ARCHITECTURAL EXPLORATION OF FPGAS AND RTL2GDSII IMPLEMENTATION OF AN FPGA

by

Mehmet Sait Eroğlu

B.S., Electronics & Communication Engineering, Istanbul Technical University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical & Electronics Engineering

Boğaziçi University

2023

ACKNOWLEDGEMENTS

I would like to thank Assist. Prof. Faik Başkaya for his support throughout the thesis period. I can create this academic work because of his professional guidance whenever needed.

I also would like to thank Electrical & Electronics Engineering department faculty members for their valuable teaching.

My company, TUBITAK BILGEM, especially my manager Yaman Özelçi encouraged me to choose this topic and helped me whenever needed. I thank my colleagues from YİTAL. I also appreciate Mustafa Arslan and Yunus Emre Eryılmaz from TÜTEL for their help.

And, my special and deepest thank goes to my dear wife, who always sets the working environment I need and make delicious coffees when I need to work until morning.

ABSTRACT

ARCHITECTURAL EXPLORATION OF FPGAS AND RTL2GDSII IMPLEMENTATION OF AN FPGA

Growing design complexity and cost has forced designers to build programmability into System-on-Chip (SoC) designs to reduce the number of costly chip re-spins and amortize IC costs over several fabrications. Programmability of embedded FPGA (eFPGA) cores is one of a handful of design solutions to meet this challenge. However, creating a new FPGA is challenging because of the significant effort that must be spent on circuit design, layout, and verification. The time required until the tape-out is approximately 1 year of a large team from architecture definition to tape-out for a new FPGA, since the process is primarily done manually. Researchers have developed automated methodologies to overcome these barriers by modeling FPGA fabrics as Verilog netlists and generating layouts using ASIC automated design tools. Simplifying and shortening the design process would be advantageous since it could reduce the time to implement eFPGAs in SoCs while enhancing architecture explorations. For this purpose, OpenFPGA is introduced, an open-source framework that enables automated prototyping for FPGA architectures. To enable various design purposes, OpenFPGA integrates several popular open-source EDA tools, i.e., VTR and Yosys, with its own custom tools. In this work, we designed an FPGA using OpenFPGA framework and investigated the issues faced in the architecture selection, circuit design, layout, and verification of such a FPGA with Türkiye's domestic 250nm CMOS technology.

ÖZET

FPGA MİMARİLERİ ARAŞTIRMASI VE DÜŞÜK ALANLI BİR FPGA MİMARİSİNİN RTL2GDSII TASARIMI

Artan tasarım karmaşıklığı ve maliyeti, tasarımcıları çip fabrikasyon maliyetlerini amortize etmek için System on Chip (SoC) tasarımlarında müstakil yeniden programlanabilir yapılar kullanmaya yönlendirdi. Gömülü FPGAler (eFPGA) bu zorluğun üstesinden gelmek için ortaya çıkan çözümlerden biridir. Ancak yeni bir FPGA oluştururken devre tasarımı, layout ve doğrulaması için önemli bir efor gerekir. Ticari bir FPGA'in tanımından tape-out'una kadar gereken süre tecrübeli büyük bir ekibin yaklaşık 1 yılıdır. Bu engellerin üstesinden gelmek için araştırmacılar, FPGA yapılarını Verilog netlisti olarak modelleyerek ve ASIC tasarım toollarını kullanarak otomatik metodolojiler geliştirdiler. Tasarım sürecini otomatikleştirmek, SoC'lerde eFPGA'leri gerçekleme süresini azaltırken aynı zamanda mimari araştırmaları geliştirecektir. Bu amaçla, FPGA mimarileri için otomatikleştirilmiş prototipleme sağlayan açık kaynaklı bir toolset olan OpenFPGA geliştirildi. Çeşitli tasarım amaçlarını gerçeklemek için OpenFPGA, VTR ve Yosys gibi popüler açık kaynaklı EDA toollarını kendi özel tool'ları ile entegre etmektedir. Bu çalışmada, Türkiye'nin yerli 250nm CMOS teknolojisiyle OpenFPGA kullanarak bir FPGA tasarladık ve böyle bir FPGA'in mimari seçimi, devre tasarımı, layout ve doğrulanmasında karşılaşılan süreçleri detaylıca inceledik. Tam otomatik tasarım ile bazı custom tasarım eforları arasında denge kurarak alanı ve hızı optimize etmeye odaklandık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

CB	Connection Block
CLB	Configurable Logic Block
EFPGA	Embedded Field Programmable Gate Array
FPGA	Field Programmable Gate Array
LUT	Look-Up Table
SB	Switch Block

1. INTRODUCTION

FPGAs have become an extremely useful medium for implementing digital designs. With SRAM-based FPGAs, the devices can be programmed in seconds or less. This makes the lengthy and costly fabrication process needed for ASICs. Faster and cheaper initial costs make FPGAs well-suited for low to medium-sized designs. The shortened design time makes FPGAs ideal for prototyping designs before mass production. However, with this ease of use comes a penalty in terms of area, speed, and power, as circuits implemented in FPGAs are at least ten times larger and three times slower than custom implementations due to the reconfigurability overhead of FPGAs. To minimize these factors, the high-speed and low-area design of the FPGA itself is necessary. In this thesis, we aim to design a low-cost FPGA with 250nm CMOS technology, which is Turkey's own developed CMOS fabrication technology. This design will consist of two main architectural and physical design stages. Since this technology is older compared to current commercial ones, there are disadvantages regarding speed and area cost. Firstly, we need to focus on the architectural details of the design for area optimization and clarify the ideal design parameters in terms of Look-up Table (LUT) and Configurable Logic Block (CLB) sizes, channel width, etc. For this purpose, we need to first meet with Verilog-to-Routing (VTR) tool. It is an open-source academic CAD tool designed to explore new FPGA architectures and CAD algorithms at the packing, placement, and routing phases of the CAD flow. VTR, which is the current name of Versatile Place&Route (VPR) tool, takes a description of an FPGA architecture and a Verilog circuit as input. It then performs packing, placement, and routing to map the design onto the FPGA [1]. The output of VTR includes the FPGA configuration bitstream needed to implement the circuit and reports of the final mapped design (e.g., critical path delay, area, etc.).

Secondly, optimizing the layout to meet the operating frequency requirements up to a point is necessary. A compact physical layout is required to achieve this goal, and producing such a layout will require more than a year's worth of work from numerous skilled engineers. Researchers have developed automated methodologies to overcome these barriers by modeling FPGA fabrics as Verilog netlists and generating layouts using ASIC design tools. Simplifying and shortening the design process would be advantageous since it could reduce the time to implement Embedded FPGAs (eFPGAs) while enhancing architecture explorations. For this purpose, OpenFPGA is introduced, an open-source framework that enables automated prototyping for FPGA architectures. To enable various design purposes, OpenFPGA integrates several popular open-source EDA tools, i.e., VTR and Yosys, with its own custom tools [2]. In this work, we mainly used OpenFPGA framework to implement selected architecture as a synthesizable Verilog netlist. We investigated the issues faced in the architecture selection, circuit design, layout, and verification. We focused on optimizing area and speed by trading off between fully automated design and some custom design efforts. This work demonstrates that high area cost and speed losses due to the nature of FPGAs and disadvantages of an old technology node can be overcome to some extent with semi-custom design techniques. Physical design is the second main part of this thesis.

This thesis is constructed as follows. Chapter 2 provides a background for FPGA architectures of the thesis. Chapter 3 mainly includes general explorations of FPGA Architectures and choosing an optimal architecture to design using VTR tool. Chapter 4 dives into physical design, layout, and simulations of the designed FPGA by comparing alternative choices using OpenFPGA and Synopsys Design Tools. Chapter 5 recapitulates the findings and concludes the thesis.

2. BACKGROUND

2.1. Classical FPGA Architectures

FPGA architectures typically follow a structure with repeatable modules. Figure 2.1 depicts a generalized example of an FPGA in which configurable logic blocks (CLBs) are arranged as a grid and are interconnected by programmable routing resources. I/O blocks are arranged around the grid's perimeter and are linked to the programmable routing interconnect.

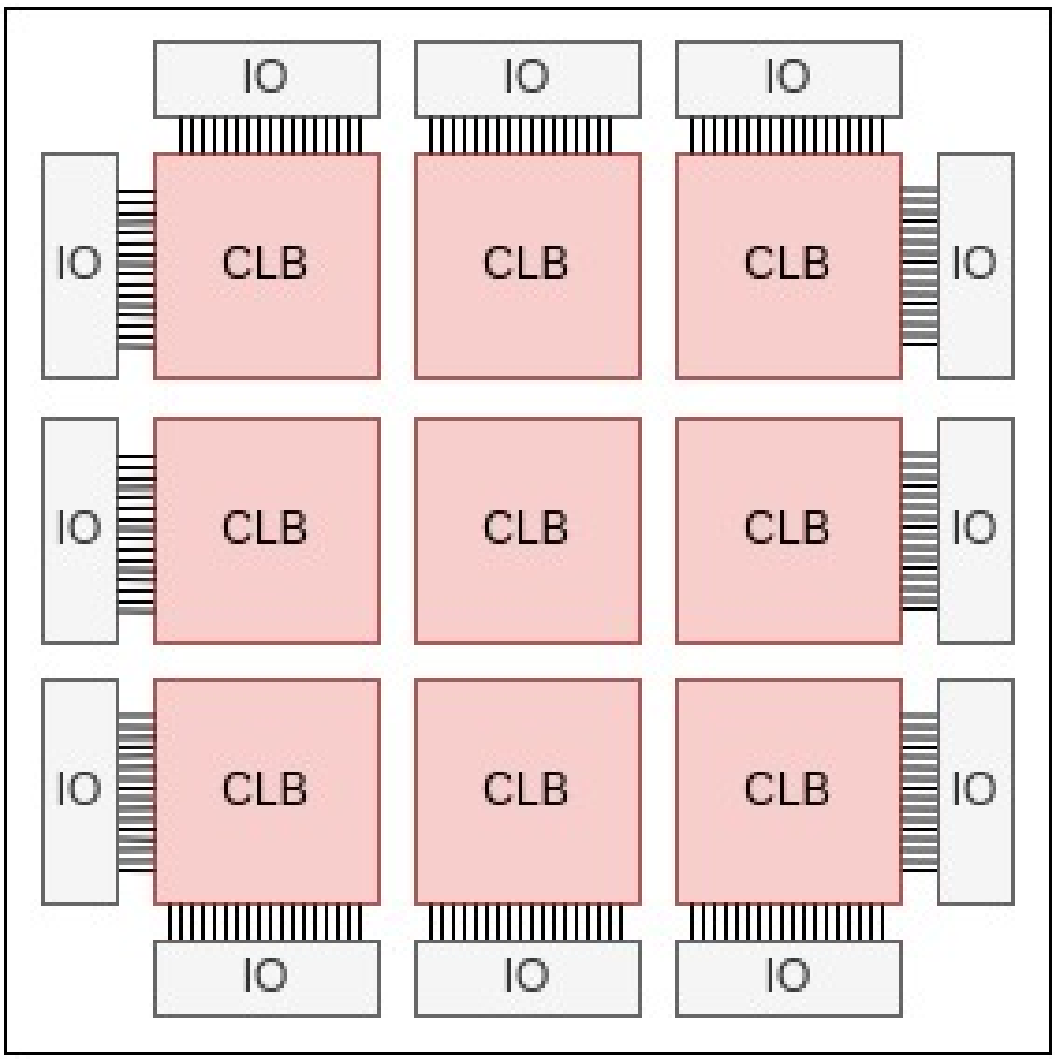

Figure 2.1. A generic FPGA architecture.

