Exploring FPGA Logic Block Architecture for Reduced Configuration Memory

Fasahat HUSSAIN[1], Muhammad Mazher IQBAL[1], Husain PARVEZ[1], Muhammad RASHID[2]
[1]*Karachi Institute of Economics and Technology, 75190, Pakistan*
[2]*Umm Al Qura University, Saudi Arabia*
mazher.iqbal@kiet.edu.pk

***Abstract*—The reduction of reconfiguration delay, during the partial dynamic reconfiguration of FPGAs, is important. In this context, the bitstream compression technique is one of the widely used techniques. These compression techniques only minimize the size of the bitstream whereas the actual configuration memory size on FPGA remains the same, which consumes area as well as power. Therefore, alternative techniques are required to decrease area and power consumption along with the reconfiguration delays. This work optimizes the configuration memory requirements in the Configurable Logic Block (CLB) of FPGA with SRAM table sharing technique. The SRAM table of a Look-Up-Table (LUT) is shared with one or more LUTs in the same CLB by employing Negation-Permutation-Negation (NPN) classification. Furthermore, the relevant CAD tools are modified to explore the heterogeneous degree of SRAM table sharing within a CLB. For validation, extensive explorations are performed on the 20 largest MCNC benchmark circuits. It has been found that the configuration memory requirements of LUTs are reduced by 30% while retaining the same area, occupancy, and delay. Moreover, it can be further reduced by 50% provided that the FPGA occupancy is allowed to increase by only 15% while retaining the same delay.**

***Index Terms*—clustering algorithms, field programmable gate arrays, programmable logic arrays, reconfigurable architectures, reconfigurable logic.**

I. INTRODUCTION

Modern Field Programmable Gate Arrays (FPGA) architectures have considerably evolved after they were introduced almost four decades ago [1]. The internal architecture of an FPGA is composed of two major configurable portions: logic blocks and routing resources. The logic blocks are configured to map the logic functionality of a given circuit, while the configurable routing resources establish some appropriate connections between the logic functionality. The basic logic element of a FPGA is still a Static Random Access Memory (SRAM) based Look Up Table (LUT). However coarse-grained blocks (such as memories, adders with carry chains, multipliers and advanced routing architectures) have increased the overall FPGA's complexity [2].

For example, the commercial logic elements are modified to improve the efficiency of complex arithmetic functions by using hard adders, carry chains and fracture-able LUTs, along with the corresponding modifications and restrictions imposed on the CAD (Computer-Aided Design) tool algorithms [3]. Moreover, there are specialized logic block design efforts to optimize application-specific domains such as deep learning inference [4], low precision multiply and accumulate operations [5] and self-repairing hard arithmetic blocks [6]. Furthermore, multiplexers are added to allow the creation of wider functions.

In addition to the efficiency of complex arithmetic functions and specialized logic block design, LUTs can also act as small storage elements such as bit memories [7]. The LUT architectures have evolved to support faster arithmetic operations and now include carry logic to support cascaded addition. The flip flop, which is always an integral part of a BLE (Basic Logic Element), now allows connectivity from external inputs, internal carry or the LUT itself.

Similarly, other integral parts of FPGA architecture are memory blocks, digital signal processing and multiplier blocks with flexible connections. Additionally, the dynamic partial reconfiguration is now a standard feature of modern FPGAs that allows a run-time reconfiguration of partial bitstream on selected regions of FPGA [8].

SRAM-based FPGAs are widely used due to their flexibility, speed and ease of fabrication in Complementary Metal Oxide Semiconductor (CMOS) process technology [9]. Despite the increasing complexity of logic blocks in SRAM-based FPGA architecture; it has been shown that they occupy only 10-20% of the total FPGA area, whereas almost 80-90% of the area is occupied by the configurable routing resources [3]. Therefore, the reduction of reconfiguration overheads in SRAM-based FPGAs is an interesting research problem and is being explored extensively [10]. Moreover, the adoption of non-volatile memory such as Flash memory or Anti-Fuse [11] in FPGAs is another direction for the practitioners and researchers of this domain [12-13].

A. Related Work

Since the inception of FPGAs in 1985, the quest to find the most efficient logic element is extensively explored. Traditionally, the LUT sizes for the best area-delay product are 4-6 with 4 better for the area and 6 appropriate for delay [11], [14]. More recently, it has been shown that the performance of FPGAs can be improved through the adoption of a 7-input logic block that is constructed using two 4-input logic blocks [15]. There have been several other contributions to improve the area, delay and configuration memory requirements of logic modules [16–19] which are briefly described below:

The COGRE logic block [16] is a compactly organized gate-based (AND, OR and NOT) reconfigurable element that can map large portions of logic functions. It is generated by using frequently used NPN equivalence class functions in

Digital Object Identifier 10.4316/AECE.2022.03002

a benchmark circuit. The results for the 6-input COGRE cell, when compared with 6-LUT, reveal a reduction of 46% area and 32% configuration memory cells at the expense of a 7% increase in the delay. The major drawback of this approach is that the COGRE cell, optimized for one set of benchmark circuits, might not perform better for other benchmark circuits.

To address the flexibility issues, observed in COGRE logic blocks, the And-Invert-Cones [17] provide a better compromise between programming flexibility, area, delay as well as a total number of inputs and outputs. It is composed of And gates and Not gates which are grouped in the form of cone-like sub-graphs. Furthermore, the corresponding synthesis and technology mapping algorithms are modified for application mapping. Experimental results in [17] reveal a 22-32% reduction in delay and 16% reduction in area. Another flexible solution, known as the Scalable Logic Module (SLM) [18], uses Shannon expansion to break down a k-input function into a smaller partial (k-1) functions. It employs a smaller-sized LUT, along with some additional logic, to improve the area and configuration memory requirements. Experimentation in [18] shows a reduction of 20% and 33% in area and configuration memory cells, respectively, with an increase of 12% in the critical path.

In addition to the flexible and scalable approaches of [17] and [18], the NPN equivalence class approach is presented in [19–22]. The core idea in NPN equivalence is to share SRAM memory tables across LUTs in CLBs. The NPN classification describes a feature of synthesis, where the k-input combinatorial Boolean functions are implemented using the same circuit with few modifications provided that they share the same class. By using the same class functions, the SRAM tables are shared across 'M' such LUTs where 'M' is the degree of sharing. The degree of sharing determines how many LUTs share a single SRAM table within a single CLB. Synthesized Boolean functions, targeted for the k-input LUT having the same NPN class, are mapped on LUTs sharing the same SRAM tables. Additional storage bits are needed for the negation of inputs and output; however, the overall SRAM memory cells are reduced.

