pictorially describes some of them. In particular, Figure 3-(a) shows four PG embodiments using Ambipolar Silicon-NanoWires (Si-NW) [17]. The first two (left) are composed of a single Si-NW transistor and implement the AND and NOR logic gates. The remaining two (center and right) consist of a pair Si-NW transistors and implement the XNOR or XOR logic gates.
Figure 3-(b) shows two possible pass-gates using standard MOSFET transmission-gates. The first one (left) implements the AND, whereas the second one (right) implements the NOR. Since both configurations require four MOSFETs, silicon devices have less expressive power.
Finally, Figure 3-(c) shows pass-gates mapped on graphene p-n junctions [18]. A graphene p-n junction consists of two metal back-gates (blue and green triangles) driven by logic signals (x and y). Logic signals with same polarity turn the junction ON. The first PG (top left) implements the NOR gate; the outer input connections x and y are both compared to a logical-0 reference. It works as follows: when both x and y are set to 0-logic, the input evaluation signal (red ramp) is allowed to propagate; in all the remaining cases at least one p-n junction is OFF and the evaluation signal is stopped. Similarly, the second pass-gate (bottom left) implements the AND; the evaluation signal propagates *iff* both x and y are fed wit 1-logic. Notice that for NOR and NAND SiNWs need less devices (1 vs. 2). The last

[2] A Sense Amplifier can be used at the main output in order to quickly identify the 1-logic and reshape the clock-supply signal.

[3] We assume the output node is regularly sampled through a standard clock synchronized with the clocked-power

[4] A synchronization between input signals and clocked-power is needed; this aspect is a circuit level detail which is out of the scope of this work.

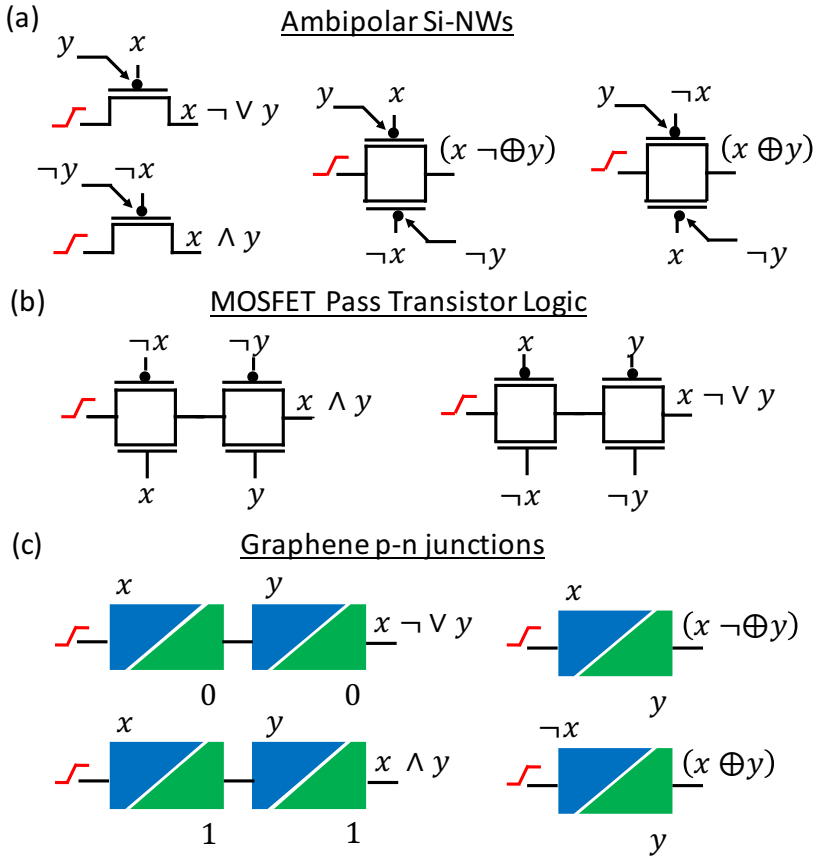

Figure 3: Possible PGs for different logic primitives.

two pass-gates (top and bottom right) implement the the XNOR and XOR gates. In this case graphene shows higher expressive power than SiNW. It is therefore clear how different technologies can be better exploited using different logic primitives.

4 Simulation Results

The experimental results reported in this section provide a fair comparison against state-of-the-art solutions. The objective is threefold: (*i*) quantify the higher expressive power obtained through "multi-function" decomposition, (*ii*) demonstrate the flexibility of MFPDs and the efficiency of the synthesis tool we implemented (*iii*) show that the MGPL style allows large gains w.r.t. PTL and, most importantly, it is well suited for ultra-low power digital circuits.