2.2. Basic Logic Element (BLE) and Configurable Logic Block (CLB)

A BLE is the basic module that implements logic functions. Figure 2.2 depicts a simple BLE consisting of an LUT and a flip flop. A K-input LUT can realize all K-input one-output boolean function by configuring SRAM bits. Flip flop allows BLEs to implement sequential logic. This logic element can operate in both combinational and sequential mode by configuring the multiplexer.

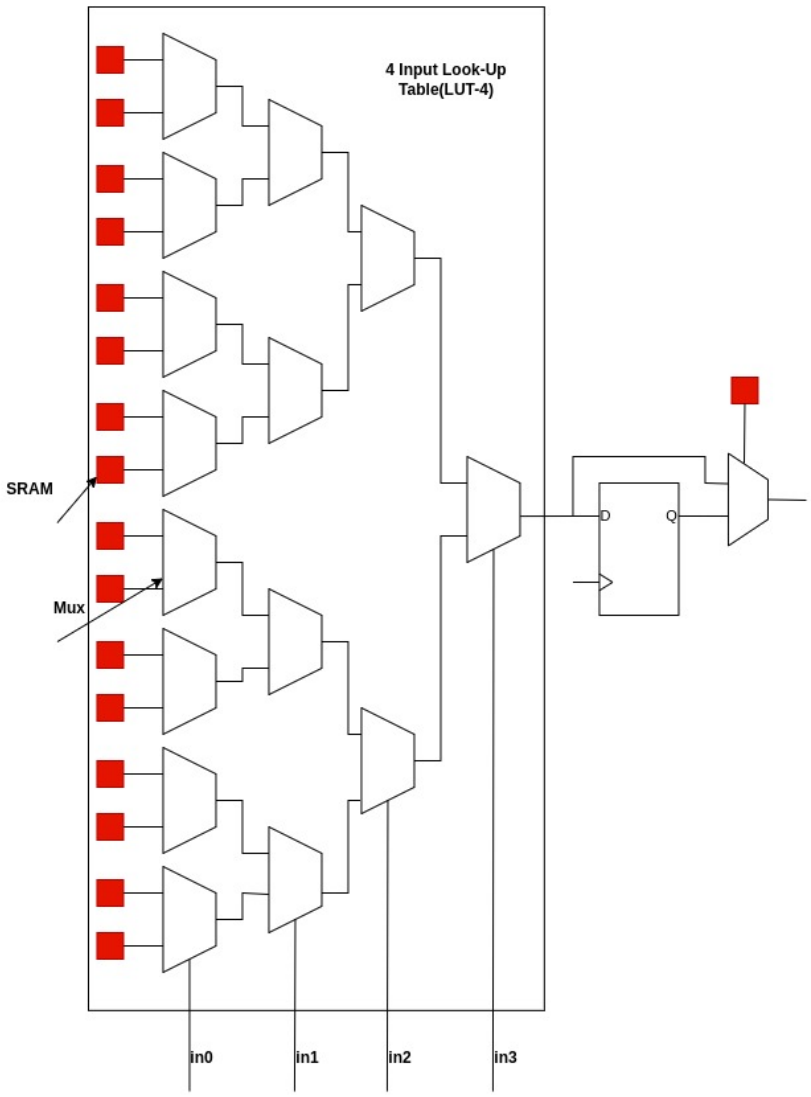

Figure 2.2. 4-input LUT (LUT-4) architecture.

A detailed CLB architecture is depicted in Figure 2.3, in which a number of Basic Logic Elements (BLEs) are connected together by a local routing architecture. The local routing architecture, which consists of various sized multiplexers, crossbars CLB inputs and BLE outputs to BLE inputs. Each BLE input is driven by a multiplexer, whose inputs come from sum of the CLB inputs and BLE outputs, as shown in Figure 2.3. The local routing architecture makes sure that BLEs and CLB inputs are connected to each other. Any big logic function may be implemented by a CLB by connecting LUTs and flip flops due to its full connectivity.

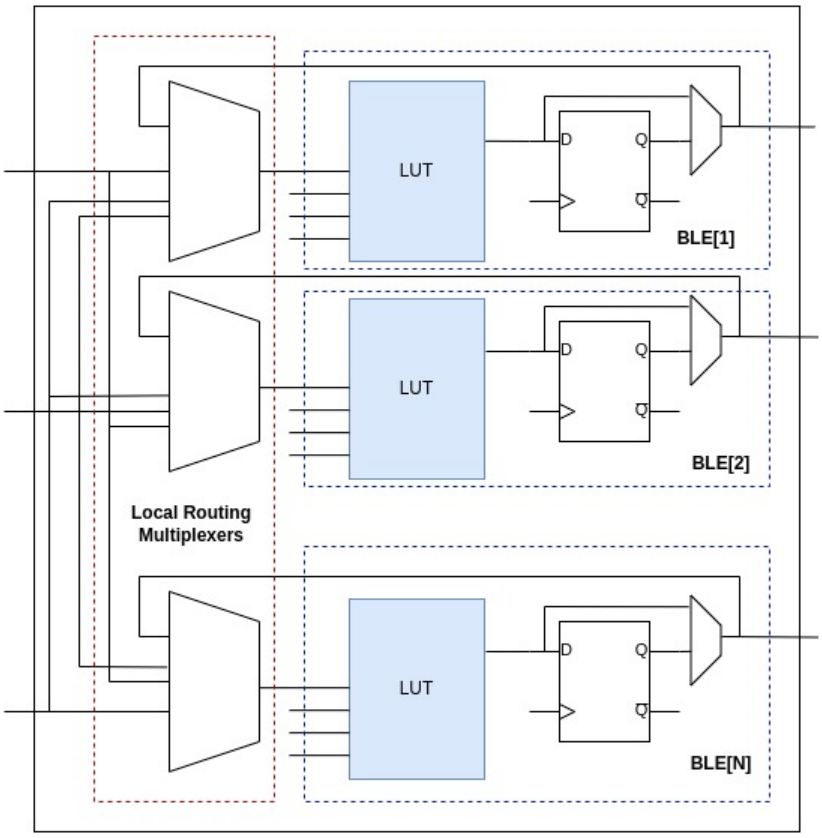

Figure 2.3. Configurable logic block (CLB) architecture.

How many logic functions that can be mapped to a CLB is defined as its logic capacity, which is primarily determined by the following parameters: LUT input size K, number of BLEs in a CLB N, the number of CLB inputs I. Large K, N, and I values enhance the logic performance of CLBs while simultaneously linearly boosting their area, latency, and power. Local routing multiplexers, for instance, have a relationship

between N and I since their input size is N + I. Although big CLBs can cut back on the need of global routing architecture, the increased size of the CLB may cancel out the cost benefits. In Chapter 3, these architectural details will be explained and determine the best trade-off between CLB logic metrics for a medium-sized low-cost FPGA design will be determined.

2.3. FPGA Routing Architectures

Figure 2.4 depicts a typical island-style FPGA architecture. This is the most common architecture used in modern FPGAs. This architecture is known as island-style architecture because the configurable logic blocks in it resemble islands in a sea of routing interconnect.

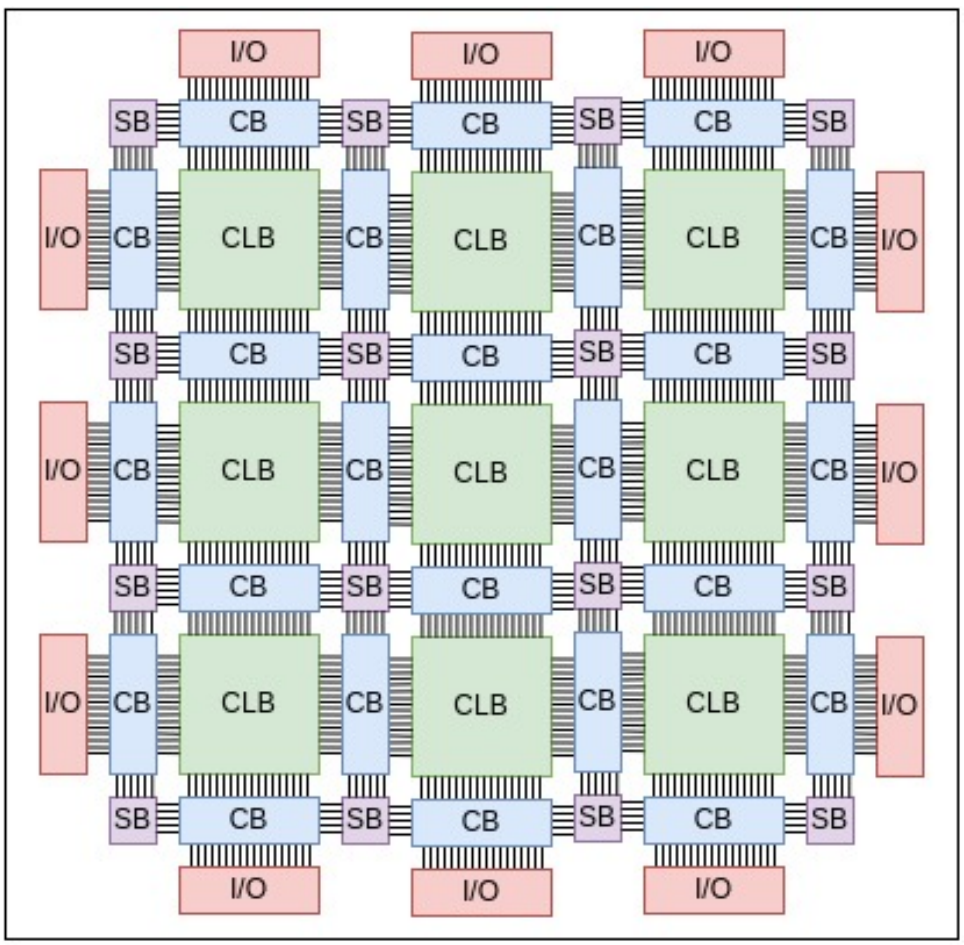

Figure 2.4. Island style FPGA architecture.

