MIXSyn: An Efficient Logic Synthesis Methodology for Mixed XOR-AND/OR Dominated Circuits

Luca Amarú, Pierre-Emmanuel Gaillardon, Giovanni De Micheli
Integrated Systems Laboratory (LSI), EPFL, Switzerland

***Abstract*—We present a new logic synthesis methodology, called *MIXSyn*, that produces area-efficient results for mixed XOR-AND/OR dominated logic functions. *MIXSyn* is a two step synthesis process. The first step is a hybrid logic optimization that enables selective and distinct optimization of AND/OR and XOR-intensive portions of the logic circuit. The second step is a library-free technology mapping that enhances design flexibility with a tractable computational cost. *MIXSyn* has been tested on a set of large MCNC benchmarks. Experimental results indicate that *MIXSyn* produces CMOS circuits with 18.0% and 9.2% fewer devices, on the average, with respect to state-of-art academic and commercial synthesis tools, respectively. *MIXSyn* is also capable to exploit the opportunity of novel XOR implementations offered by the use of double-gate ambipolar devices. Experimental results show that *MIXSyn* can reduce the number of ambipolar transistors by 20.9% and 15.3%, on the average, with respect to state-of-art academic and commercial synthesis tools, respectively.**

I. Introduction

Automated logic synthesis has been the corner stone of modern electronic design automation methodology since the past 25 years and still plays a fundamental role in supporting the exponential growth of *Application Specific Integrated Circuits* (ASICs) design complexity. A standard logic synthesis flow for ASICs consists of two main phases: logic optimization and technology mapping. Logic optimization transforms the circuit to minimize its implementation cost, while technology mapping transposes it onto its best standard-cell implementation. In these general terms, logic optimization and technology mapping are intractable [1]. Many heuristic methods have been proposed in the past to solve these problems in polynomial time complexity while maintaining a good results quality [2]. However, the efficiency of these heuristics is heavily dependent on the targeted circuit type, i.e. on the type of logic function that designers want to implement in silicon. Unfortunately, in a majority of real-life applications, different types of functions coexist within the same circuit. The most frequent functions are AND/OR and XOR intensive functions. NAND/NOR functions have a compact implementation in the well-established *Complementary Metal Oxide Semiconductor* (CMOS) technology. This property has driven the development of multilevel AND/OR-optimization methods, e.g. algebraic factorization techniques [3]–[5]. XOR-optimization has received less attention in the past but good methods based on *Binary Decision Diagrams* (BDDs) have been recently proposed to deal with XORs [6], [7]. On the technology mapping perspective, the efficiency of the mapping operation mainly depends on the richness of the standard-cell library [12] and on the capability to recognize if each library element (logic gate) can implement a certain portion of the logic function. The richness of the library is usually pre-determined by the standard cell full custom design carried out off-line. Instead, the recognition task, commonly called *matching*, is often solved by Boolean matching techniques since most logic gates have a small number of inputs (i.e. less than 6) [11].

In summary, current heuristic methods for logic synthesis provide satisfactory results for a specific function type but cannot produce near-optimal results also for the others, missing the opportunity to have smaller and faster ASICs.

In order to overcome these limitations, we present *MIXSyn*, a novel area-efficient logic synthesis methodology targeting mixed XOR-AND/OR dominated circuits. We propose (i) a two-step logic optimization that enables selective and distinct manipulation of AND/OR and XOR-intensive portions of the logic circuit. Intermediate *EXternal Don't Care* (EXDC) conditions are computed to improve the optimization quality. Stemming from work in [13], we propose (ii) an area-oriented library-free technology mapping method to take full advantage of complex gates with small computational effort. A subject-graph decomposition with an enhanced base-functions is used to natively support both AND/OR and XOR operations. We finally give efficient algorithms to assign and build on the fly complementary gates. Results show that *MIXSyn* for CMOS outperforms modern academic and commercial synthesis tools having 18.0% and 9.2% fewer devices on average over a set of MCNC benchmarks. In addition, we forecast the performance of our flow in the context of emerging ambipolar technologies. Such technology allows us to efficiently embed the XOR functionality in a unique double-gate device, leading to a very compact implementation of the XOR function. Results show that *MIXSyn* can exploit the ambipolar technology opportunity by giving average transistor-count improvements up to 20.9% and 15.3% compared to academic and commercial synthesis flows.

The remainder of this paper is organized as follows. Section II provides a background on logic synthesis for ASICs and introduces the notations used in the paper. In Section III, the proposed logic optimization and technology mapping methods are detailed. Then, in Section IV, experimental methods and results for *MIXSyn* are presented and compared with state-of-art commercial and academic synthesis tools. The interest of novel technologies with higher logic expressive power is also detailed. We finally conclude the paper in Section V.

II. Background and Notation

This section presents some background on logic optimization and technology mapping and introduces the notations used in the paper.

A. Notations

The support of the Boolean function f, denoted by $supp(f)$, is the set of all variables on which f depends. A Boolean network is a *Directed Acyclic Graph* (DAG) with nodes corresponding to logic functions and inputs/outputs directed edges corresponding to function inputs and output, respectively. We define *Primary Inputs* (PIs) as nodes without fanins in the current network and *Primary Outputs* (POs) as a predefined subset of nodes. The *level* of a node is the length of the longest path from any PI to the node. When the DAG is a rooted tree, we define the *reachable depth* of a node as the length of the longest path from any PI to the root passing through that node. We define a network with no more than k-inputs a k-feasible network. A network is k-bounded if each node is k-feasible. A subject-graph is a k-bounded DAG used for technology mapping. In a subject-graph, the set of all the nodes function type form the *base functions set*.