We set up five different synthesis flows, the first four are for pass-gates logic circuits, the target of this work, the fifth one is for standard cells-based circuits.

1. MFPDs (the solution proposed in this work): circuits described using the PLA format [16] are processed with our tool *Kanon* for multi-function decomposition using the connec-

Table 2: Binary MFPDs efficiency w.r.t. PDs, BDDs, BBDDs and AIGs

	PI	PO	I	MFPD Nodes w/o opt	MFPD Nodes w/ opt	MFPD Savings [%]	MFPD Levels	PD Nodes	PD Levels	BBDD Nodes	BBDD Levels	BDD Nodes	BDD Levels	AIG Nodes	AIG Levels
sao2	10	4	58	229	152	21.40	5	267	10	92	10	170	10	269	19
o64	130	1	65	65	65	0.00	1	130	2	322	130	70	65	195	10
5xp1	7	10	75	161	111	29.81	3	201	6	67	7	92	7	241	20
c8	28	18	79	156	108	23.72	5	209	10	624	28	183	28	232	13
duke2	22	29	87	401	287	20.70	8	582	15	1281	22	1025	22	950	35
apex1	45	45	206	921	677	24.32	8	1217	16	16085	45	28427	45	3998	41
misex1	8	7	32	67	31	47.76	3	51	5	71	8	57	8	122	13
misex2	25	18	29	101	75	27.72	6	133	12	356	25	180	25	153	12
b12	15	9	431	1007	579	36.64	3	1242	6	142	15	112	15	156	14
k2	45	45	936	3791	2103	42.44	8	3870	15	6353	45	28427	45	3822	32
bigkey	486	421	6151	19054	10771	35.95	4	24772	8	221630	486	7044	486	12095	23
s13207.1	700	790	10987	53868	33005	20.87	9	71675	17	1450670	700	678161	700	8025	42
Total				**79821**	**47964 (1x)**	**39.91**		**104349 (2.17x)**		**1697693 (35.39x)**		**743948 (15.51x)**		**30258 (0.63x)**	
Average							**6 (1x)**		**11 (1.83x)**		**127 (21.12x)**		**122 (20.33x)**		**23 (3.83x)**

tive set $\mathcal{G} = \{\{x\neg \vee y\}, \{x\neg \oplus y\}, \{x \oplus y\}\}$[5]; the resulting MFPDs are mapped onto MGPL circuits using different technologies.

2. PDs (introduced in [15]): circuits described using the PLA format [16] are processed using *Gemini*, a single-function XNOR decomposition tool; resulting PDs are mapped onto PXL circuits using different technologies.

3. Biconditional-BDDs (described in [12]): circuits are first synthesized using a standard multi-level synthesis tool and then translated into BDDDs using single-function XOR decomposition scheme; the resulting BBDDs are mapped onto a PTL-like (i.e., tree-based) structure using different technologies.

4. BDDs: circuits are processed with the CUDD package [14]; BDD structures, obtained with a single-function MUX-based decomposition, are mapped on PTL-like circuit using different technologies.

5. AIGs: obtained with the ABC synthesis tool [16]; AIGs are mapped on a CMOS library containing only AND and INV gates.

It is worth emphasizing that AIGs can't be directly used for pass-gates logic circuit; we included AIGs as a reference point to better evaluate MFPDs.

The experiments were run on a set of open-source benchmarks from the LGSynth91 suite [19], and accurate SPICE simulations were used for the characterization of the obtained netlists. Please note that the size of such benchmarks is comparable to that of those used in other synthesis-related works, e.g., [12]. Without loss of generality, only combinational logic cones have been considered for synthesis, i.e., in-to-out and register-to-register logic cones. Table 2 gives a summary of the results. Columns **PI**, **PO** and **I** represent the total number of primary inputs, primary outputs and implicants of each benchmark. Under the labels **MFPD**, **PD**, **BBDD**, **BDD** and **AIG** we report the figures of merit of each data-structure. Regarding MFPDs, the column **w/o opt** refers to MFPDs after the build process, while column **w/ opt** refers to MFPDs after optimization (column **Savings** reports optimization gain); for **BBDD** and **BDD** numbers refer to optimized structures.

[5]Even though more Boolean operators can be used, here we force our tool working in a worst-case conditions where only three primitives are allowed.