The SRAM table sharing technique is initially proposed for LUT-4 in a CLB of FPGAs [19]. Later, the same idea is extended to work with higher LUT inputs such as LUT 4-7 in CLB architectures with 2-4 LUTs sharing a single SRAM table [20-22]. Moreover, the area gains of 6-7% are reported with no effect on delay. Furthermore, application-specific FPGA architectures have also shown some reduction in the area after deploying the same idea of SRAM table sharing [23]. Such as, application-specific inflexible FPGA (ASIF [24]) is an application-specific FPGA architecture employs SRAM Table sharing technique in its logic blocks for the reduction of the total area and reconfiguration time. ASIF's other variants are proposed in [25-29].

B. Research Gap and Contribution

Despite the significant contributions of previous works on SRAM table sharing using the NPN equivalence class approach [19-22], three aspects remain unexplored. These three aspects may reveal a further reduction in area and configuration memory requirements. The first major aspect that needs to be explored in SRAM table sharing is to analyze the effect of a higher degree of sharing in a CLB. It implies that it is important to investigate the maximum number of LUTs in a single CLB that can share a single SRAM table without degrading the area and delay parameters. Secondly, the effects of a heterogeneous degree of sharing are required to be analyzed. It implies that a single CLB may have multiple SRAM tables and may have a varying degree of sharing. Finally, investigating the impact of SRAM table sharing on reducing the number of configuration cells is critical. No previous work on the SRAM table sharing technique has explored these three aspects. Therefore, a more extensive exploration along with the modification of CAD tools is required to achieve the above-mentioned objectives.

The key technical contributions of this work include the following:

- **Explore CLB architecture with a higher and heterogeneous degree of SRAM table sharing:** An SRAM memory table is explored to be shared between 2-16 LUTs in the same CLB (i.e., a higher degree of sharing). Similarly, a CLB may consist of multiple shared SRAM tables having a different degree of sharing (i.e., heterogeneous degree of sharing).
- **Modification of CAD tools to support a higher and heterogeneous degree of SRAM table sharing:** Changes in the clustering algorithms are proposed to support applications mapping on the new architecture.
- **Exploration for reduced SRAM memory requirements:** Experiments are performed to explore CLB architectures with reduced SRAM memory requirements with or without compromise on area and delay.

The aforementioned contributions are achieved by representing the proposed CLB architectures in VTR (Verilog-To-Routing) [30] and the corresponding modifications are performed in associated CAD algorithms. The validation is performed with MCNC (Microelectronics Center of North Carolina) benchmark suites [31], which are frequently employed for the evaluation of the newly proposed architectures. Consequently, the results are explored for the area, delay and configuration memory requirements of LUTs. The exploration has been performed in two different ways: (1) compromising the area and delay parameters, and (2) without compromising the area and delay parameters It has been concluded that CLBs with both higher and heterogeneous degrees of sharing have shown best-compromised results. It implies that with a slight compromise in area, the configuration memory requirements can be significantly reduced. This reduction in configuration memory can significantly reduce the reconfiguration times of the FPGA. Our current work focuses only on the SRAM-table sharing in logic blocks without carry chain logic. In our future work, we will consider testing this approach on LUTs with carry chain logic. The core methodology and technique, however, will not change. But, we need to investigate how the carry chain logic affects the remaining netlist's SRAM table sharing.

The rest of this article is organized as follows: Section II

provides the necessary background information about FPGA architectures, the CAD toolchain that is used to program the FPGAs, and the NPN classification technique which is the basis for SRAM table sharing. Section III presents the proposed modifications in CLB architecture and the corresponding CAD tools. Section IV presents the experimental setup and obtained results. Finally, the article is concluded in Section V.

II. Background

This section presents the fundamentals of internal FPGA Architecture in Section II (A). Subsequently, an appropriate background on the VTR CAD tool and NPN classification technique is presented in Section II (B) and Section II (C), respectively.

A. FPGA Architecture

The internal architecture of FPGA is composed of two main portions, 1) configurable logic blocks and 2) configurable routing resources. A mesh-based FPGA is composed of a two-dimensional array of logic blocks that are connected through configurable routing resources. The logic blocks can be configured to map the logic functionality of a given circuit. Similarly, the configurable routing resources create connections between the logic functionality using uni-directional, single driver routing channels. A CLB comprises a single basic logic element (BLE) or multiple BLEs connected through a programmable cross-bar. A BLE consists of a K-LUT, a Flip-Flop (FF) and a Multiplexer (MUX). A LUT is constructed with multiplexers and memory cells. The number of memory cells in each LUT is related to the number of inputs of a LUT. For example, a K-input LUT (LUT-K) has 2^k memory cells and can implement the truth table of the K-variables Boolean function.

B. VTR Design Tool

The VTR CAD tool [30] represents different FPGA architectures in the form of data structures. It is used to transform an application circuit from a high-level register transfer level (RTL) definition to a connected list of CLBs/IOs which is then mapped onto an FPGA architecture representation such that the area, delay and power results are reported to evaluate the effectiveness of the FPGA architecture. As a first step, the ODIN tool [32] synthesizes a high-level Verilog description to a connected list of components, belonging to a standard cell library, which includes the definitions of basic logic gates and flip-flops. Subsequently, the technology mapping tool (named ABC [33]) converts the netlist of standard cells to a connected list of LUTs and flip-flops. Similarly, the packing algorithm (TV-Pack [23]) clusters the corresponding LUTs and flip-flops to form a connected netlist of CLBs. Furthermore, a placement module places the CLB and IO instances of a netlist to CLBs and IOs of the FPGA [34]. Moreover, a routing module routes all the nets of a netlist on the FPGA routing resources [34]. Finally, the bitstream is generated by using the information provided by the technology mapping, placement and routing module of the VTR CAD flow.

C. NPN Classification

The concept of NPN equivalency is defined as: two functions, say f and g, are considered NPN equivalent, if one can be derived from the other by negating (N) and/or permuting (P) some/all of the inputs and/or by negating (N) the outputs. These two functions are then called NPN equivalent as they have the same NPN class. The NPN classification technique is utilized to categorize a larger number of functions into a smaller set of NPN classes. For n-inputs, there are $2^{exp(2n)}$ distinct possible representations of Boolean functions.

The Boolean space grows very rapidly as the number of inputs increases. However, the number of NPN classes reduces the huge space to a smaller set of unique functions. For example, a 4-input LUT supports 22n = 65536 possible functions, while these are classified into 222 distinct NPN-equivalent classes only. It is relevant to mention that no formula can be given for the number of distinct functions. However, [35] provides more details about NPN classification. Furthermore, the functions belonging to the same NPN class can be represented through the same bitstream values provided that there is an option to negate (N) and/or permute (P) the inputs, and/or negate (N) the outputs. Consequently, the number of SRAMs in the LUTs can be reduced by sharing the SRAM table between two or more LUTs.