Connection Blocks and Switch Blocks are the two types of blocks that make up global routing resources. Although both CBs and SBs are constructed using programmable routing multiplexers, their interconnection topologies differ. While SBs connect routing tracks to one another, CBs connect routing tracks to the inputs and outputs of CLBs. In contrast to local routing design, global routing multiplexers can only connect to a portion of the routing lines. Using sparse connections yields a better trade-off between routing area and routability since Clos showed that multi-level sparse crossbars might achieve flawless routability as fully-connected solutions while drastically lowering routing area [3]. As a result, point-to-point connections in global routing architectures are realized using multiple sparse CBs and SBs. The following parameters are commonly used in global routing architecture to quantify sparse connectivities: Channels are used to group routing tracks, and W indicates how many routing tracks are present in each channel. In the context of CBs, Fc,in is the percentage of routing tracks that can be attached to a CLB input pin. The number of routing tracks that may be linked via a CLB output pin is expressed as Fc,out. Fs stands for the maximum number of routing tracks that each incoming routing track in an SB can link to. An illustration of a global routing architecture with a channel width of four is shown in Figure 2.5.

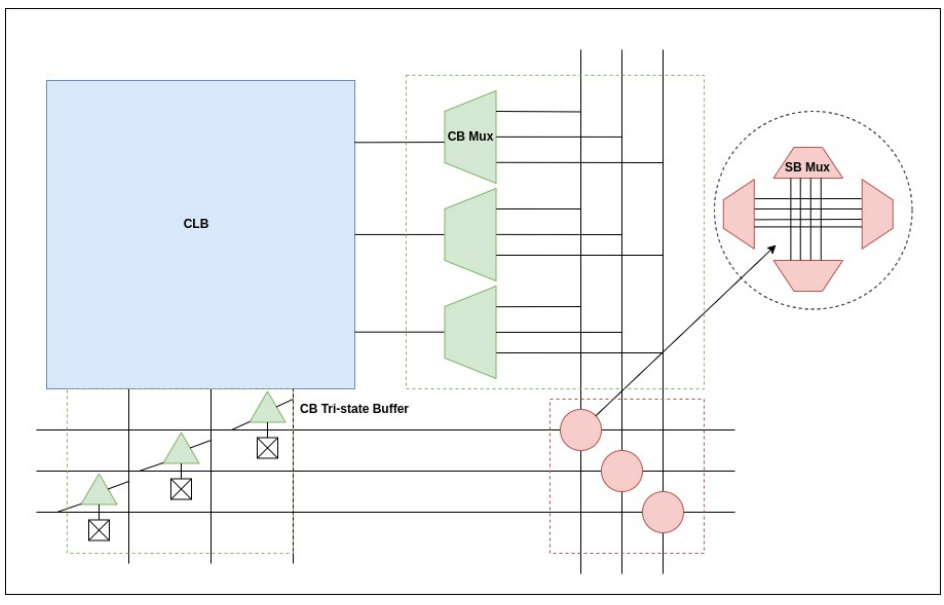

Figure 2.5. Global routing architecture.

Bidirectional or unidirectional (also known as directional) routing tracks can be connected via a switch box. Figure 2.6 depicts a bidirectional and a unidirectional switch box with Fs of 3. Both of these switch boxes' input tracks (or wires) connect to three other tracks in the same switch box. The only limitation of unidirectional switch boxes is that the width of their routing channels must be even. Modern commercial FPGA architectures employ single-driver, directional routing tracks. The authors of [4] demonstrate that by using single-driver directional wiring instead of bidirectional wiring, they can achieve a 25% improvement in area, a 9% improvement in delay, and a 32% improvement in area-delay. All of these benefits are obtained without requiring any significant changes to the FPGA CAD flow.

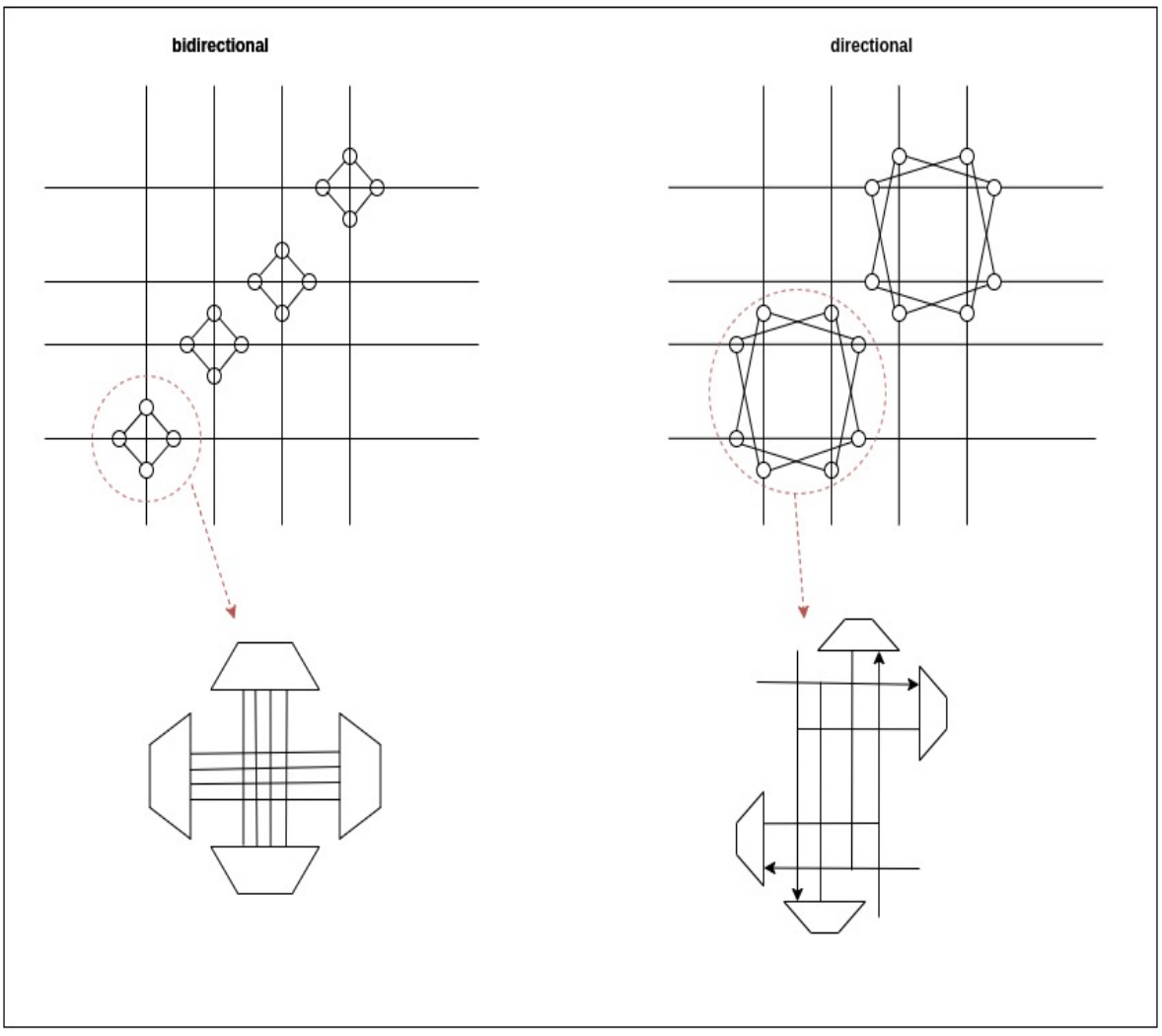

Figure 2.6. Switch box block.

Multi-length wires are created in mesh-based FPGAs to reduce delay. Figure 2.7 depicts a variety of wire lengths. Longer wire segments span multiple blocks and necessitate fewer switches, reducing routing area and latency. They do, however, reduce routing flexibility, which reduces the likelihood of successfully routing a hardware circuit. Modern commercial FPGAs commonly use a combination of long and short wires to balance the routing network's flexibility, area, and delay.

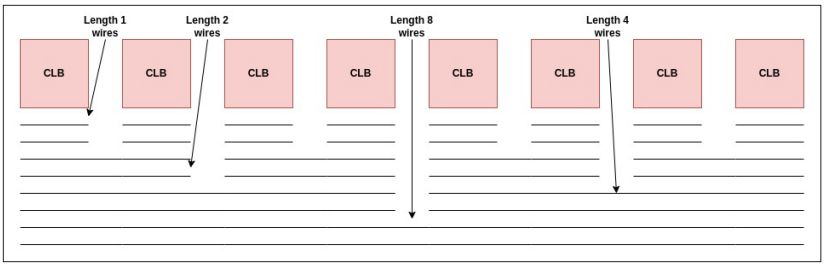

Figure 2.7. Channel segment distribution.

2.4. Architectural Enhancements

2.4.1. Tile-based Heterogeneity

FPGAs typically employ a tile-based heterogeneous architecture, in which the entire FPGA is arranged with repetitive tiles, as shown in Figure 2.8.

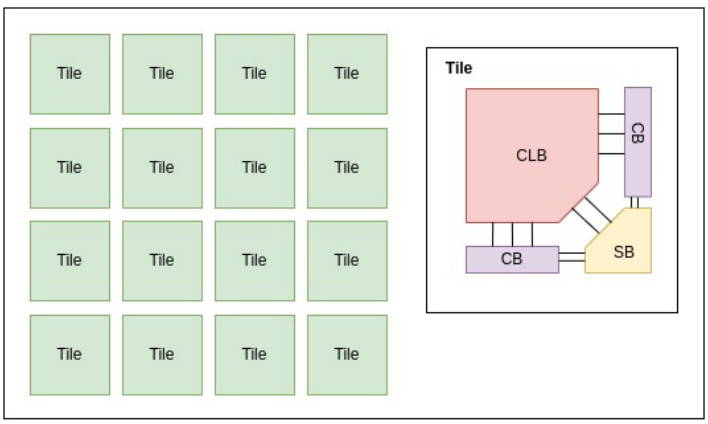

Figure 2.8. Tile-based FPGA architecture.