Let us first consider the MFPD structure. The proposed reduction rules allow to save on average about 40%. Noticeably, large savings have been recorder for all the benchmarks, except for `o64`. For this case we observed the PLA table is a diagonal matrix of '1s' which prevents MFPD optimization.
Regarding the expressive power, MFPDs clearly outperform *BBDDs* (which are 35.39x larger), *BDDs* (15.51x larger) and *PDs* (2.1x larger). Only *AIGs* are more compact (0.63x); indeed, their main strength is the possibility of reusing cascades of common sub-expressions and local don't-care conditions, which is not allowed on pass-gates logic.
Another important aspect concerns the depth of the data- structures. Also in this case MFPDs are more efficient, not just w.r.t. *BBDDs* (which are 21.12x deeper), *BDDs* (20.33x deeper) and *PDs* (1.83x deeper), but also when compared to *AIGs* (3.83x deeper). MFPDs are indeed well suited for pass-gates logic circuits, where smaller depth translates into shorter delays and smaller voltage noise. The huge savings achieved are the consequence of the efficient multi-function decomposition, in particular: (*i*) the availability of more Boolean operators w.r.t. *BDDs*, *BBDDs* and *PDs*, (*ii*) the fact that inputs variables belonging to the dont-care set are dropped during decomposition (see Algorithm 1), (*iii*) the regularity of the implication table that allows large minimization (see Algorithm 2).
Regarding the CPU execution time, the MFPD synthesis is, on the average, 38x faster w.r.t. the procedures used for decision diagrms. For instance, the MFPD for the largest benchmark (`s13207.1`) is built and optimized in 7.08s, whereas the equivalent BBDD takes 241.9s. This is due to the lower computational workloads of MFPD manipulation algorithms which avoid diagram reconstruction during optimization.

Table 3: Device count after synthesis and mapping.

	Graphene		**Ambipolar (Si-NW)**		**PTL**	
	MFPD	**BBDD**	**MFPD**	**BBDD**	**MFPD**	**BBDD**
sao2	304	184	152	368	608	1472
o64	130	644	65	1288	260	5152
5xp1	216	134	117	268	516	1072
c8	215	1248	109	2496	444	9984
duke2	570	2562	291	5124	1196	20496
apex1	1341	32170	690	64340	2864	257360
misex1	56	142	37	284	196	1136
misex2	146	712	79	1424	348	5696
b12	1124	284	613	568	2724	2272
k2	4172	12706	2137	25412	8820	101648
bigkey	21535	443260	10778	886520	43168	3546080
s13207.1	65662	2901340	33353	5802680	136196	23210720
Total	**95471**	**3395386**	**48421**	**6790772**	**197340**	**27163088**
	(1x)	**(35.56x)**	**(1x)**	**(140.24x)**	**(1x)**	**(137.64x)**

To demonstrate the "orthogonality" of both MFPDs and the MGPL style over different technologies, we mapped the benchmarks under analysis using three different types of devices: graphene p-n junctions (*Graphene*), Silicon NanoWires Pass-Transistors (*Si-NW PT*), and traditional MOSFET-based Pass-Transistors (*Si-MOS PT*). As briefly described in Section

3.1, each of these technologies has different "optimal" (i.e., with highest expressive power) primitives. Table 3 reports the post-synthesis results obtained using our tool. Since MFPDs are a superclass of Pass Diagrams, we only provide comparison to BBDD-based synthesis. BBDDs represent the most recent solution proposed for emerging technologies [12] and their superiority w.r.t. other solutions have been already demonstrated. Notice that MF-

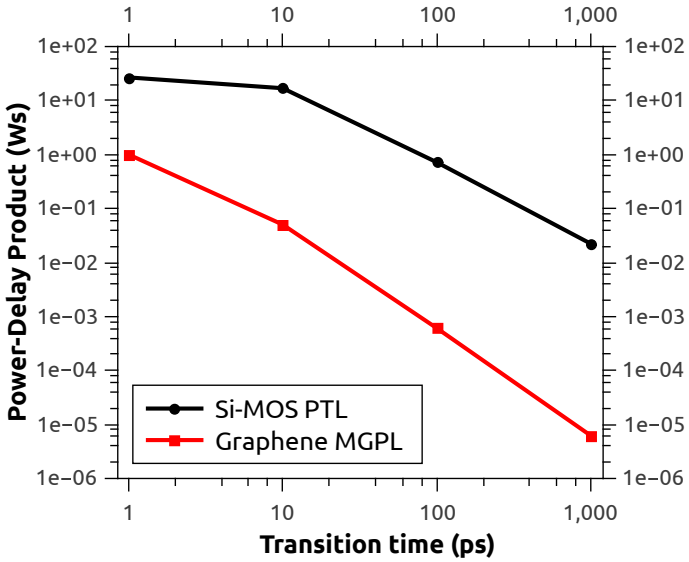

Figure 4: Normalized PDP vs transition time.