B. Logic Optimization

Boolean function logic optimization aims to reduce the size of the circuit, minimizing some general metrics such as gates, literals or nets count. Standard optimization methods target AND/OR-dominated logic functions. AND/OR optimization is usually carried out by algebraic factorization [3]–[5]. In order to support also XOR optimization, a method based on *Binary Decision Diagrams* (BDDs), named BDS, was proposed in [6]. BDS iteratively search for the most efficient decomposition among AND, OR, XOR and MUX. Despite BDS provides excellent results for XOR-intensive functions and achieves reasonable results for others, AND/OR-intensive functions still get more benefit from algebraic factorization methods. A version of BDS targeting LUTs has been proposed in [7], named BDS-pga, incorporating further decomposition schemes that generates area-minimal k-bounded networks. Other notable attempts to optimize XOR-intensive logic functions are based on functional linear decomposition [8] and ESOP form minimization [9].

C. Technology Mapping

Technology mapping onto standard cells consists of three main phases: 1) subject-graph construction, 2) matching, to recognize if a certain portion of the subject-graph can be implemented with a given cell and 3) selection, to choose the best matching cells to optimize some given metric. Traditionally, subject-graph construction consists of the decomposition of the Boolean network into a 2-bounded DAG having only NAND and INVs as base functions [10]. There are two main matching techniques: tree matching and Boolean matching. Tree matching [10] involves a Boolean tree isomorphism problem that can be solved efficiently. However, since the tree representation is not canonical, the tree matching can miss some potential matches. Instead, Boolean matching is comparing directly cell and sub-graph Boolean functions via tautology check therefore every possible matching is detected [11]. For the final selection phase, generic algorithms can be used, e.g. *Dynamic Programming* (DP) algorithms that are guaranteed to find an optimal solution in polynomial time complexity.

III. Proposed Logic Synthesis: *MIXSyn*

MIXSyn is an area-oriented logic synthesis methodology with novel logic optimization and technology mapping methods. Logic optimization is performed using a hybrid two step approach to identify and selectively manipulate AND/OR and XOR dominated portions of the logic circuit. After logic optimization, library-free technology mapping is employed to enhance design flexibility by building on the fly custom complex gates (e.g. XOR-based gates) at the transistor level. In the next subsections, we present in detail the proposed logic optimization and technology mapping methods.

A. Hybrid Logic Optimization

1) **Motivation***:* Over the past years, the problem of logic optimization has been approached by proposing efficient methods to deal with one of two major classes of functions that frequently appear in logic designs: AND/OR and XOR intensive functions. Most logic designs contain both of them intertwined in such a way that it is not possible to clearly split one part from the other. Hence, a single step logic optimization can minimize either the AND/OR or the XOR dominated portion of the circuit and produce suboptimal results for the other. An integrated optimization method that produces near-optimal results for both types of functions is presented hereafter.

2) **Algorithm***:* We propose to perform logic optimization in *MIXSyn* through a hybrid two step approach. The pseudocode for the hybrid logic optimization is shown in Algorithm 1. First, XOR-optimization is applied to the *Input Boolean Network* (IBN) to explicitly highlight XOR/XNOR nodes (Alg.1-$\boldsymbol{\alpha}$). Then, the *XOR Optimized Boolean Network* (XOBN) is split in two sub-networks: the *First Boolean Network* (FBN) having the evidenced XOR/XNOR nodes and the *Second Boolean Network* (SBN) with remaining nodes (Alg.1-$\boldsymbol{\beta}$). SBN has two types of inputs: 1) *Primary Outputs* (POs) from the FBN and 2) a subset of the *Primary Inputs* (PIs) of the IBN. If there is a non-empty set of *Common Inputs* (CIs) between the second type of SBN primary inputs and FBN primary inputs (i.e. if $supp(SBN \backslash \{POs_{FBN}\}) \cap supp(FBN) = CIs \neq \emptyset$) it is also possible to specify the input *Controllability Don't Care* set (CDC$_{in}$) for the SBN (Alg.1-$\boldsymbol{\chi}$). In particular, the $CDC_{in}(SBN)$ contains the *Common Inputs* (CIs) combinations that never occur. The SBN is AND/OR-dominated and hence an AND/OR-optimization method is applied (Alg.1-$\boldsymbol{\psi}$). Finally, the *AND/OR Optimized Boolean Network* (AOOBN) is merged with FBN resulting in the global *Optimized Boolean Network* (OBN) (Alg.1-$\boldsymbol{\omega}$).

Algorithm 1 Hybrid Logic Optimization

```
INPUT: Input Boolean Network ($IBN$)
OUTPUT: Optimized Boolean Network ($OBN$)
  $XOBN$=XOR-Optimization($IBN$)                              ($\boldsymbol{\alpha}$)
  split $XOBN$ in $FBN$ and $SBN$:                             ($\boldsymbol{\beta}$)
  $FBN \leftarrow XOBN$ evidenced XOR/XNOR nodes
  $SBN \leftarrow XOBN$ remaining nodes
  $CDC_{in}(SBN) \leftarrow \emptyset$
  if ($supp(SBN \backslash \{POs_{FBN}\}) \cap supp(FBN) \neq \emptyset$) then
      compute $CDC_{in}(SBN)$                                  ($\boldsymbol{\chi}$)
  end if
  $AOOBN$=AND/OR-Optimization($SBN$,CDC$_{in}$)                ($\boldsymbol{\psi}$)
  $OBN$=$AOOBN \cup FBN$                                       ($\boldsymbol{\omega}$)
```