The use of heterogeneous blocks in Figure 2.9 is intended to improve the trade-off between programmability and efficiency. To overcome the limitations imposed by FPGA implementation overheads, hard macros are embedded in modern FPGA architectures to implement the most commonly used logic functions. Hard macro blocks, such as the signal processing blocks and memory units shown in Figure 2.9, replace various-sized tile array parts. Manual FPGA layouts outperform automatically generated layouts in terms of area and performance. As shown in Figure 2.8, each tile contains a CLB, two CBs, and one SB, while routing tracks are only connected by SBs. This allows engineers to spend more effort for layout issues and makes easier to floorplan the tile macros.

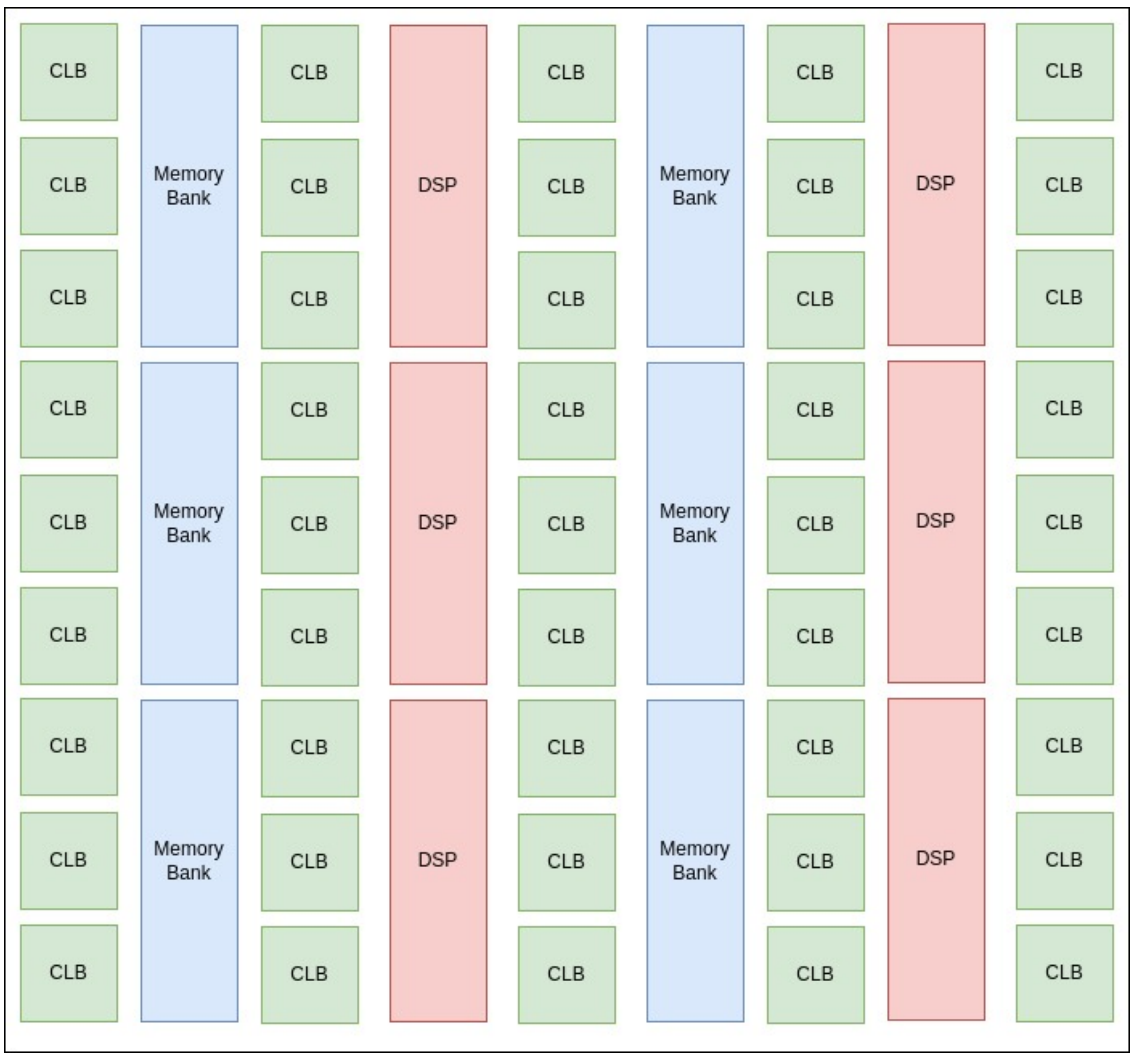

Figure 2.9. Heterogeneous tiles.

2.4.2. Fracturable LUT

Because classical FPGAs only have one type of LUT with a fixed input size, they frequently have low utilization rates. A K-input LUT can be split into two (K - 1)input LUTs in modern FPGAs, doubling its logic capacity. In comparison to the classical LUT structure in Figure 2.2, fracturable LUT in Figure 2.10 has an additional output and it can map 2 different function having shared inputs. For example, the 3-input LUT[0] can handle the 3-input logic function f0(x0, x1, x2) by using in0, in1, and in2. By sharing in1 and in2 with 3-input LUT[0], the 3-input LUT[1] can still implement another 3-input logic function f1(x1, x2, x3). The 4-input fracturable LUT, on the other hand, can implement two small functions with no common inputs. Logic functions f2(x0, x1) and f3(x2, x3), for example, can be mapped to 3-input LUT[0] and 3-input LUT[1], respectively. This capability significantly improves LUT efficiency.

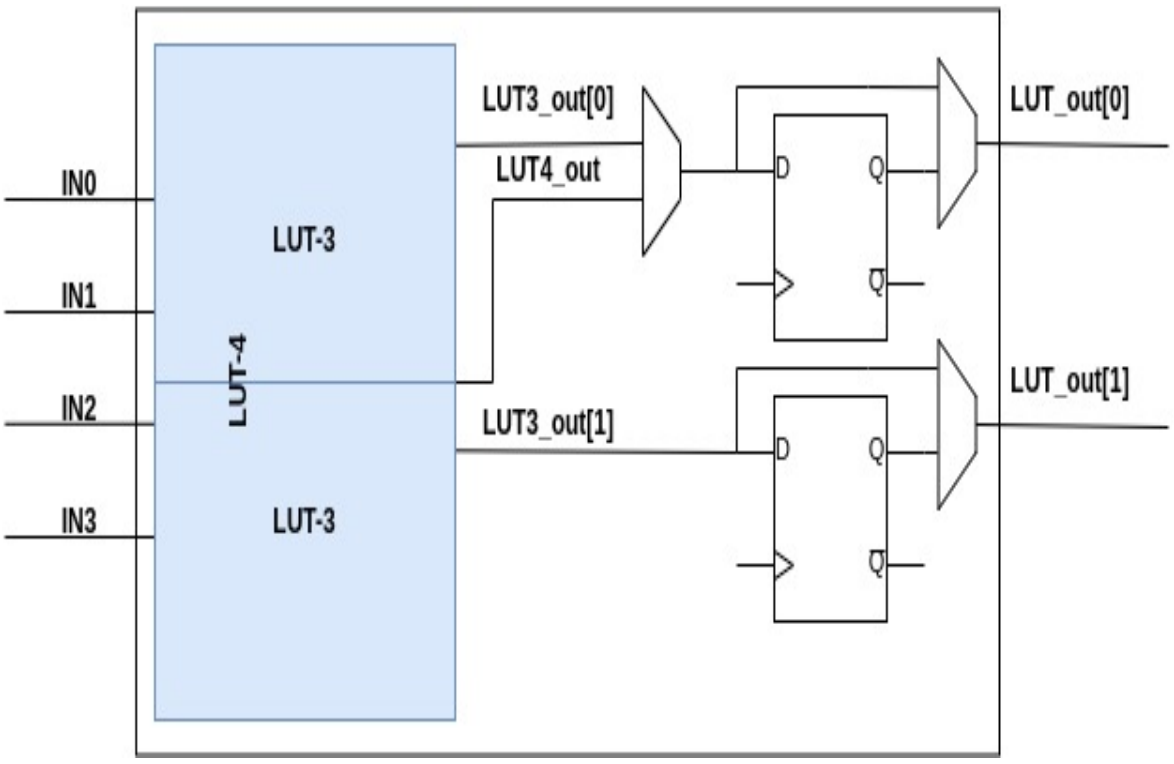

Figure 2.10. Fracturable LUT-4.

3. FPGA ARCHITECTURE EXPLORATION WITH VTR

In this chapter, the effect of logic block (CLB) architecture on FPGA performance and density will be studied in order to design an area-speed efficient FPGA. In particular, we investigate the impact of the size of the look-up table (LUT) in terms of K and the size of the CLB in terms of the number of LUTs per CLB which is N on the speed and logic density of an FPGA. Even though this issue was examined earlier in [5] and [6], both of which used the same experimental method developed by Rose and Tang, we felt compelled for a number of reasons to revisit the topic.

First, previous research developed a formula based on the smallest size of the transistor and minimum DRC distance to estimate routing structures area, yet we will follow a standard cell-based design flow. Therefore, utilizing the synthesis tool will allow us to achieve more accurate area results.

Second, previous research mainly concentrated on LUTs with fixed sizes (4-LUT, 6-LUT, etc.), but in this article, we will use fracturable-LUTs, which are a modern architectural improvement. Fracturable LUTs are already being used in commercial products, which results in higher levels of efficiency.

In this work, the twenty largest benchmark circuits from MCNC [7] have been used. These benchmarks are not the most recent suggestions for architectural exploration of architectures because they do not use any hard blocks; instead, they only use logic and registers. The fact that a homogenous FPGA that does not include any heterogenous hard IP blocks will be designed in this work makes those benchmarks a good fit for the purpose.

Because contemporary commercial products already use those sized LUTs, fracturable 4-LUTs and 6-LUTs will be concentrated. LUTs with a size greater than 7 increase the routing costs significantly. Finding the optimal size is the primary goal.

Rose has demonstrated that when the CLB size is greater than four, smaller LUTs (sizes 2 and 3) are almost as area-efficient as 4-input LUTs. This is the case even though the CLB size is greater. The speed performance of FPGAs with smaller LUT sizes is significantly worse (by almost a factor of 2) than the performance of FPGAs with larger LUT sizes. Therefore, the values for the K parameter in 4, 6, 8, and 10 will be investigated.