The work in this article uses the NPN class information similar to the approach followed in [21]. The ABC synthesis tool synthesizes the given benchmark logic functions for a k-input LUT into k-input Boolean functions. The corresponding class information is also given as output for each of the synthesized Boolean functions. The modified LUT and the CLB architecture, along with the corresponding CAD tools to support an NPN mapping, are discussed in Section III.

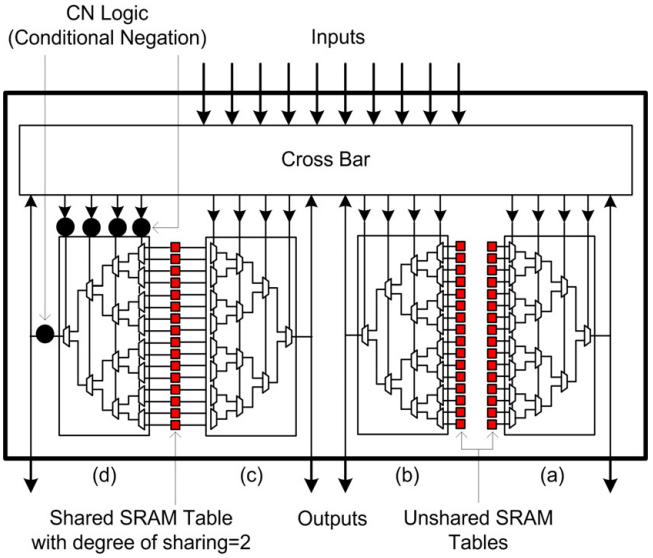

Figure 1. Configurable Logic Block with 4 BLEs and 1shared group with a degree of sharing=2

III. Proposed CLB Architecture Exploration and CAD Flow

This section presents the changes required for the exploration of CLB architecture in Section III (A). The proposed architectural modifications need to be harnessed through proper changes in the clustering algorithm of corresponding CAD tools. Therefore, Section III (B) presents the necessary changes in the clustering algorithm.

A. Modified CLB Architecture

An example of two LUTs, sharing a single SRAM table, is shown in Fig. 1 which represents a configurable logic block with 4 LUTs. The four inputs of each LUT are connected with a crossbar which connects the global inputs and outputs to individual inputs of the LUT. Two LUTs named (a) and (b) in Fig. 1 are connected with their corresponding SRAM tables, each having 16 SRAM cells. However, the LUTs named (c) and (d) are sharing a single SRAM table which is termed as a single group with a degree of sharing=2.

The concept of NPN classification eases the implementation of similar NPN class combinatorial functions using the same circuit using the optional negation of input and output or permutation of inputs. Consider three basic examples:

Example 1: Consider the functions f1 = a b’ and f2 = a’b, they are P-equivalent, i.e., one function’s inputs can be swapped to implement the other function using the same gate.

Example 2: Consider the functions f1 = ab + c and f2 = a’c’ + b’c’. If f1 is inverted we get, (ab+c)’ = (ab)’c’ = (a’ + b’) c’ = a’c’ + b’c’ (using de Morgan’s Theorem); the other function f2 is inverted output of f1.

As we see, there may be cases where the function inputs or outputs may be negated or inputs permuted and share the same logic circuit. The Complementary Negation (CN) circuitry is an addition to the shared BLE, but its usage is solely dependent on the mapped Boolean functions. As you can in example 2, the f2 function is derived by negating the output of f1. It is a simple output negation in this example and is only achieved by CN logic. However, P-equivalent functions do not need CN logic (Example 1). Therefore, if there are 4 BLE(s) in a shared LUT mode, three of the BLE(s) will have the CN module to implement the same NPN class functions, while the one BLE will be directly implementing the Boolean function with no NPN requirements, no input or output negation.

The optional CN circuit slightly increases the delay; however, the critical path delay can be mitigated by shifting the critical logic functions onto the LUTs without CN gates.

B. Proposed Modifications in Clustering Algorithm

The clustering algorithm in VTR [35] is modified to cluster various logic elements of the user circuit based on their class information. Under the standard scenario (without any modifications), the clustering algorithm uses an attraction function to group the closely connected logic elements and place them in the current CLB. The same process is repeated till all the logic elements are placed inside a CLB. In this article, the normal clustering algorithm is modified to incorporate the class information for packing. In other words, the T-VPACK clustering algorithm in VTR is modified to obtain an input blif file, representing the user circuit and the class information of each Boolean function in the user circuit.

1) Basic Principle

Modifications are made to cluster the Boolean function based on their NPN class information and the attractive function. The clustering algorithm uses a cost function (named as an attractive function) to find out which logic blocks need to be clustered together based on wire length and delay. The goal is to decide if the random BLE placement is good enough based on the reduction in the cost function for the possible move, else the move is rejected. Based on the occupancy of the current CLB, and the sharing status of BLEs, the incoming Boolean function may or may not be accommodated in the current CLB. If the SRAM table of a BLE is shared with one or more BLEs, all of the shared BLEs can map only the same class function. As Boolean functions get mapped into a CLB, the available unoccupied BLE(s) in the CLB are either tagged as shared or unshared.

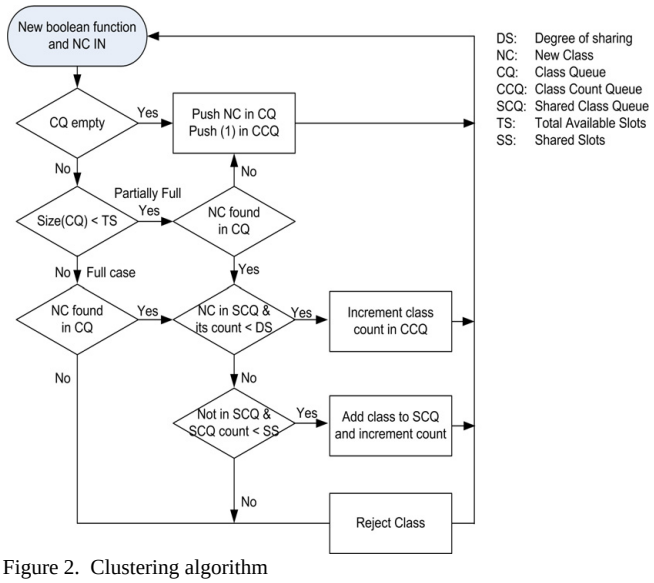

Figure 2. Clustering algorithm

The unshared BLE(s) can still get any Boolean function. However, the available shared BLE can only accept the same class function. This may leave CLB(s) partially mapped if the next function is not one of the classes already mapped. If the incoming Boolean function is rejected based on the class information, a newer function along with the corresponding class information is evaluated by the attractive function. Thus, the clustering algorithm works iteratively to map all the Boolean functions onto the CLB(s). It is important to note that the modified clustering algorithm maintains few dynamic queues to keep track of shared and unshared LUTs within a CLB. These queues include the class queue (CQ) and class count queue (CCQ) to keep track of the number of shared and unshared LUTs occupations. The shared CQ and shared CCQ allow users to try a combination of one or more shared LUTs (2,3,4). These queues allow the incoming class function to be mapped to the shared LUT among N LUTs for the current CLB provided the incoming class matches the one in the shared queue.