3) **Example***:* For the sake of clarity, we report a simple example of hybrid logic optimization and compare it with standard AND/OR and XOR optimization methods alone. The objective function (f) in this example has the following SOP form: $f = ab + bc + \overline{a}\overline{b} + ca$. A single step XOR-optimization can reduce the objective function in $f = bc + (a \odot b) + ca$. Instead, a single step AND/OR-optimization can factorize c and obtain $f = ab + \overline{a}\overline{b} + c(a + b)$. The proposed hybrid optimization method in Alg.1 can further minimize the objective function as shown in Fig.1. First, the XOR-optimization step highlights $x = (a \odot b)$ in the XOBN ($\boldsymbol{\alpha}$). After this, the XOBN is split in the FBN and SBN ($\boldsymbol{\beta}$). The FBN consists of the node $x = (a \odot b)$ and its primary inputs a and b. Instead, the SBN comprises w, v and f nodes and a, b, c and x as primary inputs. Considered that $supp(x) \cap supp(SBN \backslash x) = \{a, b\} \cap \{a, b, c\} = \{a, b\}$, CIs is not empty and it is possible to compute CDC$_{in}$(SBN) ($\boldsymbol{\chi}$). The CIs combinations that never occur, CDC$_{in}$(SBN), are specified with $x = (a \odot b)$: CDC$_{in}(x, a, b)$=$\{101, 110, 011, 000\}$. The successive AND/OR-optimization is then able to reduce the SBN in $f = x + c$ ($\boldsymbol{\psi}$). Finally, the FBN (only x node with its PIs) is merged with the AOOBN resulting in the final OBN ($\boldsymbol{\omega}$), with $f_{OBN} = (a \odot b) + c$.

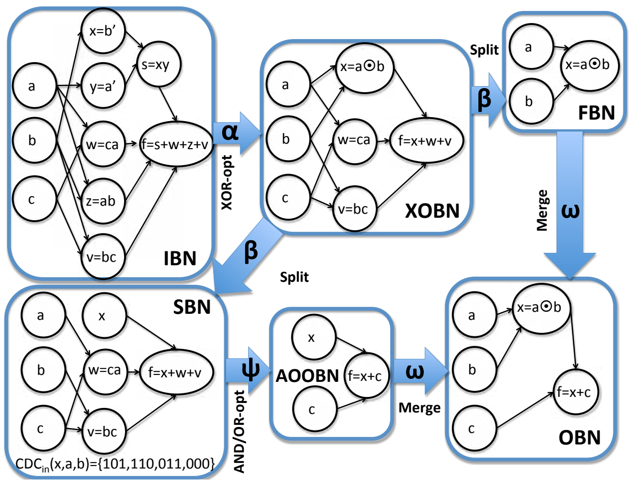

Fig. 1: Hybrid Optimization example with $f = ab + bc + \overline{a}\overline{b} + ca$.

B. Library-free Technology Mapping

1) **Motivation***:* The quality of the technology mapping operation mainly depends on the richness of the standard cell library [12] and on the ability of the mapping tool to recognize where such cells can be used (matching). Boolean matching can address the recognition issue with little computational burden since most cells have a small number of inputs (i.e. at most 5 or 6 inputs) [11]. On the other hand, the number of cells in the library, i.e. the library size, directly affects the complexity of technology mapping imposing a tradeoff between the computational cost and the design flexibility. To overcome this limitation, library-free technology mapping has been proposed in [13]–[16] where each cell/gate is built on the fly at the transistor level. Given a maximum gate fan-in m, library free technology mapping can implement all the 2^{2^m} possible functions[1] of m variables in a single gate. Boolean matching techniques are not needed here as a graph, or tree, covering method do the equivalent task. For the aforementioned reasons, we employ library-free technology mapping in *MIXSyn*.

2) **Algorithm***:* We perform technology mapping in *MIXSyn* using a library-free approach. Our algorithm consists of three main parts: *Pre-decomposition*, *Gate-assignment* and *Gate-building*. It differentiates from works in [13]–[16] in the *Pre-decomposition* and *Gate-assignment* methods that here are designed to support XORs. The pseudocode for technology mapping is shown in Algorithm 2.

Algorithm 2 Library-free Technology Mapping

```
INPUT: Optimized Boolean Network ($OBN$), max gate fan-in $m$
OUTPUT: Network of gates
  $G \leftarrow$ subject-graph-gecomposition($OBN$)          ($\alpha$)
  $F \leftarrow$ decompose $G$ in a forest of trees          ($\alpha$)
  remove internal INV nodes from $F$                         ($\alpha$)
  $IXI$-free-$F \leftarrow$ XOR-decompose $F$                ($\alpha$)
  $V \leftarrow \emptyset$
  for all tree $T$ in $IXI$-free-$F$ do
    $V \leftarrow$ Greedy-Tree-Decomposition($T$,$m$)       ($\beta$)
  end for
  replace internal INV nodes in $V$
  for all tree $S$ in $V$ do
    propagate internal INVs in $S$ to leaves                 ($\omega$)
    evaluate best gate polarity                              ($\omega$)
    build gate on the fly                                    ($\omega$)
  end for
  assign and build INVs                                      ($\omega$)
```

Pre-Decomposition (Alg.2-α): The first step of technology mapping aims to decompose the input *Optimized Boolean Network* (OBN) into a subject-graph with a limited base function set. Our *base function set* comprises 2-input AND/OR/XOR/XNOR and INV. In this way, it is possible to preserve XOR operations in the subject-graph that are otherwise hidden by standard NAND/INV decomposition [10]. Since we are targeting minimum area results and direct DAG covering for minimum area is known to be NP-complete [1], we decompose the subject-graph, that is a 2-bounded DAG, in a forest of trees. Each generated tree has a subset of *primary inputs* at the leaves and nodes representing a logic function from the *base function set*. Unfortunately, *internal* XOR/XNOR and INV nodes do not have an efficient physical realization such as series/parallel construction rules holding for *internal* AND/OR nodes. For this reason, each tree in the forest is further decomposed to have XOR/XNOR nodes only at the first level of the tree, after leaves. In addition, *internal* INV nodes are temporarily removed from the trees. Indeed, we take care of inverters influence during *gate-building*. At the end of *gate-assignment*, INV nodes are reinserted in their original position as each tree is only decomposed but not functionally transformed during these phases. The output of the *pre-decomposition* phase is an *Internal-XOR/INV-free* (IXI-free) forest of binary trees.

[1]Note that some of the 2^{2^m} functions have the same gate implementation due to input permutation. However, the number of different gates implementation with m inputs is still large.