3.1. Verilog to Routing of MCNC Benchmarks with VTR8.0

The VTR tool will be used to find the optimal CLB size, out of the fracturable 4-LUT and 6-LUT options. VTR is a tool that places and routes a design in accordance with the resources of the FPGA that has been defined based on the architecture definition file and the Verilog design file that is provided as input. The tool was initially developed by Betz at the University of Toronto in the 1980s, and it has become the open source design tool that is most widely used in FPGA CAD research. The most recently released version of the tool is VTR8.0, which will be used in this work. The most significant advantage provided by the tool is that it makes it possible to rapidly define design ideas in FPGA architecture and immediately observe the effect on the performance of the design. Use of embedded VTR tool that is available in OpenFPGA rather than a standalone VTR flow was preferred in order to place and route the circuits since the OpenFPGA task configuration with multi-architectures is user-friendly and generates detailed synthesis and P&R reports.

The frac4_LUT and frac6_LUT are both swept with N = 4,6,8,10 as one of their design parameters. We are interested in observing their results. It is necessary for us to find the optimum value of the CLB input size (I) in order to fully utilize the capacity of the CLBs. A full crossbar structure is utilized by the local routing structure contained within the CLB. This structure connects the outputs and inputs of the CLB to the inputs of the LUT. In order to accomplish this, multiplexers are available at each LUT input for this. Because each LUT makes use of two DFFs, the value of output number is equal to two times N. In point of fact, it is predictable that if I=KxN is chosen, the

complete utilization of CLBs' capacities would be satisfied. However, Betz and Rose's research has demonstrated that CLBs can be utilized to their full capacity even when the value of I is lower than that of KxN. This is due to a number of factors, including the following:

- In order to save inputs, some of the inputs are feedback from the outputs of other LUTs that are contained within the same CLB.
- Some of the LUTs in the CLB share inputs with one another and some of the LUTs do not need all of their K-inputs to be used in order to function properly.

After plugging in I=2xN+2 for K=4, the researchers discovered that there was a 98% utilization rate. An equation similar to this one needs to be figured out for each possible value of K. Based on the findings of a number of scans presented in the mentioned article, the previously cited equation demonstrates that complete capacity utilization can be attained with a generalized average error coefficient of 10.1% [5]. The optimal CLB input number is expressed as

$$I = \left(\frac{K}{2}\right) \cdot (N + 1), \tag{3.1}$$

where K is LUT input number and N is LUT number in a CLB. As a result, the necessary input numbers (I) are listed in Table 3.1.

Table 3.1. CLB input numbers (I).

K/N	4	6
4	10	15
6	14	21
8	18	27
10	22	33

The K, I, and N values contained in the table served as the basis for our parametric generation of the necessary architecture definition files for VTR. A simplified content of VTR architecture file is shown in Figure 3.1.

```
</tile>
    <tile name="clb" area="${CLB_AREA}">
    <input name="I" num_pins="${I}" equivalent="full"/>
    <output name="O" num_pins="2*${N}" equivalent="none"/>
    <clock name="clk" num_pins="1"/>
    <pinlocations pattern="spread"/>
</tile>
<pb_type name="fle" num_pb="${N}">
    <input name="in" num_pins="${FRAC_LUT_IN}"/>
    <output name="out" num_pins="2"/>
    <clock name="clk" num_pins="1"/>
</pb_type>
<interconnect>
    <complete name="crossbar"
    input="clb.I fle[${N}-1:0].out"
    output="fle[${N}-1:0].in"/>
    <complete name="clks" input="clb.clk"
    output="fle[${N}-1:0].clk"/>
</interconnect>
```

Figure 3.1. VTR architecture file content.

MCNC's benchmark netlists were placed and routed across eight different architectures that are automatically generated.

3.2. The Results of the Experimental Area with Varying K and N

The channel width (W) is one of the important design parameters that need to be determined in order to see the area results. With the use of the VTR tool, the minimum number of channels that are necessary for routing the netlist onto FPGA resources was figured out. The specifics of this routing are described in the background chapter. If the user does not provide the channel width number as an input which is one of the

VTR options, the VTR will determine, through a process of iteration, the required channel widths. However, due to the nature of these iterations, the flow takes longer to complete than it would with a certain channel width. The channel width numbers were collected in a single file from the report file named vprstdout.log, which was produced separately for each benchmark with a basic Python script. Figure 3.2 provides a part of the file where the data were gathered for the k4n4 architecture.

```
Circuit successfully routed with a channel width factor of 38.
Circuit successfully routed with a channel width factor of 40.
Circuit successfully routed with a channel width factor of 46.
Circuit successfully routed with a channel width factor of 38.
Circuit successfully routed with a channel width factor of 50.
Circuit successfully routed with a channel width factor of 44.
Circuit successfully routed with a channel width factor of 38.
```

Figure 3.2. Minimum routable channel widths report.

On the basis of these findings, a 20% increase in the smallest channel width number that can successfully route all benchmarks for each architecture was determined as a fixed channel width number, and it was fixed in the subsequent run. Therefore, while comparing the area efficiency of the architectures, the goal was to provide a more reasonable comparison environment by ensuring that all benchmarks can be implemented. This was done with the intention of providing a clearer picture of the relative merits of each architecture, since there is a tendency to avoid high-stress situations in the interest of achieving healthier outcomes during the design process. Table 3.2 displays the minimum channel width numbers for each architecture.

Table 3.2. Channel width numbers.

k4_n4	k4_n6	k4_n8	k4_n10	k6_n4	k6_n6	k6_n8	k6_n10
70	80	90	90	80	90	80	80

After that, a field programmable gate array (FPGA) with a grid size of 2x2 was generated using the channel width numbers found for each architecture. During the synthesis, Design Compiler which is the synthesis tool included in the Synopsys Design Tools was utilized. The technology utilized in this work is 250nm CMOS technology.

Synthesizable netlists generated by OpenFPGA were utilized in order to ensure that the synthesis results were comparable to one another. No timing constraint entry has been made. The following describes the calculation model for the synthesis results carried out on FPGAs with a grid size of 2x2.

Tileable architecture is generally preferred by modern FPGAs. The reason for this is that the required manual layout effort can be concentrated on a smaller number of tiles, and the same tiles can be used repeatedly throughout the structure. This reduces the amount of time and effort needed to layout the structure. All of the necessary repetitive tileable structures, which are included in other grid-sized FPGAs, such as CLB, connection blocks and switch blocks (CB-SB), are also available in FPGAs with a grid size of 2x2. Figure 3.3 shows tileable structures that can be implemented in an FPGA.

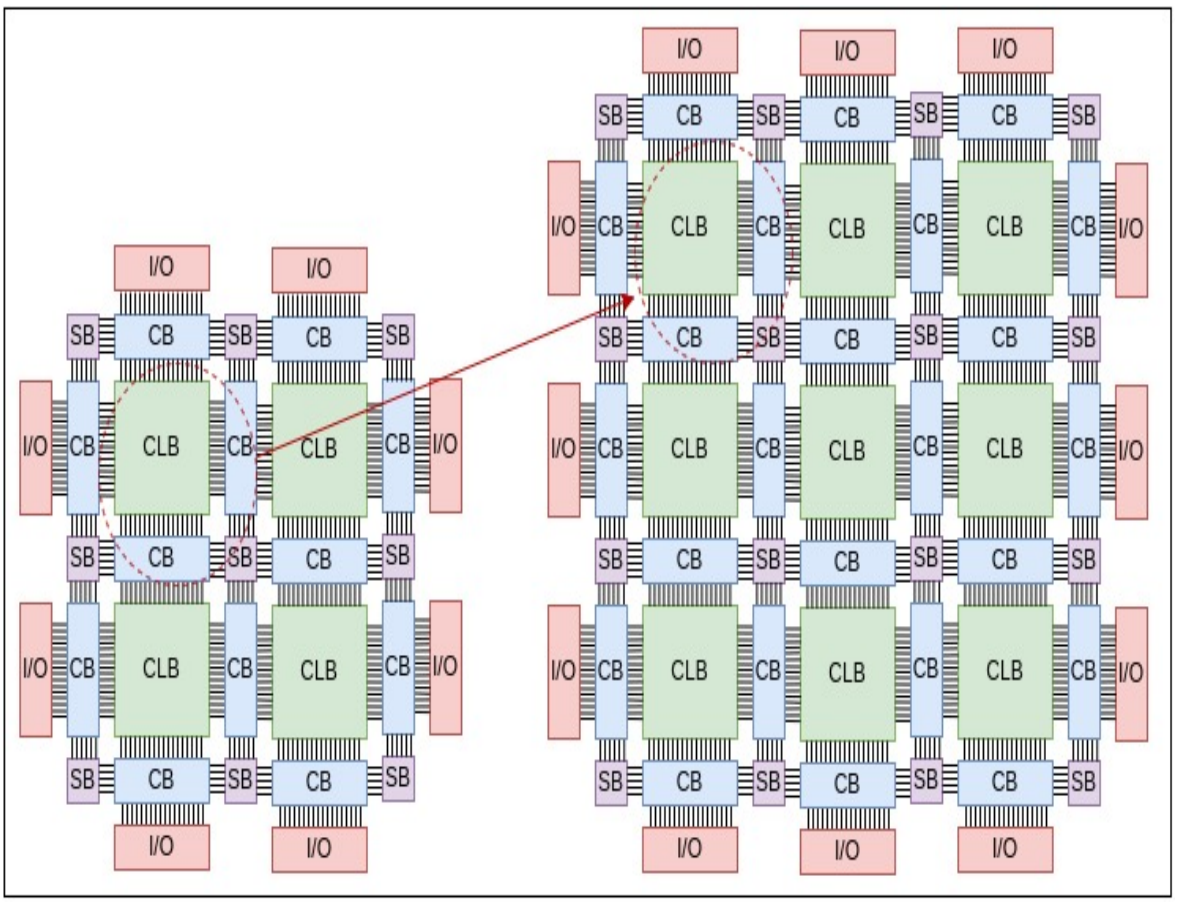

Figure 3.3. Tileable FPGA layouts.

The area information, measured in μm^2 units, for each block that makes up the synthesized 2x2 FPGA is presented in Table 3.3.