2) Flow Chart for Clustering Algorithm

The modified T-VPACK algorithm that deals with the grouping of BLEs, based on their class information, is depicted through a flow chart in Fig. 2. The flow chart in Fig. 2 can be explained with a simple example. Consider that there are 16 LUTs in each CLB of an FPGA. Each CLB has a few shared LUTs and a few non-shared LUTs. The class CQ and class count queue CCQ are initially empty, as no functions have been mapped onto LUTs. The first incoming function can be mapped to any LUT. However,

the subsequent incoming function is mapped according to the shared and non-shared LUTs rules. As an example, consider three different degrees of sharing (2, 3 and 4). It implies that 2 LUTs share a single SRAM, 3 other LUTs share a single SRAM and 4 other LUTs share a single SRAM. Consequently, the total number of unshared LUTs with their corresponding SRAMs is 16-(2+3+4) = 7.

TABLE I. MCNC NETLISTS FOR LUT-6, CLUSTER-SIZE 16

Index	Netlist Name	Number of CLBs	FPGA Size	Min Channel Width*
1.	pdc	228	16x16	76
2.	ex5p	48	7x7	50
3.	spla	191	14x14	78
4.	apex4	61	8x8	68
5.	ex1010	197	15x15	78
6.	frisc	185	14x14	90
7.	apex2	93	10x10	50
8.	seq	83	10x10	52
9.	misex3	75	9x9	42
10.	elliptic	134	12x12	68
11.	alu4	74	9x9	34
12.	des	35	6x6	38
13.	s298	83	10x10	40
14.	bigkey	43	7x7	34
15.	diffeq	55	8x8	46
16.	dsip	43	6x6	36
17.	tseng	50	8x8	80
18.	clma	390	20x20	94
19.	s38584.1	306	18x18	54
20.	s38417	238	16x16	48

The clustering algorithm assigns the first LUT to the incoming Boolean function. For the second function of the same class, the function gets mapped to the same LUT. However, if the second function has a different class than the first, it gets mapped to a new LUT. Similarly, for all the next new functions, the class information is checked. If the new function matches and the shared count is greater than 1, it indicates a shared class. Subsequently, the shared count is checked to see if it does not exceed 4, 3 or 2 in this example. If it does not match any of the mapped LUTs' classes, the newer function is mapped to an unmapped LUT. At any time, the counts of shared and unshared LUTs are continuously checked according to the given CLB configuration.

Finally, if all the unshared LUTs are mapped and shared LUTs are partially full and a new Boolean function of a new class does not happen to be any of the shared LUTs classes, it is rejected back to the FPGA mapping requestor. As it is evident, the shared slots requires tracking to ensure same class goes in the shared slot and it's count does not exceed the sharing configuration, a list and its count is maintained for each shared slot. The mapper also ensures the N-BLE CLB count is not exceeded. Once all N- BLE(s) are mapped, the next empty CLB is then mapped.

It is important to note that there may be a higher number of rejections in a high degree of sharing which compel CLBs to remain partially mapped. The partial mapping of CLBs decreases the packing efficiency as compared to the case where CLBs get close to fully mapped. Higher rejection rate, i.e., the ratio of total rejections to total acceptances, results in more CLB usage thereby increasing the total FPGA area compared to the case where the rejection rate is small or none. It is therefore necessary to apply a range of digital circuits, from various benchmark test case suites, to validate the new architecture and compare the obtained results with existing architectures. The validation results of the new architecture, along with its comparison, are presented in Section IV.

IV. EXPERIMENTATION DETAILS

The experimental setup, required to test our architecture, consists of running the FPGA VTR tool on a given benchmark test. The resulting run log provides details of all the corresponding consumed resources in mapping digital logic circuits onto FPGA. However, the resources of interest in our exploration include the number of occupied CLB(s), the logic and routing area for a particular netlist, the delay and configuration memory requirements.

A. Benchmark Circuits

Several benchmark suits are used to test out different possibilities of Boolean functions for a k-input LUT. One of the commonly used, having a mix of combinatorial and sequential logic, is the MCNC benchmark suite [31]. The MCNC suite of benchmarks, as shown in Table I, is used to evaluate the efficiency of the suggested architecture. To summarize, there are 20 test cases in all, 7 of them are pure combinatorial logic circuits while the remaining 13 also involve sequential circuits. The MCNC netlists are synthesized with LUT-6 and have 16 BLEs in a single CLB cluster. Table I shows the number of CLBs, FPGA dimensions and the minimum channel width required for each netlist as shown in column 3, column 4 and column 5, respectively.

TABLE II. CONFIGURATIONS FOR DEGREE OF SHARING INSIDE A CLB

S.No	Homogeneous degree of sharing	SNo	Heterogeneous degree of sharing
1	[21111111111111]	27	[633211]
2	[2211111111111]	28	[6541]
3	[222111111111]	29	[75211]
4	[222211111111]	30	[6433]
5	[22222111111]	31	[533221]
6	[2222221111]	32	[75211]
7	[222222211]	33	[6433]
8	[22222222]	34	[3332221]
9	[31111111111111]	35	[64321]
10	[331111111111]	36	[6541]
11	[3331111111]	37	[4322211]
12	[33331111]	38	[652111]
13	[333331]	39	[74221]
14	[4111111111111]	40	[533221]
15	[4411111111]	41	[73321]
16	[4441111]	42	[7711]
17	[4444]	43	[65221]
18	[511111111111]	44	[4432111]
19	[55111111]	45	[55411]
20	[5551]	46	[743]
21	[61111111111]	47	[6442]
22	[661111]	48	[6541]
23	[7111111111]	49	[772]
24	[7711]	50	[652111]
25	[811111111]	51	[6532]
26	[88]	52	[44332]