Gate-Assignment (Alg.2-β): Once the *internal-XOR/INV-free* forest of trees has been created, a tree covering/decomposition method generates sub-trees with at most m inputs (maximum gate fan-in) in one-to-one correspondence with logic gates. In Section III-B3, we present a greedy decomposition to solve this task rather than the *Dynamic Programming* (DP) approach used in literature [13]. We will show that the proposed greedy algorithm produces optimal results in our context with a smaller runtime complexity than DP.

Gate-Building (Alg.2-ω): After the tree-decomposition phase, the logic function of each gate is defined by a Boolean rooted tree S having at most m leaves. Before building the gate, some transformations must be applied to S. First, internal inverters are propagated to the leaves to enable standard gate construction rules, e.g. series/parallel construction rules. Then, the best function polarity to be implemented is evaluated by counting the number of required inverters for both polarities. If the opposite function polarity is preferable for gate implementation, an inverter is put at the root of S and it is then propagated to the leaves. Later, each gate is built starting from S using standard transistor-level gate design techniques [18]. Finally, inverters are assigned and built on the fly where needed.

3) **Greedy Tree Decomposition***:* The task of the greedy tree decomposition algorithm is to generate Boolean binary sub-trees that have an efficient one-to-one correspondence with logic gates. Each generated tree must be m-feasible, i.e. it must have at most m inputs according to the maximum gate fan-in limitation. Moreover, the tree decomposition must result into a minimum area set of gates. To this end, an area cost function associated with each node is needed to drive the final decomposition. We use the number of leaves as indicator of the implementation complexity for each subtree to remain technology independent. Under this assumption, the tree decomposition task might be reformulated as: "decompose a binary-tree in m-feasible subtrees minimizing the number of leaves".

In Algorithm 3, a greedy method is proposed to solve the problem. The proposed technique starts with the original tree root as current node. It first checks if there exists a m-feasible subtree rooted at the current node. If it exists, the found subtree is detached from the original tree and the parent of the current node updates its *maximum reachable depth* (Alg.3-α). If the remaining tree is empty the procedure stops (Alg.3-ω), otherwise the decomposition algorithm is recursively called from the original root (Alg.3-β). Instead, if a m-feasible subtree rooted at the current node is not found, the algorithm is recalled from the *maximum reachable depth* children of the current node (Alg.3-χ). To prove the optimality of this greedy technique,

Algorithm 3 Binary-tree decomposition in m-feasible sub-trees minimizing the number of leaves

```
GLOBAL VARIABLE: Binary-Tree $T$
INPUT: Root $R$, $m$
OUTPUT: $m$-feasible Sub-Trees
FUNCTION: BynTreeDecomp($R$, $m$)
  if ($\exists$ $m$-feasible subtree rooted at $R$
  with only leaves in input) then
    detach the found subtree from $T$                 ($\alpha$)
    update $R$ parent reachable depth                 ($\alpha$)
    if ($T \neq \emptyset$) then
      BynTreeDecomp($T$ root, $m$)                     ($\beta$)
    else
      STOP                                             ($\omega$)
    end if
  else
    $V$=max reachable depth child of $R$              ($\chi$)
    BynTreeDecomp(V, $m$)                              ($\chi$)
  end if
```

we show that each substructure found is optimal with respect to the overall decomposition process. In other words, we show that there does not exist another subtree that reduces the overall number of leaves if considered in place of the one found by our algorithm. The following theorem states that this is our case.

Theorem: Algorithm 3 decomposes a binary-tree in m-feasible subtrees with the minimum overall number of leaves.

Proof: (Proof by Contradiction) Assume that there exists a sub-tree (T_B) more favorable with respect to the one found by the proposed algorithm (T_A). T_A is rooted at node D. All the sub-trees rooted at D, or at its descendants, are suboptimal by construction with respect to T_A. Then, we suppose that T_B is rooted at some D ancestor. In this scenario, T_B cannot have only leaves in input still respecting m-feasibility. Consequently, T_B generates at least one additional leaf, with respect to T_A, in the tree portion below D. For the tree portion above D, T_B saves at most the leaf induced by T_A at D parent node. Thus, it follows that using T_B in place of T_A is not reducing the overall number of leaves, contradicting our assumption. *Q.E.D.*

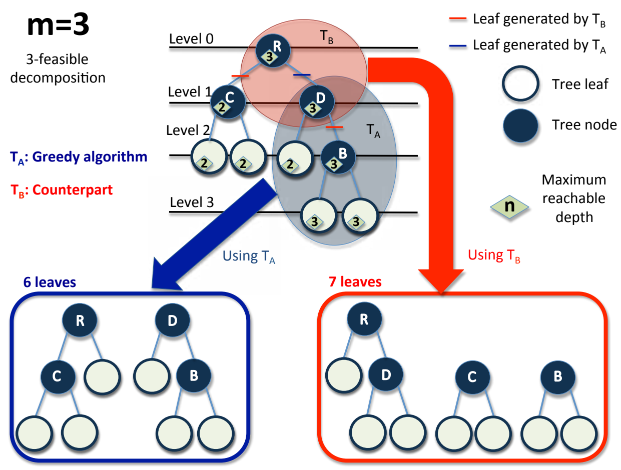

Fig. 2: Greedy tree decomposition example. T_A is the subtree generated by the greedy algorithm while T_B is the counterpart subtree.

Example: In Fig.2, an example is provided about the optimality of the greedy decomposition algorithm. In this case, T_B (counterpart sub-tree) is rooted at the original tree root R, that is the only ancestor node for D, the root of T_A (greedy decomposition sub-tree). As $m = 3$, T_B cannot have only leaves in input and still be 3-feasible. Therefore, T_B generates one additional leaf in the tree portion below node D (node B). On the other hand, T_B saves the leaf generated by T_A above node D. So far, T_B generates the same number of leaves with respect to T_A. Already at this point, it is possible to say that the subtree T_A is minimizing the overall number of leaves of the decomposed tree. However, T_B imposes also an additional leaf at R left child (C) while T_A does not. Thus, using T_B in place of T_A results in one additional leaf in the final decomposed tree.