Table 3.3. Areas of all tile blocks.

	k4_n4	k4_n6	k4_n8	k4_n10	k6_n4	k6_n6	k6_n8	k6_n10
CLB	51310.688	85559.69	126064.3	189001.9	108627.5	187935.3	282468	380439.3
CX10	17216	19494.75	24723.75	26199.5	17807	25994.5	29492.5	30698.5
CX11	10230	14056.83	19223.42	22694.33	13418	22182.5	28636	31705.5
CX12	16967.75	19270.75	24425.75	25703.5	19350.5	25722	31154	30339
CY01	15465	17720.25	22649.75	24252.5	17919	23961.5	29182	29032
CY11	8462.3333	12575.75	17537.83	20665.75	12118.5	20213	27261.5	30225.5
CY21	17081.25	19258.25	24184.5	25537	17861	25328.5	29543.5	30703
SB00	31565	34221	39543	40451	30535	37518	43549	39811
SB01	37330	40955.67	49443.67	51579	36862	46692	56229	50940
SB02	30959	33806	39041	40330	30518	37371	42796	39794
SB10	36885.667	41557.67	50196	51506	36524	47842	55942	51138
SB11	48295.667	54141.22	64980.78	65591.33	48148	63992	71421	64944
SB12	37512.667	41151.67	49795.67	51382.33	36777	46856	55831	50991
SB20	31002	33979	37977	39422	30613	36566	42727	39569
SB21	37120	41142.67	49721	51213.33	36518	46917	55720	50811
SB22	31582	34403	40702	41758	30665	38790	43238	41325

Below, the parameterized area formulas that are used while observing the area efficiencies of different FPGA architectures are listed as

$$\text{Logic Area} = N^2 CLB, \tag{3.2}$$

$$\begin{aligned}\text{Routing Area} = {} & N^2(CX11 + CY11 + SB11) \\ & + N(CX10 + CX12 + CY01 + CY21 + SB10 \\ & \text{S}B01 + SB21 + SB12 - CX11 - CY11 - 2SB11) \\ & + (SB00 + SB20 + SB02 + SB22 - SB10 - SB01 \\ & \text{-}SB21 - SB12 + SB11),\end{aligned} \tag{3.3}$$

$$\text{Total Area} = \text{Logic Area} + \text{Routing Area}. \tag{3.4}$$

Table 3.4. Benchmark layouts table.

	k4_n4	k4_n6	k4_n8	k4_n10	k6_n4	k6_n6	k6_n8	k6_n10
alu4	15x15	12X12	10X10	9X9	11X11	9x9	8X8	7X7
apex2	16x16	13X13	11X11	10X10	13X13	11x11	9X9	8X8
apex4	16x16	13X13	11X11	10X10	13X13	10x10	9X9	8X8
bigkey	17x17	14X14	14X14	14X14	14X14	14x14	14X14	14X14
clma	34x34	27X27	23X23	21X21	25X25	20x20	18X18	16X16
des	19x19	16X16	16X16	16X16	16X16	16x16	16X16	16X16
diffeq	13x13	11X11	9X9	8X8	11X11	9x9	8X8	7X7
dsip	16x16	14X14	14X14	14X14	14X14	14x14	14X14	14X14
elliptic	24x24	19X19	16X16	15X15	21X21	17x17	15X15	13X13
ex5p	18x18	14X14	12X12	11X11	14X14	11x11	10X10	8X8
ex1010	12x12	9X9	8X8	7X7	9X9	7x7	6X6	5X5
frisc	26x26	20X20	18X18	16X16	21X21	17x17	15X15	13X13
misex3	13x13	10X10	9X9	8X8X	9X9	8x8	7X7	6X6
pdc	26x26	20X20	18X18	16X16	20X20	16x16	14X14	13X13
s298	3x3	2X2	2X2	2X2	2X2	2x2	2X2	2X2
s38417	28x28	23X23	20X20	18X18	23X23	19x19	16X16	14X14
s38584	29x29	24X24	20X20	18X18	22X22	18x18	15X15	14X14
seq	17x17	13X13	12X12	11X11	13X13	11x11	9X9	8X8
spla	25x25	19X19	17X17	15X15	20X20	16x16	14X14	12X12
tseng	14x14	12X12	10X10	9X9	11X11	9x9	8X8	7X7

In Table 3.4, each of the required layout grid sizes is presented in its own column. While calculating the area, more accurate results were provided by scaling the area values with the percentage of the CLBs used in the given layout. For example, for a benchmark that is mapped onto a 15x15 layout that uses only 215 CLBs instead of the full 225 CLBs in the FPGA, the utilization rate is 95.55%, and the calculated area is multiplied by 0.955.

The findings that were gathered demonstrate that as the CLB size grows, the required FPGA layout grid size gets smaller. On the other hand, the total FPGA area will be clarified by the sum of the logic area and the routing area with an expected decrease. Figure 3.4 and Figure 3.5 illustrate the required Logic areas for each architecture and benchmark. Figure 3.6 and Figure 3.7 demonstrate the required Routing areas for each architecture and benchmark.

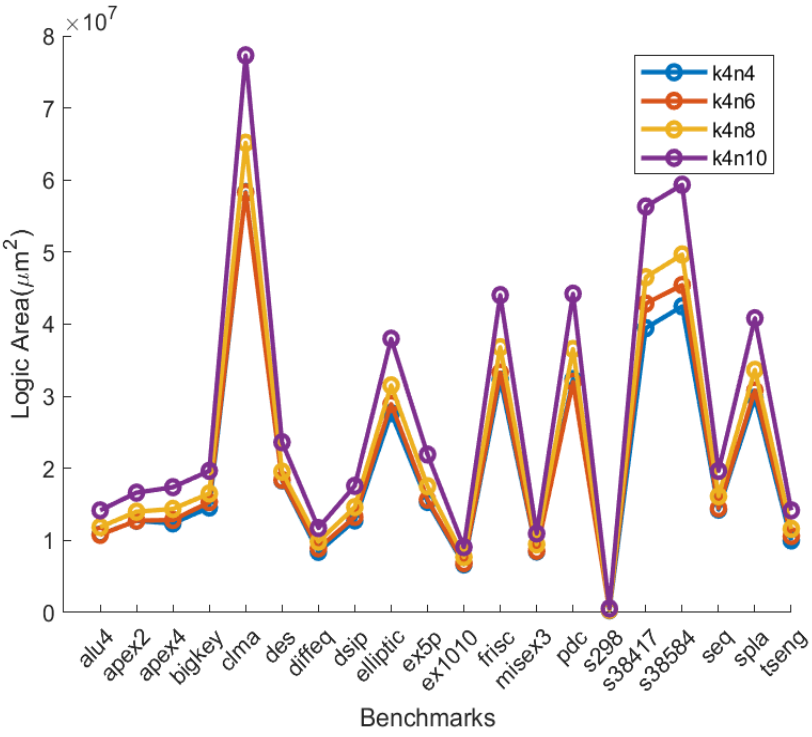

Figure 3.4. K4 logic areas.

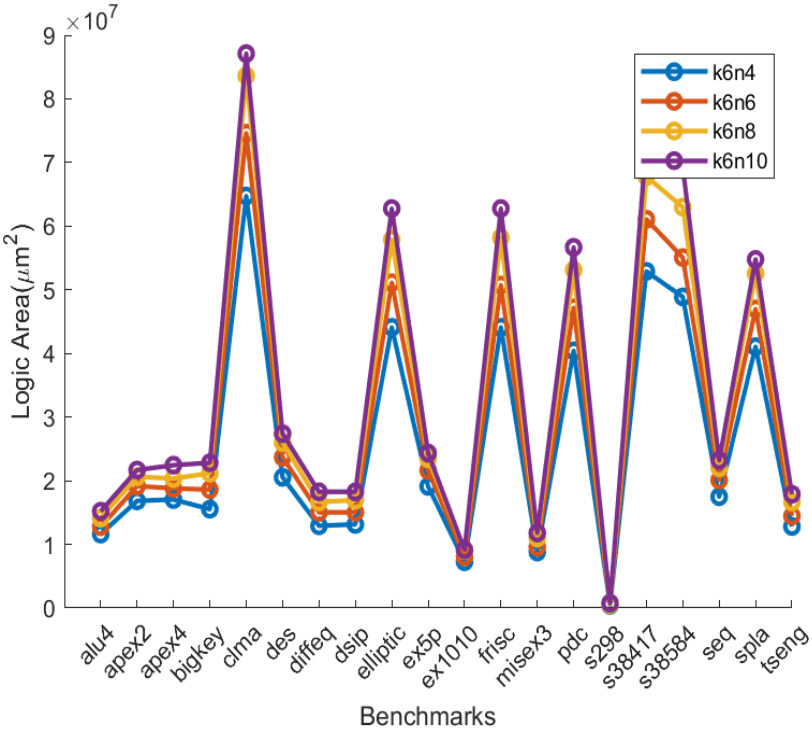

Figure 3.5. K6 logic areas.

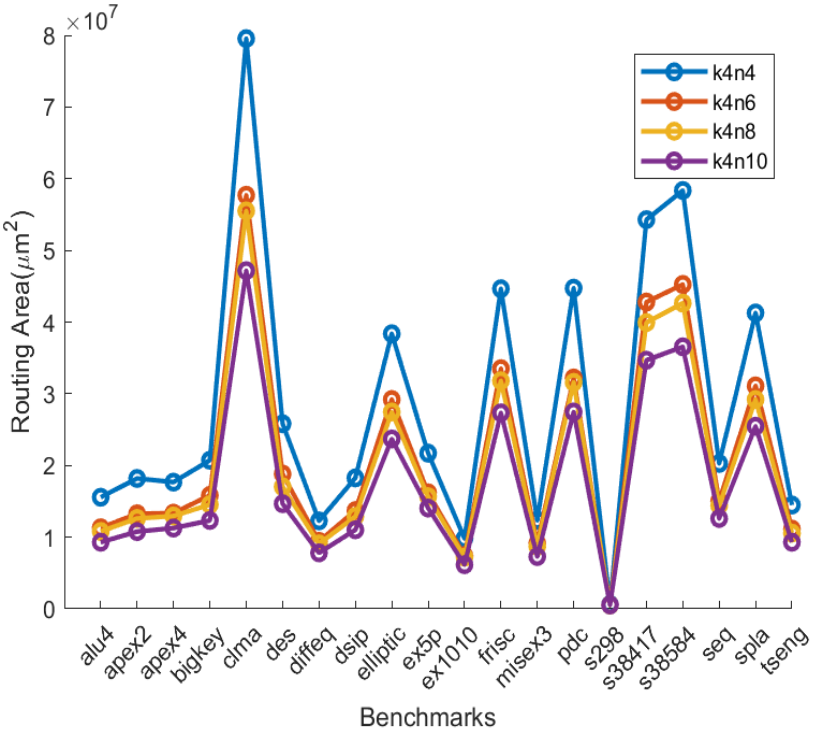

Figure 3.6. K4 routing areas.