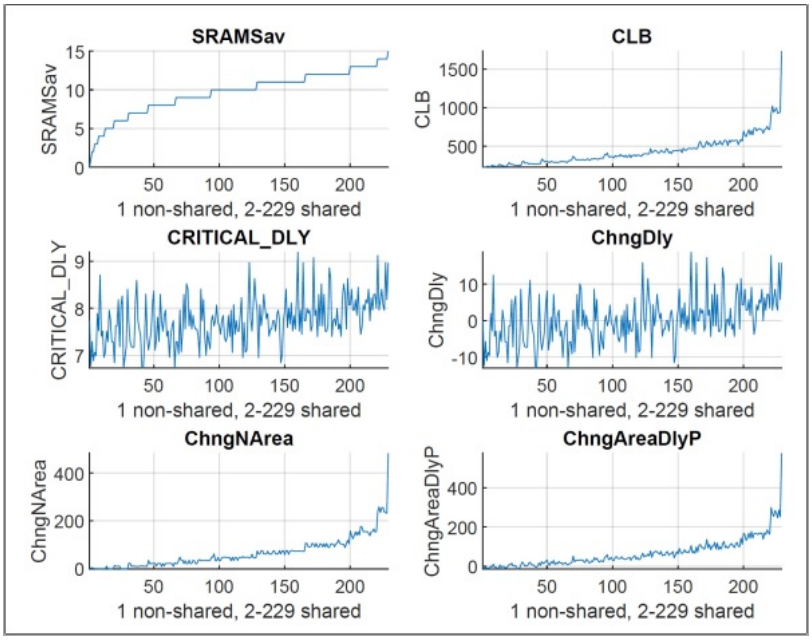

Figure 3. Experimentation results for the netlist named PDC

B. Exploration of a CLB Architecture

This work explores the sharing of an SRAM memory table between 2-8 LUTs in the same CLB, i.e., the degree of sharing is varied between 2 to 8. Similarly, a CLB may consist of multiple shared groups having a different degree of sharing (i.e., the heterogeneous degree of sharing). For a CLB with 16 BLEs, an exhaustive search yields 228 different sequences, containing a degree of sharing from 2 to8. Table II lists some of the 228 configurations used to map the benchmark circuits onto a CLB size of 16 BLE(s).

The first type of sharing consists of a homogeneous degree of sharing where a CLB consists of some shared BLEs with the same degree of sharing along with some unshared BLEs. The homogeneous degree of sharing is shown in S.No 1-26 of Table II. For example, the configuration [4 4 4 1 1 1 1] refers to a CLB with three sharing cases (shown as 4, 4, 4). In each sharing case, 4 LUTs share the same SRAM table. The remaining four LUTs are not shared (1, 1, 1, 1). Therefore, the CLB utilizes only 7 SRAM tables instead of 16.

In addition to the homogeneous sharing, another type of sharing consists of heterogeneous degrees of sharing i.e., a CLB may consist of shared cases with different degrees of sharing. Table II shows a few CLB configurations having a heterogeneous degree of sharing (from S. No 27-52). The idea to explore these different CLB architectures is to find the best combination in terms of FPGA area, critical delay, and/or configuration memory count. The Area-Delay product is used to find an optimal architecture.

To find the best generic FPGA architecture with SRAM table sharing, experiments are performed in three steps.

- Experimentation is performed for individual netlists where the FPGA architecture is tailored concerning individual netlists (Section IV (C)).
- The average results for all the netlists are analyzed (Section IV (D)).
- Experimentation is performed on fixed FPGA configurations and average results are reported for all netlists (Section IV (E)). The reference configuration is the non-shared case, where each of the sixteen BLE(s) has its own LUT, thus allowing maximum mapping flexibility to the incoming LUT function of the benchmark. This non-shared case is used as a baseline value to find the changes in various design parameters for a shared case. The respective percent change concerning the non-shared case is calculated and the most negative ones suggest that the sharing sequence architecture yields the best results.

The sharing sequences suggest savings of SRAM(s) and consequently the logical area as the degree of sharing is increased. However, this saving is consistent for any netlist that is mapped onto a LUT sharing architecture. As an example, a sequence of [6 3 3 2 1 1], implies six LUTs share a single SRAM in the first shared group, three LUTs share another SRAM, another three LUTs share another SRAM, two LUTs share a single SRAM, and remaining two LUTs don't share any SRAM. Each shared SRAM table saves the overall area and configuration memory count of the CLB; however, additional CN cells need to be incorporated at the inputs and output of LUT. In the context of the overall FPGA area, the apparent tremendous gains per CLB provided by reduced SRAM usage, is a tiny part of the overall logical area, as interconnect memory also consumes a large part of the logical area.

C. CLB Architectures for Individual Netlists

As a first step, each netlist is mapped onto the FPGA for 228 different CLB configurations using the VTR tool. The FPGA dimensions and channel width are varied for each netlist. The 130nm custom architecture file is adjusted for the delay variations due to extra inputs and outputs required for CN logic, as the logic delays of CN logic are measured for the 150nm process node using Cadence Virtuoso. The VTR logs are examined to analyze the number of CLB(s), the size of the FPGA, the critical path delay, the routing and logical area, and the number of configuration memories.

Consequently, Fig. 3 shows the results for one such netlist named PDC. This figure shows six sub-graphs. The x-axis of all the sub-graphs plots 229 different CLB configurations with the first configuration as the non-shared CLB. The y-

axis of the six sub-graphs respectively represents SRAM tables saved, number of CLBs utilized by the netlist, critical path delay of the netlist, change in delay, change in the area, and change in area/delay product. Based on these results, a few CLB configurations as discussed in Table II are selected, which provide better results for each netlist.

An important aspect of the ineffectiveness of large SRAM savings is the same NPN class requirement of the target netlist. This is an unknown sequence; therefore, we see that for all bench-marks, the number of CLB(s) used to map the netlist increases as the degree of sharing increases, thus confirming the random unpredictable nature of class sequence for a netlist. As an example, please see Fig. 3 for the high CLB usage as the sharing increases from left to right on the x-axis for the pdc benchmark. The minimum CLB usage for this benchmark is 230 and it grows to 1715 for the degree of sharing equal to 16, which is the best SRAM saving but turns out to be the worst in terms of CLB usage. This increases the overall FPGA grid size, thus impacting the Area-Delay Product graph as well. As the FPGA size increases, the logical and routing area also increases, and the total area shows a steady increase with increasing the degree of sharing.

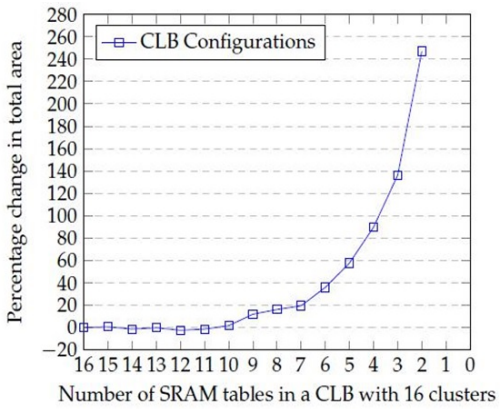

Figure 4. Average results for different CLB configurations

The critical path delay shows a random pattern instead, suggesting there are some sequences for the degree of sharing which have a critical path as low as the unshared case, showing the effectiveness of the sharing technique. The Area-Delay Product, a transistor technology-independent value, is commonly used to evaluate one architecture gain against another. When the value is low, the architecture is considered better for the area and/or delay for the same benchmark circuits. The corresponding graph confirms that there are sequences for which one can achieve the same area delay product with lower LUT SRAMs. The class distribution of NPN classes in the 20 benchmark suite is wide and random enough to analyze FPGA mappings across various FPGA architectures. The obtained results confirm that the variations of NPN class frequency in different benchmark circuits are broad enough to get one sharing sequence in minimum percentage for the Area-Delay Product parameter.