Complexity: The run-time complexity of the greedy algorithm is $O(|T|^2/m)$, where $|T|$ is the number of nodes in the tree. This approach is faster than the dynamic programming method proposed in [13]: their complexity was $O(|T|^2)$ and did not scale with m. Moreover, the greedy nature of our algorithm makes the constant factor hidden in the O-notation smaller than the dynamic programming one, which suffers from postorder and preorder traversal of tree [17].

IV. Experimental Results

In this section, we present experimental results for our proposed area-oriented synthesis methodology: *MIXSyn*. We consider two target technologies: standard CMOS and ambipolar technology. Area results (expressed as transistors count) from state-of-art academic (ABC, BDS) and commercial (Synopsys) synthesis tools are compared with *MIXSyn* for both target technologies.

A. Target Technologies

We consider two different target transistor technologies for *MIXSyn*: the standard *Complementary Metal Oxide Semiconductor* (CMOS) and the novel ambipolar *Field Effect Transistors* (FETs).

1) **CMOS:** CMOS is the standard technology for constructing integrated circuits by means of symmetrical pairs of MOSFETs. MOSFETs are unipolar transistors whose n-type or p-type behavior is determined during fabrication. While traditional standard cell mappers use predefined libraries of logic gates, we build on the fly complementary gates during library-free technology mapping in *MIXSyn*. To accomplish this task in CMOS technology, we employ transistor-level design methods described in [18].

2) **Ambipolar FETs:** Ambipolar FETs are *Double-Independent-Gate* (DIG) devices whose polarity can be switched on-line applying a specific binary voltage on the additional gate, usually called *Polarity Gate* (PG) [19]. The *Conventional Gate* (CG) instead controls the ambipolar transistor's on-state as in usual MOSFETs. Fig.3(a) summarizes the on-line reconfiguration of the Ambipolar FETs polarity. Ambipolar FETs on-state is logical biconditional on

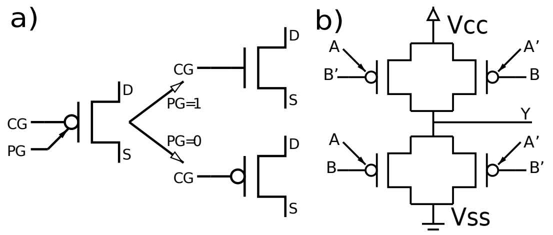

Fig. 3: (a) Ambipolar FETs polarity control (b) XOR-2 gate in [19].

both gates values. For this reason, ambipolar transistors enable a compact realization of the XOR function, such as the XOR-2 gate depicted in Fig.3(b). *Pull-Up* (PU) and *Pull-Down* (PD) networks of the XOR-2 gate in Fig.3(b) are good at passing both V_{cc} and V_{ss}: this gives the opportunity to implement also efficient *Transmission Gate* (TG) logic elements with ambipolar transistors [20]. Design methods for logic gates construction in [18], updated accordingly to [19] for ambipolar FETs, are employed to build on the fly complementary ambipolar gates during *MIXSyn* library-free technology mapping.

B. Methodology

MIXSyn is compared to ABC, BDS and Synopsys Design Compiler (DC) logic synthesis flows over a heterogeneous set of benchmarks taken from the MCNC suite.

1) **MIXSyn:** Our proposed methods are implemented in C language. Interaction with external optimization tools is done via Perl scripts. In the mixed logic optimization phase, AND/OR-optimization is performed with ABC [5] while XOR-optimization is done by BDS-pga [7][2]. The current *MIXSyn* implementation does not include the CDC computation described in Alg.1-χ, as ABC does not properly support extensive EXDC [21]. On the technology mapping perspective, the subject-graph construction task is already carried out during the optimization phase. Indeed, BDS-pga can produce 2-bounded networks and AIG from ABC are compatible with our base function-set. MOSFETs or ambipolar FETs are finally used as basic components to build on the fly complementary gates with a maximum fan-in $m = 4$. In ambipolar technology, the opportunity to

[2]Note that since BDS-pga is based on BDDs (and uses canonical global ROBDDs where possible), the input circuit partition/representation does not preclude to iterate the hybrid optimization to improve the synthesis quality.

have full-swing complementary TG gates is particularly interesting [20]. For this reason, a different version of *MIXSyn* targeting mixed static/TG ambipolar logic gates (*MIXSyn* TG/ST) is also evaluated.

2) **Reference Flows:** ABC, BDS and Design Compiler synthesis tools are not library free and therefore need in input a reference library. We have built a reference library with INV, XOR-2, XNOR-2, NAND-{2,3,4}, NOR-{2,3,4}, AOI/OAI and generalized AOI/AOI [19] standard cells. As we are targeting low area results, no timing information is provided in the reference library to avoid area/timing trade-off. The given area information is the transistor count for the implementation of each cell in static complementary style. Defaults and options for the reference flows are:

- ABC: ABC *resyn2* optimization script and ABC mapper.
- BDS: BDS logic optimization and ABC mapper.
- DC: Synopsys Design Compiler *compile -area_effort high*.

C. Results and Discussion

Area results, expressed in transistors count (T), are presented in Table I for both target technologies. *MIXSyn* is referred in the Table I as *MX* for brevity. Comparison with reference flows is proposed over two set of benchmarks: XOR-intensive benchmarks (reduced-set) and mixed XOR-AND/OR-intensive benchmarks (full-set). To distinguish the reduced set from the full-set, the *XOR-Intensiveness* (XI) of each benchmark is evaluated as follows. BDS-pga is employed to create a 2-bounded network. Then, the XI is the ratio between the number of XOR/XNOR nodes and the total number of nodes in the network. We refer to XOR intensive benchmarks as the ones having XI>0.1.