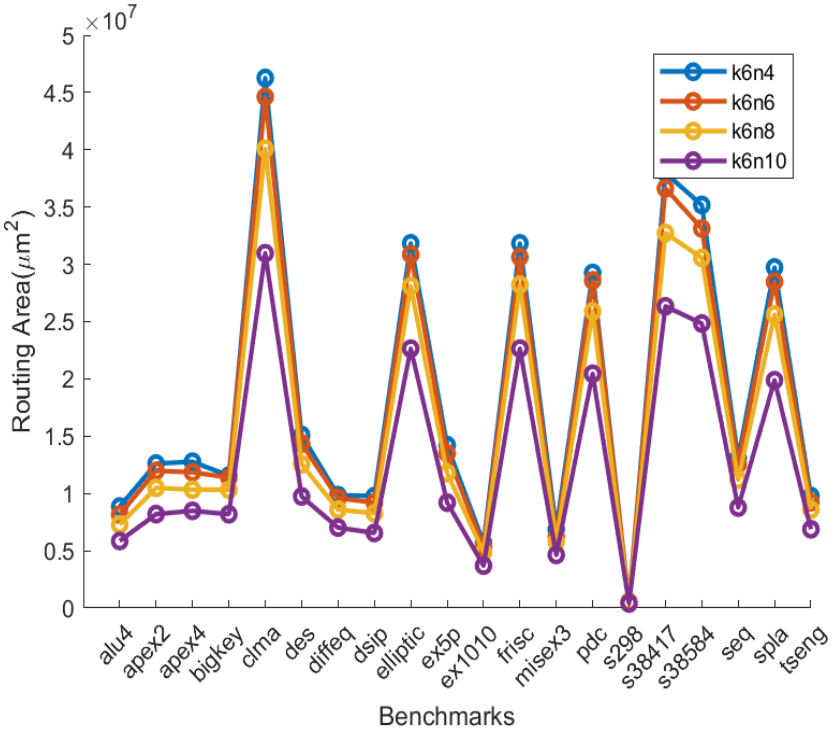

Figure 3.7. K6 routing areas.

Even if the grid layout size and the routing area decrease, the logic area increases as N gets higher. It can be observed how the total area is affected as well as the increase/decrease in the logic and routing areas by looking at the graph, which includes mean values in Figure 3.8 and Figure 3.9. The structures with frac_4-LUT, when the increase in the logic area for the k4n6 structure is compared to the decrease in the routing area, the routing area is significantly greater. The growth in the logic area is sufficient to compensate for the reduction in the routing area for N, after the number 8. Since the number of multiplexers required for local routing architecture and the input number of each multiplexer increases with N, the growth of the area of tree-like structured multiplexers is exponential. On the other hand, in structures that make use of frac_6-LUT the routing area does not decrease by a significant amount, but the logical area grows quickly.

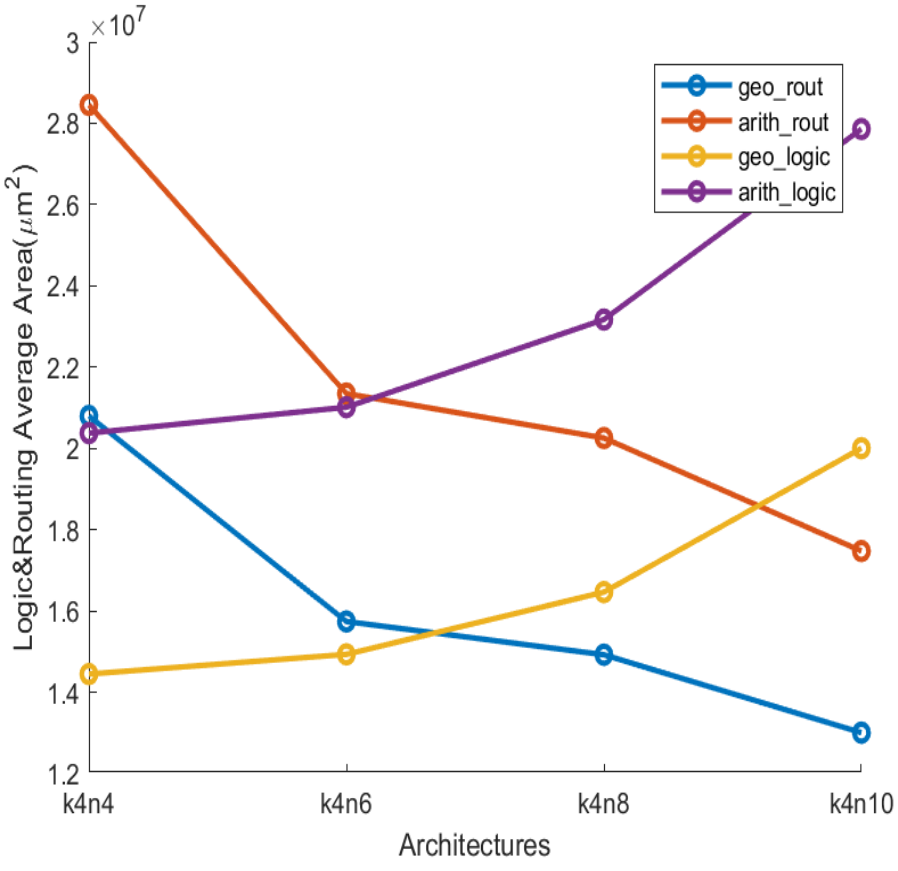

Figure 3.8. K4 logic and routing mean areas.

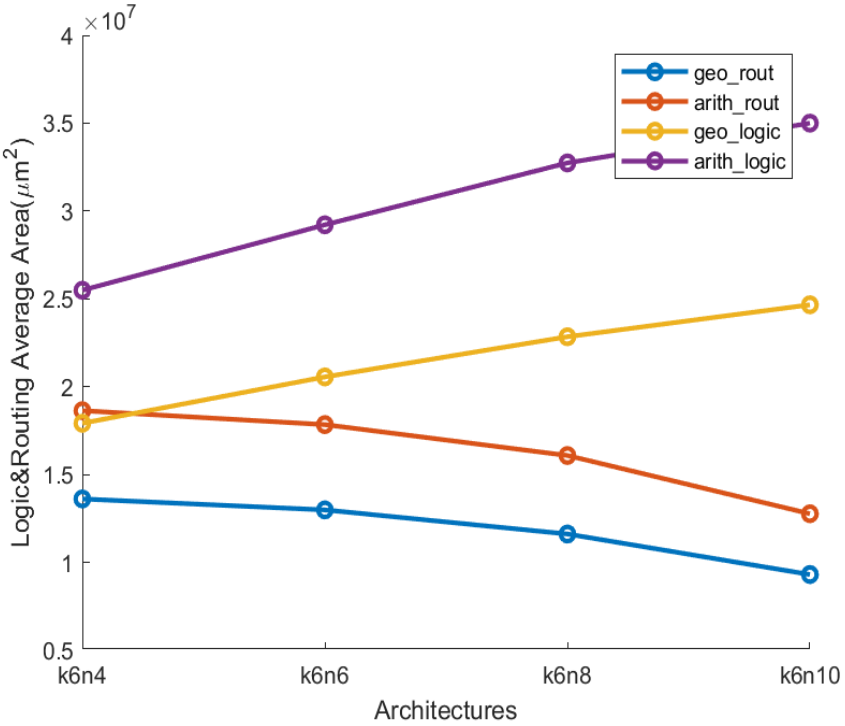

Figure 3.9. K6 logic and routing areas.

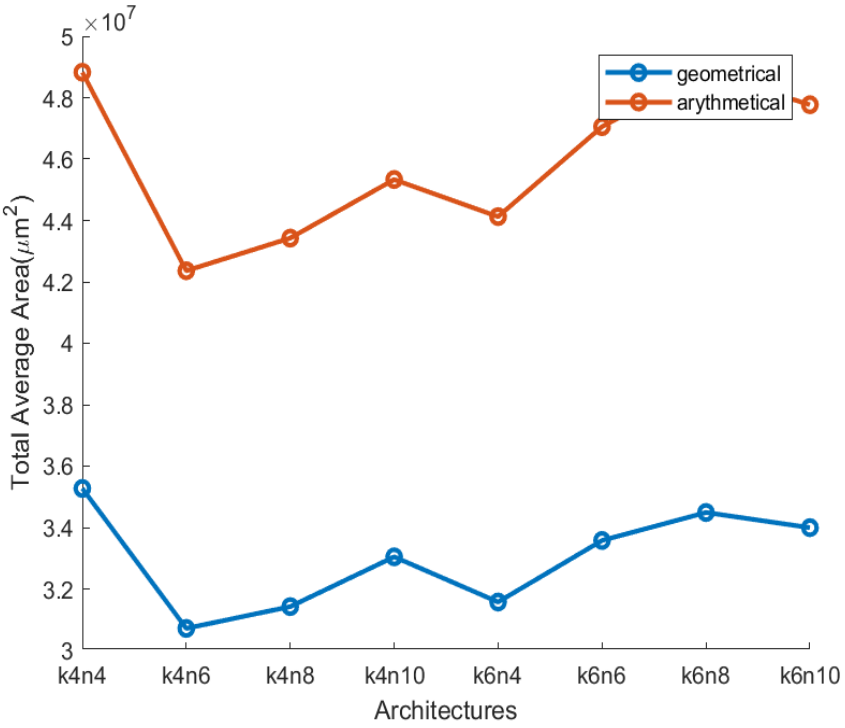

Figure 3.10. Total mean areas.

By examining the Average Total Area values in Figure 3.10, it is possible to conclude which CLB size will give the best area results in FPGA. k4n6 architecture provides the best value in terms of its area efficiency. However, k4n4, k4n8, and k6n4 architectures are comparable to one another in terms of area. It is necessary to take into consideration not only the performance in terms of area but also the performance in terms of speed. Before beginning the layout design process, it would be reasonable to first synthesize the FPGA netlists required for each architecture in flattened form under equal conditions. After that, one could compare the post-synthesis simulation results of a medium-sized design for speed performance observation. The tool and computer resource requirements of the flat synthesis workload grow exponentially with the growth of the structure, which is why a design with a medium-sized structure is preferred. In addition to this, the goal was to ensure that the design did not end up being too simplistic to produce misleading results. The size needed for this purpose is provided by a UART design consisting of a UART TX and UART RX. The inputs and outputs of this design are a bus, which results in more reliable delay values. Operating frequencies and combinational delays for the design's various output ports are detailed in Table 3.5.