D. Average Results for all Netlists

To get a meaningful result that does not get skewed by one benchmark or another, the average of the calculated parameters is computed for all twenty benchmarks. The resulting plot in Fig. 4 shows the average percentage change in the total area on the y-axis whereas the required number of SRAM tables for the few best from 229 different CLB configurations is shown on the x-axis.. The total area deviation of CLB remains close to zero when SRAM tables are reduced from 16 to 10. As the SRAM table savings are further reduced to 7, only a 20% increase in the area is reported. However, as the SRAM tables are further reduced below 7, the area increases at an exponential rate. Hence, it can be deduced that by using SRAM table sharing, a CLB cluster with 16 BLEs needs to have only 10~7 SRAM tables, which incurs an area compromise of 0~20%, respectively with configuration memory improvement in the logic block from 38~56% and negligible effect on delay.

E. Results on Fixed Sized FPGA

As a third step, the fixed-sized FPGA with fixed channel width is constructed for the selected CLB configurations obtained from the previous experiments. By selecting a fixed-sized FPGA, the effect of new CLB architectures can be viewed on the occupancy of the FPGA for each netlist. This scheme will give a true effect if the proposed architecture is to be included in a commercial FPGA. The fixed-sized FPGA is constructed by including the maximum number of CLBs and the maximum channel width required by any of the netlists.

The netlist results for the fixed-sized FPGAs are reported in Table III. It shows the ten best CLB configurations having SRAM tables with different degrees of sharing. The first column shows the CLB configuration with the corresponding degree of sharing. The number 1 in the CLB configuration represents that a single SRAM table is connected with a single BLE. Similarly, 2 mean that a single SRAM table is shared with two BLEs and so on. Each of the ten CLB configurations has a total of 16 BLEs, but the number of SRAM tables varies for each configuration. Therefore, the first row with 16 ones represents the reference CLB architecture. The "Total SRAMs" used in each CLB configuration, and the "Percentage SRAM saving" are reported in the second and third columns, respectively. The "Occupancy" column represents the average number of CLBs on the FPGA occupied by all the circuits in the benchmark. Similarly, the "delay" column represents the average delay for all the netlists. The 6th and 7th columns report the true number of SRAM tables used, and their percentage gain concerning the reference CLB architecture. As the percentage occupancy increases for CLB configurations having smaller number of SRAM tables, the total SRAM tables concerning occupancy (column 6) is higher than the SRAM savings reported in "% SRAM saving" (column 3).

The first row in Table III shows the reference CLB configuration, while the results in SRAM savings (column 3) and True SRAM savings (column 7) are compared with their respective values in the first row. The following major conclusions can be drawn from Table III:

- **Effects of reduced SRAM tables on occupancy:** It should be noted that as the total SRAM tables are reduced for different CLB configurations (shown in column 2), the CLB occupancy on the FPGA increases (shown in column 4).

TABLE III. AVERAGE MCNC RESULTS ON FIXED-SIZED FPGA

CLB configuration shown w.r.t degree of sharing (1)	Total SRAMS (2)	% SRAMs Saving (3)	Occupancy (4)	Delay (5)	Total SRAMs w.r.t occupancy (6)	True % SRAM Saving (7)
[1111111111111111]	16	00.0	103.1	5.46	1649	00.0
[22222111111]	11	31.3	104.0	5.46	1144	30.6
[2222311111]	10	37.5	106.6	5.59	1066	35.3
[222331111]	9	43.7	110.1	5.31	991	39.9
[33331111]	8	50.0	113.0	5.56	904	45.2
[4441111]	7	56.3	119.3	5.55	835	49.3
[445111]	6	62.5	129.6	5.52	778	52.8
[67111]	5	68.7	149.1	5.41	745	54.8
[6811]	4	75.0	173.8	5.60	695	57.8
[781]	3	81.3	214.3	5.75	643	61.0

However, the CLB occupancy gradually increased from 103.1 to 119.3 when the SRAM tables are reduced from 16 to 7. This amounts to only a 0.5% ~15% increase in CLB occupancy of FPGA when SRAM tables are reduced to 11~7, respectively. The occupancy increases to 129.6 and 149.1 with further SRAM table reductions to 6 and 5. These reductions might be acceptable for few applications as the SRAM tables for these two cases have reduced by 63% and 69% more reduction of SRAM tables down to 4 and 3 increases the occupancy to 173.8 and 214.3.

- **Representation of true SRAM tables after considering the corresponding increase in occupancy:** Column 3 in Table III shows the percentage reduction in SRAM tables for any CLB architecture. However, this is not the true gain, as a netlist is synthesized into more CLBs with reduced SRAM tables. In other words, a more realistic gain can be considered after considering the increase in CLB occupancy on the FPGA. Consequently, column 7 in Table III represents the true percentage savings in the SRAM tables.
- **Effects of reduced SRAM tables on delay:** As the SRAM tables in a CLB are reduced, each netlist occupies more CLB blocks on the FPGA which slightly increases the delay (as shown in column 4). However, for most of the cases, the increase in delay is well within 2.5%. Only the last two CLB configurations, with 4 and 3 SRAM tables, exhibit more skewed delays.
- **Effects of the higher and heterogeneous degree of sharing:** The results in Table III show the CLB architecture with a degree of sharing up to 8. Moreover, a heterogeneous degree of sharing is also shown such that the degree heterogeneity is architecture with a degree of sharing up to 8. Moreover, a heterogeneous degree of sharing is also shown such that the degree of heterogeneity is restricted to maximum of two different types of sharing. The degree of sharing in each type is greater than 1.

F. Practical Design Netlists

To measure the effectiveness of the proposed approach, three different sized more real netlists were tested with ten sets of SRAM sharing. The CLB consumption was varied for various sets of sharing. Compared to the non-shared mode, the CLB consumption reduced in the shared mode for the first of the sharing sets for two of the three netlists. Overall, the logical area of the resulting FPGA was less than one in the non-shared mode for one netlist. Accordingly, the routing area also reduced, resulting in the total area reduction of 50%. Table IV shows the FPGA mapping parameter values for the three netlists in non-shared mode and shared mode (best sharing sequence selected in terms of least CLB consumption). Table IV lists the best mapping result for each of the three netlists in shared versus non-shared mode.