1) **CMOS Technology:** Results for CMOS technology are reported in the right part of Table I. Positive transistors count improvements are highlighted in bold.

Full Set of Benchmarks, Table I-(a): *MIXSyn* is the best synthesis flow for CMOS technology having on the average 4931 devices. The reason behind is that *MIXSyn* is designed to minimize the AND/OR and XOR dominated portions of the logic circuit with an appropriate optimization method for each case. Moreover, the library-free mapping phase in *MIXSyn* takes full-advantage of all the possible logic gates realization with $m = 4$ inputs. DC is the second best CMOS synthesis flow having 10.2% more devices on average than *MIXSyn*. BDS and ABC have an equal inferior performance compared to the others, producing about 22.0% more devices than *MIXSyn*.

Reduced Set of Benchmarks, Table I-(b): Considering only XOR-intensive benchmarks, the trend remains the same. Sorted by increasing average transistor count, the performances are: *MIXSyn* (4129 devices), DC (-1.1% w.r.t. *MIXSyn*), BDS (-15.0% w.r.t. *MIXSyn*) and ABC (-20.1% w.r.t. *MIXSyn*). The advantage of the hybrid logic optimization in *MIXSyn* is less marked for XOR-intensive benchmarks as they can be efficiently minimized by a single step optimization method. However, the enhanced design flexibility of the *MIXSyn* library-free mapping still produces the best result.

Average Number of Devices per Gate: The average number of transistors per logic gate (D), indicates that CMOS synthesis flows are producing logic gates of equivalent size (in terms of transistor count). Therefore, *MIXSyn* generates circuits with commensurable average gate fan-in, series transistors stack and circuit depth with respect to DC, BDS and ABC synthesis flows. Note that this result also holds for ambipolar technology.

2) **Ambipolar Technology:** Results for ambipolar technology are reported in the left part of Table I and positive transistor-count improvements are highlighted in bold.

Full Set of Benchmarks, Table I-(a): *MIXSyn TG/ST* is the best synthesis flow for ambipolar technology with 4204 devices on the average. *MIXSyn TG/ST* exploits the efficient mixed static/TG style opportunity offered by ambipolar transistors (e.g. XOR-2 in Fig.3(b) or XOR-3 in [20]). The static version of ambipolar *MIXSyn* have 4.6% more devices on the average than *MIXSyn TG/ST*. Ambipolar DC requires 18.1% more devices on average than *MIXSyn TG/ST*.

Reduced Set of Benchmarks, Table I-(b): For XOR-intensive benchmarks, *MIXSyn TG/ST* is again the best ambipolar synthesis flow, with 3136 devices on average. The static version of *MIXSyn* and DC require 9.4% and 14.9% more devices on average than *MIXSyn TG/ST*. In ambipolar technology, the advantage of *MIXSyn* methodology is appreciable even for the reduced set of benchmarks. Indeed, *MIXSyn* subject-graph decomposition natively supports the XOR nodes highlighted during the XOR-optimization. Such XOR operations have an efficient implementation with ambipolar devices.

3) **Ambipolar vs. CMOS Technologies:** Comparison between ambipolar and CMOS synthesis flows are reported in Table I in red.

Full Set of Benchmarks, Table I-(a): *MIXSyn TG/ST* is the most suited synthesis flow to take advantage of ambipolar technology. Indeed, it can reduce by 14.7% the average transistor count of the best CMOS flow (*MIXSyn*). The static version of ambipolar *MIXSyn* can reduce by 10.8% the average number of devices of CMOS *MIXSyn*. Instead, traditional flows used with ambipolar technology can reduce the average transistor count over their CMOS counterparts (8.6%, 11.1% and 12.4% respectively), but fail to do the same over CMOS *MIXSyn* (-0.7%, -8.4% and -7.7% respectively). This is because the improvements of *MIXSyn* over DC, BDS and ABC with CMOS technology are larger than the ones carried by the ambipolar technology for the same reference flows.

Reduced Set of Benchmarks, Table I-(b): XOR-intensive benchmarks derive the greatest benefit from ambipolar technology [19]. For these benchmarks, *MIXSyn TG/ST* can boost up to 24.0% the average transistor count reduction over CMOS *MIXSyn*. In addition, also ambipolar DC, BDS and ABC flows are now able to reduce the average number of devices with respect to *MIXSyn* (12.7%, 4.4% and 4.1% respectively) thanks to the high XI of the considered benchmarks, which appears favorable with ambipolar devices [19].

Average Number of Devices per Gate: Averaged over the full (reduced) set of benchmarks, ambipolar synthesis flows have about 15.0% (20.0%) less devices per logic gate compared to their CMOS counterparts. This is thanks to the compact realization of the XOR function in ambipolar technology. This result confirms that circuits in ambipolar technology are potentially denser than in CMOS.

4) **Discussion:** *MIXSyn* produces the best results, in terms of transistor count, compared to DC, BDS and ABC synthesis tools for both CMOS and ambipolar technologies. The advantage of *MIXSyn* over the reference flows comes from three major facts. First, the logic optimization method in *MIXSyn* is dependent on the circuit type and not fixed *a priori* as in the reference flows. Second, the subject-graph decomposition with enhanced base functions set allows *MIXSyn* to easily recognize and map XOR functions, otherwise hidden by conventional NAND/INV decomposition [10]. Third, the library-free technology mapping approach permits to have a *virtual library* with 2^{2^m} different logic functions, a number not practical with standard-cells already for small values of m (4 or 5) [22]. We expect that the *MIXSyn* advantage will become even more relevant when CDC computation will be included. Although timing results are not considered and presented in this paper, note that the library free mapping technique can be adapted to obtain also delay efficient results [13]. In this case, the delay estimation accuracy is supported by the continuous advance in dynamic cell characterization. Even though it is out of the scope of this work, note that library-free mapping implies a subsequent physical design that is different from the one employed after traditional standard-cell based technology mappers. The physical synthesis must be carried out at the transistor level and the interconnection impact on the circuit becomes more relevant [22]. The transistor-level physical synthesis can be accomplished by circuit