PDs nodes can be mapped with NOR, XOR and XNOR, while BBDDs only allow XOR mapping. Each of this pass-gates count different devices depending on technology (Figure 3). As a result of the multi-function decomposition, circuits synthesized using MFPDs are smaller in size, hence more area and power efficient. The more compact structure of MGPL circuits allows very high power/energy reduction. We underline the energy efficiency of the MGPL style for emerging technologies, Graphene in particular. Figure 4 provides a technological comparison between Graphene-based MGPL circuits and Silicon-MOS PTL circuits. The plot shows the power-delay product (PDP) averaged over all the benchmarks as function of the transition time T_r of the input signals. The plot highlights the "adiabatic" nature of both implementations, i.e., PDP reduces as T_r increases. However, and this is the most important aspect, graphene circuits are more energy efficient, not just in terms of absolute numbers (mainly due to the intrinsic characteristics of the material, e.g., very low voltage drop [15]), but also in terms of "scalability". For a range of transition times of 3 orders of magnitude (1 to 1000 ps), the PDP of graphene reduces by more than 5 orders of magnitude, whereas that of silicon reduces only by 3 orders of magnitude.

5 Conclusions

In this work we introduced a novel abstract representation for Boolean switching functions: the MFPD. Such structure is obtained with a multi-function logic decomposition that allows very compact circuit representations, the MGPL style. The proposed logic synthesis algorithms integrated within our tool (*Kanon*) demonstrate that MFPD synthesis show superior characteristics w.r.t. state of the art solutions, in particular *(i)* higher area efficiency (almost 15.51x better than BDDs) and *(ii)* shallower logical circuits (77% w.r.t. AIGs).

References

[1] 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, Jul 1997.

[2] 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.

[3] V. G. Oklobdzija *et al.*, "Pass-transistor adiabatic logic using single power-clock supply," *IEEE Transactions on Circuits and Systems II*, vol. 44, no. 10, pp. 842–846, 1997.

[4] C. Perera *et al.*, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[5] S. Houri *et al.*, "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.

[6] L. Ding *et al.*, "Cmos-based carbon nanotube pass-transistor logic integrated circuits," *Nature communications*, vol. 3, p. 677, 2012.

[7] S. Miryala *et al.*, "Ultra Low-Power Computation via Graphene-Based Adiabatic Logic Gates," in *DSD'14*, 2014, pp. 365–371.

[8] M. Suzuki *et al.*, "A 1.5-ns 32-b cmos alu in double pass-transistor logic," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 11, pp. 1145–1151, Nov 1993.

[9] J. D. Lee *et al.*, "Application of dynamic pass-transistor logic to an 8-bit multiplier," *Journal-Korean Physical Society*, vol. 38, no. 3, pp. 220–223, 2001.

[10] T.-Y. Wu *et al.*, "Low-leakage and low-power implementation of high-speed 65nm logic gates," in *EDSSC'08*, Dec 2008, pp. 1–4.

[11] V. Bertacco *et al.*, "Decision diagrams and pass transistor logic synthesis," in *Int'l Workshop on Logic Synthesis*, vol. 168, 1997.

[12] L. Amaru *et al.*, "Biconditional binary decision diagrams: A novel canonical logic representation form," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 487–500, Dec 2014.

[13] R. Drechsler and W. Günther, *Towards One-Pass Synthesis*. Springer Science & Business Media, 2013.

[14] F. Somenzi, "Cudd: Cu decision diagram package release 2.3. 0," *University of Colorado at Boulder*, 1998.

[15] V. Tenace *et al.*, "One-pass logic synthesis for graphene-based Pass-XNOR logic circuits," in *DAC'15: ACM/IEEE Design Automation Conference*, 2015, pp. 1–6.

[16] B. L. Synthesis and V. Group, "Abc: A system for sequential synthesis and verification," http://www.eecs.berkeley.edu/ alanmi/abc/, 2014.

[17] M. De Marchi *et al.*, "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire fets," in *IEDM*, 2012, pp. 4–8.

[18] S. Miryala *et al.*, "Ultra-low power circuits using graphene p–n junctions and adiabatic computing," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 962–972, 2015.

[19] "Collection of digital design benchmarks," http://goo.gl/6fOVfN, 2015.