The given formula calculates the overall reductions in SRAM cells in the total size of FPGA.

$$\text{Number of SRAMs Reduced} = [2k*(d-1)*N*N] - [CN*(d-1)*N*N] \quad (1)$$

where, k is the LUT input size, d is the degree of sharing and N is the size of FPGA whereas CN is the area of CN logic.

G. Comparison with Previous Works

This work can be compared with logic blocks proposed in COGRE [16], SLM [18] and previously proposed SRAM table sharing-based CLB architectures [21]. The 6-input COGRE cell [16] is 46% smaller and consumes 32% less configuration memory than a 6-input LUT. However, the delay increased by 6.96%. Furthermore, the COGRE cell is optimized for one set of benchmark circuits and might not provide optimized results for another benchmark circuit. Similarly, the 6-input SLM cell [18] is a generic logic block with a 52% smaller area and 58% less configuration memory than a 6-input LUT. However, the delay of the BLE has increased by more than 140%. After combining the SLM BLE into CLBs, and then placement and routing of the netlist, the overall increase in delay is recorded to be around 12%. The 6-input shared SRAM table LUT proposed in this work (with 16 BLEs in a CLB using only 7 SRAM tables) is on average 25% smaller and consumes on average 55% less configuration memory than a standard 6-input LUT with a negligible effect on delay (i.e., around only 2%).

While comparing the higher and heterogeneous degree of sharing proposed in this work with the previous work in [21], it has been found that many groups of smaller degrees of sharing (2, 3, 4) work best as compared to the unshared case. A large degree of sharing provides relatively higher gains in terms of apparent SRAM savings, however, the mapping of application to FPGA(s) results in an abundant use of CLBs. This is due to the fact that a large number of LUT(s) may go unshared due to different classes of LUTs.

This can be particularly noted in Table III where the CLB

TABLE IV. FPGA MAPPING PARAMETERS FOR SHARED AND NON-SHARED MODES

XML Architecture File	DSN	N	SHR	CLB Blocks	FPGA Size	Channel Width	Logical Area	Routing Area	Delay	Total Area
K6_N16_L4_130nm.xml	blob_merge_6	16	[1]	427	21	86	18937100.00	4555060.00	11.76	23492160.00
K6_N16_L4_130nm.xml	blob_merge_6	16	[2 2 2 2 2]	444	22	90	20783600.00	5446860.00	12.42	26230460.00
K6_N16_L4_130nm.xml	sha_6	16	[1]	156	13	58	7257110.00	1271160.00	14.89	23736910.00
K6_N16_L4_130nm.xml	sha_6	16	[2 2 2 2 2]	152	13	58	7257110.00	1271160.00	14.66	25771450.00
K6_N16_L4_130nm.xml	stereovision_6	16	[1]	21	5	26	1073540.00	115557.00	3.84	26039660.00
K6_N16_L4_130nm.xml	stereovision_6	16	[2 2 2 2 2]	16	4	28	687063.00	79528.30	3.76	25771450.00

proposed in the last row termed as [7 8 1] requires only 3-SRAM tables. However, the number of occupied CLBs has increased to 214%. Since the reconfiguration delay is directly proportional to the size of the bitstream (number of SRAMs to be written in an FPGA). We can safely comment that the reduction in SRAM count will eventually reduce the reconfiguration delay. This work has reduced the configuration memory of the logic blocks by 30~50%. However, the SRAMs of routing architectures are not reduced in this work. Hence the total reduction in reconfigurable delay of the FPGA is expected to be reduced by 10~18%.

V. CONCLUSION

This work explores the impact of shared pairs and degree of sharing of SRAM table sharing technique on area, delay and configuration bitstreams of FPGA extensively using a previously proposed clustering technique. A single SRAM table is shared among various LUTs in a CLB by placing optional inverters at their inputs and output. Moreover, the corresponding CAD tools have been modified to support the clustering of NPN equivalent classes on LUTs with shared SRAM tables. The validation of the proposal has been performed with MCNC benchmark circuits. It has been found that a CLB cluster with 16 BLEs may require only 10~7 SRAM tables. This saving in the number of utilized SRAM tables incurs an overall area reduction of 0~20%, 38~56% in configuration memory cells of the logic blocks while having only a negligible effect on the overall delay. For future direction, the SRAM sharing technique also needs to be extended for fracturable LUTs.

REFERENCES

[1] T. S. S. Phani, A. Arumalla, M. D. Prakash, “Partial dynamic reconfiguration framework for FPGA: A survey with concepts, constraints and trends,” Materials Today: Proceedings 2021. doi:10.1016/j.matpr.2021.01.851

[2] A. Podobas, K. Sano, S. Matsuoka, “A survey on coarse grained reconfigurable architectures from a performance perspective,” IEEE Access 8 (2020) 146719–146743. doi:10.1109/ACCESS.2020.3012084

[3] K.E. Murray, J. Luu, M.J.P. Walker, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K.B. Kent, J. Anderson, J. Rose, V. Betz, “Optimizing FPGA logic block architectures for arithmetic,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems 28 (6) (2020) 1378–1391. doi:10.1109/TVLSI.2020.2965772

[4] M. Eldafrawy, A. Boutros, S. Yazdanshenas, V. Betz, “FPGA logic block architectures for effcient deep learning inference,” ACM Trans. Reconfigurable Technol. Syst. 13 (3). doi:10.1145/3393668

[5] A. Boutros, M. Eldafrawy, S. Yazdanshenas, V. Betz, “Math doesn’t have to be hard: Logic block architectures to enhance low-precision multiply-accumulate on FPGAs,” in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA ’19, Association for Computing Machinery, New York, NY, USA, 2019, p. 94–103. doi:10.1145/3289602.3293912

[6] A. Valanarasi, S. SithiShameem Fathima, C. Priya, B. Babu Mohan, J. Preethipilomina, M. Jannath Juvairiya, “Optimizing FPGA logic block architectures for self-repairing hardened arithmetic,” Materials Today: Proceedings. doi:10.1016/j.matpr.2021.01.819

[7] H. Chen, H. Yang, W. Song, Z. Lu, Y. Fu, L. Li, Z. Yu, “Symmetric-mapping LUT-based method and architecture for computing X^Y-like functions,” IEEE Transactions on Circuits and Systems I: Regular Papers 68 (3) (2021) 1231–1244. doi:10.1109/TCSI.2020.3046783

[8] K. Vipin, S.A. Fahmy, “FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications,” ACM Comput. Surv. 51 (4). doi:10.1145/3193827

[9] J. S. Meena, S. M. Sze, U. Chand, T. Y. Tseng, “Overview of emerging nonvolatile memory technologies,” Nanoscale Research Letters 9 (526) (2014) 118:1–118:39. doi:10.1186/1556-276X-9-526

[10] R.T. I. Kuon, J. Rose, “FPGA architecture: Survey and challenges,” Foundations and Trends in Electronic Design Automation 2.2 (2008) 135–253. doi:10.1561/1000000005