TABLE I: Experimental results for *MIXSyn* and comparison with reference synthesis flows

Legend: T is the number of transistors, D is the average number of devices per gate/cell

Technology	Ambipolar										CMOS							
Flow	**MX TG/ST**		MX		DC		BDS		ABC		**MX**		DC		BDS		ABC	
Bench. (XI)	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D
C1355 (0.46)	904	6.27	1042	6.23	1332	5.74	1432	6.50	1524	7.71	1521	9.09	1606	8.23	1972	8.96	2120	10.81
C3540 (0.04)	3510	5.01	3580	5.08	3952	3.95	4430	4.07	4378	3.76	3723	5.13	4096	4.12	4414	4.05	4512	3.83
C6288 (0.19)	7554	4.13	8436	4.21	9038	4.54	9684	4.29	10120	5.60	10038	5.01	10146	5.48	11864	5.25	13454	7.43
C7552 (0.18)	5692	4.21	5956	4.25	5884	4.70	6532	4.48	5900	4.09	7028	5.58	6906	5.60	7472	5.13	6630	4.60
des (0.03)	9650	5.44	9650	5.44	12484	4.07	12966	4.20	12990	3.95	9940	5.60	12940	4.19	13878	4.48	13534	4.11
C1908 (0.29)	1174	3.70	1278	3.75	1218	5.13	1440	5.19	1524	6.12	1515	4.65	1542	6.31	1650	6.62	1710	6.17
pair (0.07)	4796	5.27	4796	5.27	5266	4.06	5656	4.07	5332	4.01	5131	5.63	5556	4.40	6088	4.58	5660	4.24
my_adder (0.22)	352	4.40	448	4.07	544	4.85	632	5.01	722	4.86	547	4.96	672	7.00	790	6.22	880	7.16

Table I-(a)	**Full set of Benchmarks**																	
Average	**4204**	4.80	4398	4.79	4965	4.63	5346	4.72	5312	5.01	**4931**	5.70	5433	5.67	6016	5.66	6063	6.04
	Ambipolar										CMOS							
Improv. → vs. ↓	MX TG/ST		MX		DC		BDS		ABC		MX		DC		BDS		ABC	
Amb. MX TG/ST	-	-	-4.6%	0.3%	-18.1%	3.6%	-27.2%	1.6%	-26.3%	-4.3%	-17.3%	-18.7%	-29.2%	-18.1%	-43.1%	-17.9%	-44.2%	-25.8%
Amb. MX ST	**4.4%**	-0.2%	-	-	-12.9%	3.3%	-21.5%	1.4%	-20.8%	-4.6%	-12.1%	-19.0%	-23.5%	-18.3%	-36.8%	-18.1%	-37.8%	-26.1%
Amb. DC	**15.3%**	-3.7%	**11.4%**	-3.4%	-	-	-7.7%	-1.9%	-7.0%	-8.2%	**0.7%**	-23.1%	-9.4%	-22.4%	-21.1%	-22.2%	-22.1%	-30.4%
Amb. BDS	**21.3%**	-1.7%	**17.7%**	-1.5%	**7.1%**	4.0%	-	-	**0.6%**	-6.1%	**7.7%**	-20.7%	-1.6%	-20.1%	-12.5%	-19.9%	-13.4%	-28.0%
Amb. ABC	**20.9%**	4.2%	**17.2%**	4.4%	**6.5%**	7.6%	-0.6%	5.8%	-	-	**7.2%**	-13.8%	-2.3%	-13.2%	-13.2%	-13.0%	-14.1%	-20.6%
CMOS MX ST	14.7%	15.7%	10.8%	16.0%	**-0.7%**	18.7%	**-8.4%**	17.2%	**-7.7%**	12.1%	-	-	-10.2%	0.5%	-22.0%	0.7%	-23.0%	-5.9%
CMOS DC	22.6%	15.3%	19.1%	15.5%	8.6%	18.3%	1.6%	16.7%	2.2%	11.6%	**9.2%**	-0.5%	-	-	-10.7%	0.1%	-11.6%	-6.5%
CMOS BDS	30.1%	15.2%	26.9%	15.3%	17.4%	18.2%	11.1%	16.6%	11.7%	11.5%	**18.0%**	-0.7%	**9.7%**	-0.2%	-	-	-0.8%	-6.7%
CMOS ABC	30.6%	20.5%	27.5%	20.7%	18.1%	23.3%	11.8%	21.9%	12.4%	17.0%	**18.7%**	5.6%	**10.4%**	6.1%	**0.8%**	6.3%	-	-

Table I-(b)	**Reduced set of Benchmarks (XI>0.1)**																	
Average	**3136**	4.54	3432	4.50	3603	4.99	3944	5.09	3958	5.67	**4129**	5.85	4174	6.52	4749	6.43	4959	7.23
	Ambipolar										CMOS							
Improv. → vs. ↓	MX TG/ST		MX		DC		BDS		ABC		MX		DC		BDS		ABC	
Amb. MX TG/ST	-	-	-9.4%	0.9%	-14.9%	-9.9%	-25.7%	-12.1%	-26.2%	-24.9%	-31.6%	-28.8%	-33.1%	-43.6%	-51.4%	-41.6%	-58.1%	-59.2%
Amb. MX	**8.6%**	0.9%	-	-	-5.0%	-10.9%	-14.9%	-13.1%	-15.3%	-26.0%	-20.3%	-30.0%	-21.6%	-44.8%	-38.3%	-42.9%	-44.5%	-60.6%
Amb. DC	**12.9%**	9.0%	**4.7%**	9.8%	-	-	-9.4%	-2.0%	-9.8%	-13.6%	-14.6%	-17.2%	-15.8%	-30.6%	-31.8%	-28.8%	-37.4%	-44.9%
Amb. BDS	**20.5%**	10.8%	**13.0%**	11.6%	**8.6%**	1.9%	-	-	-0.3%	-11.4%	-4.7%	-14.9%	-5.8%	-28.1%	-20.4%	-26.3%	-25.7%	-42.0%
Amb. ABC	**20.8%**	19.9%	**13.3%**	20.6%	**9.0%**	12.0%	**0.4%**	10.2%	-	-	-4.3%	-3.2%	-5.4%	-15.0%	-20.0%	-13.4%	-25.3%	-27.5%
CMOS MX	24.0%	22.4%	16.9%	23.0%	12.7%	14.7%	4.4%	13.0%	4.1%	3.0%	-	-	-1.1%	-11.4%	-15.0%	-9.9%	-20.1%	-23.6%
CMOS DC	24.8%	30.3%	17.7%	30.9%	13.7%	23.5%	5.5%	21.9%	5.2%	13.0%	**1.1%**	10.3%	-	-	-13.7%	1.4%	-18.8%	-10.9%
CMOS BDS	33.9%	29.3%	27.7%	30.0%	24.1%	22.4%	16.9%	20.8%	16.6%	11.8%	**13.0%**	9.0%	**12.1%**	-1.4%	-	-	-4.4%	-12.4%
CMOS ABC	36.8%	37.2%	30.8%	37.7%	27.3%	31.0%	20.4%	29.6%	20.2%	21.5%	**16.7%**	19.0%	**15.8%**	9.8%	**4.2%**	11.0%	-	-

to layout automated tools or using a gate-array approach, as recently proposed for ambipolar technology [23].

V. Conclusions

We propose a novel logic synthesis methodology, *MIXSyn*, capable to produce area-efficient results for mixed XOR-AND/OR dominated circuits. *MIXSyn* is designed to produce state-of-art best results for the two most frequent types of functions that appear in logic circuits: AND/OR and XOR intensive functions. The key concepts that enable this opportunity are selective, and distinct, optimization of AND/OR and XOR-intensive portions of the logic circuit followed by library-free technology mapping. In terms of transistor count, experimental results show that *MIXSyn* outperforms best academic and commercial synthesis tools for CMOS technology by 18.0% and 9.2% on the average. With an ambipolar technology, *MIXSyn* produces circuits having on average 20.9% and 15.3% fewer devices with respect to best academic and commercial synthesis tools, respectively.

References

[1] K. Keutzer and D. Richards, *Computational complexity of logic synthesis and optimization*. Proc. IWLS, May 1989.

[2] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[3] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A.R. Wang, *MIS: A Multiple-Level Logic Optimization System*, IEEE Trans. CAD, vol. 6, pp. 1062-1081, Nov.1987.

[4] E. Sentovich et al., *SIS: A System for Sequential Circuit Synthesis*, ERL, Dept. EECS, Univ. California, Berkeley, UCB/ERL M92/41, 1992.

[5] ABC Logic Synthesis Tool [Online]. Available: http://www.eecs.berkeley.edu/alanmi/abc/

[6] C. Yang and M. Ciesielski, *BDS: A BDD-Based Logic Optimization System*, IEEE Trans. CAD, vol. 21, pp. 866-876, July 2002.

[7] N. Vemuri, P. Kalla and R. Tessier, *BDD-based Logic Synthesis for LUT-based FPGAs*, ACM Trans. TODAES, Vol.7, pp. 501-525, Oct. 2002.

[8] T.S. Czajkowski, S.D. Brown, *Functionally Linear Decomposition and Synthesis for FPGAs*, IEEE Trans. CAD, 27(12): 2236-2249, 2008

[9] N. Song, M. Perkowski, *Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Function*, IEEE Trans. CAD, 15(4): 385-395, 1996.

[10] K. Keutzer, *DAGON: technology binding and local optimization by DAG matching*, Proc. DAC, pp. 341-347, 1987.

[11] L. Benini and G. De Micheli, *A survey of Boolean matching techniques for library binding*, ACM TODAES, Vol. 2, No. 3, pp.193-226, July 1997.

[12] K. Keutzer, K. Kolwicz, and M. Lega, *Impact of library size on the quality of automated synthesis*, Proc. of ICCAD, pp. 120-123, 1987.

[13] Y. Jiang, S.S. Sapatnekar and C. Bamji, *Technology Mapping for High-Performance Static CMOS and Pass Transistor Logic Designs*, IEEE Trans. VLSI, vol. 9, pp. 577-589, Oct. 2001.

[14] M. Pullerits and A. Kabbani, *Library-free synthesis for area-delay minimization*, International Conference on Microelectronics, 2008.

[15] F.S. Marques et. al, *DAG Based Library-Free Technology Mapping*, Proc. GLSVLSI, pp 293-298, 2007

[16] J. Xue, D. Al-Khalili and C.N. Rozon, *Technology Mapping in Library-free Logic Synthesis*, Proc. SPIE 5837, 919 (2005).

[17] T.H. Cormen et.al, *Introduction To Algorithms*, MIT Press, 2009.

[18] J.M. Rabaey et al., *Digital Integrated Circuits*, Prentice Hall, 2003

[19] M.H. Ben-Jamaa, K Mohanram and G. De Micheli, *An Efficient Gate Library for Ambipolar CNTFET Logic*, IEEE Trans. CAD, vol. 30, pp.242-255, Feb. 2011.

[20] A. Zukoski, X. Yang and K. Mohanram, *Universal logic modules based on DG carbon nanotube transistors*, Proc. DAC, pp.884-889, 2011.

[21] K.Chang et al., *Logic Synthesis and Circuit Customization Using Extensive External Don't-Cares*, ACM Trans. Computational Logic, Jan. 2010.

[22] R. Reis, *Physical Design Automation at Transistor Level*, Proc. NORCHIP, pp.241-245, 2008.

[23] S. Bobba et al., *Physical synthesis onto a Sea-of-Tiles with double-gate silicon nanowire transistors*, Proc. DAC, pp. 42-47, 2012.