[11] X. Zhang, C. Paerson, Y. Liu, C. Yang, C. J. Xue, J. Hu, “Low overhead online data flow tracking for intermittently powered non-volatile FPGAs,” ACM Journal on Emerging Technologies in Computing Systems. doi:10.1145/3371392

[12] A. Chen, “A review of emerging non-volatile memory (NVM) technologies and applications,” Solid-State Electronics 125 (2016) 25–38. doi:10.1016/j.sse.2016.07.006

[13] I. Hariharan, M. Kannan, “Efficient use of on-chip memories and scheduling techniques to eliminate the reconfiguration overheads in reconfigurable systems,” Journal of Circuits, Systems and Computers 28 (14) (2019) 1950246. doi:10.1142/S0218126619502463

[14] E. Ahmed, J. Rose, “The effect of LUT and cluster size on deep submicron FPGA performance and density,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions, 2004, pp. 288–298, doi:10.1109/TVLSI.2004.824300

[15] W. Feng, J. Greene, A. Mishchenko, “Improving FPGA performance with a S44 LUT structure,” in: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA ’18, Association for Computing Machinery, New York, NY, USA, 2018, p. 61–66. doi:10.1145/3174243.3174272

[16] Y. Okamoto, Y. Ichinomiya, M. Amagasaki, M. Iida and T. Sueyoshi, "COGRE: A configuration memory reduced reconfigurable logic cell architecture for area minimization," 2010 International Conference on Field Programmable Logic and Applications, 2010, pp. 304-309. doi:10.1109/FPL.2010.68

[17] P. A. Hadi, B. Hind, N. David, P. Ienne, “Rethinking FPGAs: Elude the flexibility excess of LUTs with and-inverter cones,” in International Symposium On Field Programmable Gate Arrays (2012) 119–128. doi:10.1145/2145694.2145715

[18] Q. Zhao, K. Yanagida, M. Amagasaki, M. Iida, M. Kuga, T. Sueyoshi, “A logic cell architecture exploiting the shannon expansion for the reduction of configuration memory,” in Field Programmable Logic and Applications (FPL), 2014. doi:10.1109/FPL.2014.6927460

[19] J. Meyer, F. Kocan, “Sharing of SRAM tables among NPN-equivalent LUTs in SRAM-based FPGAs,” vol. 15, IEEE Transactions on Very Large Scale Integration Systems (VLSI), 2007. doi:10.1109/TVLSI.2007.893581

[20] A. Asghar, M. M. Iqbal, W. Ahmed, M. Ali, H. Parvez, and M. Rashid, “Exploring shared SRAM tables among NPN equivalent large LUTs in SRAM-based FPGAs,” in International Conference on Field Programmable Technology (ICFPT). doi:10.1109/FPT.2016.7929540

[21] A. Asghar, M. M. Iqbal, W. Ahmed, M. Ali, H. Parvez, M. Rashid, “Exploring shared SRAM tables in FPGAs for larger LUTs and

higher degree of sharing," International Journal of Reconfigurable Computing 2017. doi:10.1155/2017/7021056

[22] A. Asghar, M. M. Iqbal, W. Ahmed, M. Ali, H. Parvez, M. Rashid, "Logic algebra for exploiting shared SRAM-table based FPGAs for large LUT inputs," 2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT), IEEE. doi:10.1109/INTELLECT.2017.8277632

[23] M. M. Iqbal, H. Parvez, F. Hussain, M. Rashid, "An application specific reconfigurable architecture with reduced area and static memory cells," Journal of Circuits, Systems and Computers 30 (4). doi:10.1142/S0218126621500651

[24] H. Parvez, and M. Habib, "ASIF: Application specific inflexible FPGA," Application-Specific Mesh-based Heterogeneous FPGA Architectures. Springer, New York, NY, 2011, pp 77-101. doi:10.1007/978-1-4419-7928-5_5

[25] U. Farooq, H. Parvez, H. Mehrez, and Z. Marrakchi, "A new heterogeneous tree-based application specific FPGA and its comparison with mesh-based application specific FPGA," In Microprocessors and Microsystems, (2012). 36(8), pp 588-605. doi:10.1016/j.micpro.2012.06.012

[26] H. Parvez, Z. Marrakchi, A. Kilic, and H. Mehrez, "Application-specific FPGA using heterogeneous logic blocks," ACM Transactions on Reconfigurable Technology and Systems (TRETS), (2011) 4(3), pp 1-14. doi:10.1145/2000832.2000836

[27] U. Farooq, H. Parvez, H. Mehrez, and Z. Marrakchi, "Exploration and optimization of a homogeneous tree-based application specific inflexible FPGA," Microelectronics Journal, 2013, 44(12), pp 1052-1062. doi:10.1016/j.mejo.2012.12.010

[28] M. M. Iqbal, H. Parvez, and M. Rashid, "Multi-Circuit": Automatic generation of an application specific configurable core for known set of application circuits," Journal of Circuits, Systems and Computers, 2016, 25(09), 1650102. doi:10.1142/S0218126616501024

[29] M. M. Iqbal, H. Parvez, "Optimizing routing network of shared hardware design for multiple application circuits," In 2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT) (pp. 1-4). IEEE. doi:10.1109/INTELLECT.2017.8277646

[30] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J. P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, V. Betz, "VTR 8: High performance CAD and customizable FPGA architecture modelling," ACM Transactions on Reconfigurable Technology and SystemsVolume 13 Issue 2 June 2020 Article No.: 9 pp 1–55. doi:10.1145/3388617

[31] K. S. McElvain, "Benchmark Set: Version 4.0," MCNC International Workshop on Logic Synthesis, 1993. doi:10.1.1.49.591

[32] P. Jamieson, K. B. Kent, F. Gharibian, L. Shannon, "Odin II - An open-source verilog HDL synthesis tool for CAD research," in: 2010 18th IEEE Annual International Symposium on Field Programmable Custom Computing Machines, 2010, pp. 149–156. Berkeley Logic Synthesis and Verification1 Group, ABC: A System for Sequential Synthesis and Verification. doi:10.1109/FCCM.2010.31

[33] R. Brayton, A. Mishchenko, "ABC: An academic industrial-strength verification tool," In: Touili T., Cook B., Jackson P. (eds) Computer Aided Verification. CAV 2010. Lecture Notes in Computer Science, vol 6174. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-14295-6_5

[34] V. P. Correia, A. I. Reis, Classifying n-input Boolean functions, VII Workshop Iberchip. doi:10.1.1.734.9856

[35] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, et al., "VTR 7.0: Next generation architecture and CAD system for FPGAs," vol. 7, ACM Transactions on Reconfgurable Technology and Systems (TRETS), 2014. doi:10.1145/2617593