Table 3.5. Delay table.

	k4_n4	k4_n6	k4_n8	k4_n10	k6_n4	k6_n6	k6_n8	k6_n10
Tx_Active	17.1	16.1	17.5	17.4	16.3	18.2	16	20.7
Tx_Serial	16.6	15.8	19.8	20	19.2	18.5	15.4	18.4
Tx_Done	18.4	16.9	18.5	20.2	18.8	16.4	19.6	24.7
Rx_DV	17.3	16.3	18.5	20.2	18.8	16.4	19.6	24.7
Rx_Byte	16.4	15.7	17.6	19.8	17.5	15.2	18.6	23.5
delay_average	17.16	16.16	18.38	19.52	18.12	16.94	17.84	22.4
clk_period	43	39	39.5	41	40	39	41	41

By analyzing the results of the area x delay products, it will be possible to determine which architecture is the most suitable. The results of the area x delay multiplication are displayed in Figure 3.11. Therefore, the K4N6 architecture is the best option when considering both speed and performance.

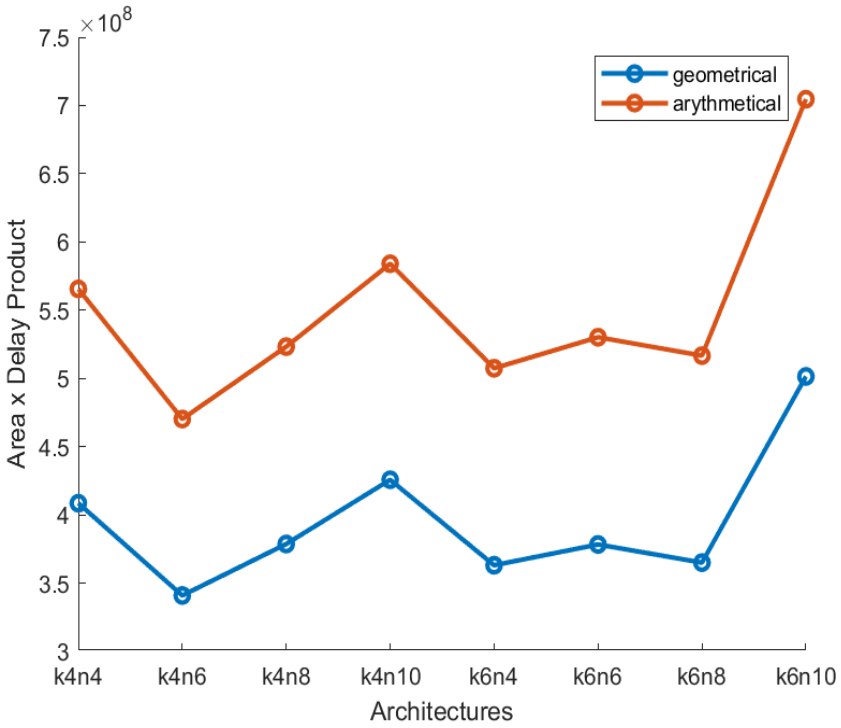

Figure 3.11. Area x delay product.

4. FPGA LAYOUT DESIGN

It was observed in Chapter 3 that the architecture which gave the best results in terms of area x speed product was k4n6, therefore FPGA was designed with this architecture. In this chapter, layout and physical design will be performed, and issues will be examined in detail. The target design was chosen as a homogeneous 20x20 grid-sized FPGA. To achieve this, all the necessary sub-modules in the 2x2 grid-sized FPGA will be implemented separately as macros and scaled to a 20x20 structure with macro placement. When it comes to layout design, it will be necessary to focus on the details of the FPGA netlist. The FPGA netlist will be generated using OpenFPGA tool.

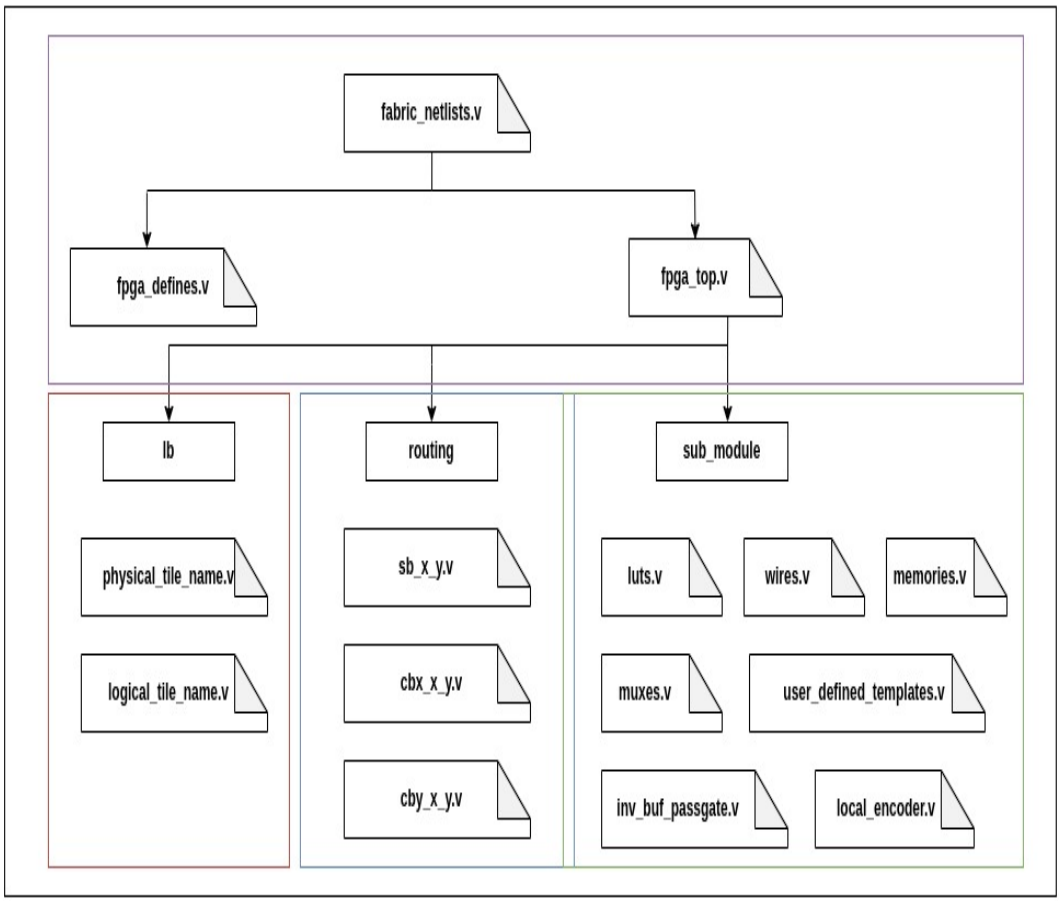

Figure 4.1. OpenFPGA netlist organisation.

4.1. OpenFPGA Netlist Organisation

The Verilog netlists which are automatically generated by the OpenFPGA are organized as illustrated in Figure 4.1 [2].

4.1.1. Top-level Netlists

This directory includes all Verilog modules used to model FPGA top module:

- Fabric_netlists.v: This file includes all the Verilog design modules that are used in FPGA top module.
- Fpga_top.v: This is the top module of FPGA.
- Fpga_defines.v: This file contains simulation related commands.

4.1.2. Logic Blocks

This directory includes all Verilog modules used to model CLBs:

- Physical_tile_name.v: A Verilog netlist which models the architecture definitions in VTR.
- Logical_tile_name.v: A Verilog netlist which models the internal of architecture definitions in VTR.

4.1.3. Routing Blocks

This directory includes all Verilog modules modeling Switch Blocks (SBs) and Connection Blocks (CBs):

- Sb_row_column.v: A Verilog netlist will be produced for each distinct Switch Block (SB). The Switch Block's coordinates in the FPGA fabric are indicated by the row and column.
- Cb_row_column.v: A Verilog netlist will be generated for each distinct X and Y direction Connection Block (CBXY).

4.1.4. Primitive Modules

This directory includes all Verilog modules that are building blocks of Logic Blocks and routing resources such as SBs and CBs:

- Luts.v: Verilog module which models the LUT
- Wires.v: Verilog modules for direct connections
- Memories.v: Verilog modules including configurable flip flop or SRAM structures
- Muxes.v: Verilog modules which model all multiplexers
- Inv_buf_passgate.v: Verilog modules including inverters and buffers
- Local_encoder.v: Verilog modules which model all encoders and decoders in FPGA

4.2. FPGA Structures to be Designed

There are two structures that make up the netlist. The configuration protocol and multiplexers of various sizes that are forming LUT and routing resources. The design of multiplexers will directly affect the speed performance of the circuit. It is estimated that the area cost will also be affected. The choice of configuration protocol will not affect the operating frequency of the circuit. However, it is predicted that the configuration protocol preference, which the circuits configure the bits of all the structures in the FPGA will seriously affect the area cost of the circuit.

4.2.1. Configuration Protocol

Now let's take a closer look at the protocol options OpenFPGA offers us for FPGA configuration. The circuitry used to program an FPGA is known as the configuration protocol. Depending on the application context, the configuration protocol in FPGAs could be very varied as an interface. Different configuration protocols are supported by OpenFPGA for configuring FPGA fabrics [2]:

- Scan_chain: A chain of reconfigurable memory is connected. A FPGA is programmed using serial bitstream loading.
- Frame_based: Frames are used to organize configurable memories. A frame of reconfigurable memory is thought to be represented by each module in an FPGA fabric, such as the Configurable Logic Block (CLB), Switch Block (SB), and Connection Block (CB). Through an address decoder, all memory banks are accessed within each frame.
- Ql_memory_bank: Configurable memories are arranged in an array, with each element having a different address that the BL/WL decoders can use to access it. The BL/WLs are effectively shared by programmable blocks per column and row in this memory bank organization.
- Standalone: Direct access to configurable memories is made possible by FPGA fabrics' ports. To put it another way, there are no protocols in place to manage the memories. Engineers can fully customize the configuration protocol thanks to this. It is not advised using standalone when developing an FPGA chip because it will result in a significant increase in the number of I/Os needed.

Major FPGA configuration protocols offered by OpenFPGA are listed above. For each protocol suitable for standard cell based design, netlists will be generated by OpenFPGA and synthesized with Design Compiler tool, and the protocol with the highest area efficiency will be preferred by examining area cost reports.

Scan chain protocol consists of one input port and one output port in FPGA and flip-flop chain connected back to back across the circuit. Configuration bits from the input port are loaded sequentially during each clk edge. The design of the flip-flop cell to be used in this protocol will greatly affect the area of the circuit because there are thousands of scan chain flip-flops in a 20x20 grid sized FPGA. One of the advantages of this protocol is that a bit sequence scanned from the chain input will be read from the output after production, so it can be determined that there is no break in the flip-flop chain, and a primitive DFT method will be applied in the FPGA. Related OpenFPGA architecture file part for the scan chain protocol is shown in Figure 4.2.

```
<circuit_model type="ccff" name="DFF" prefix="DFF"
   verilog_netlist="/verilog/dff_YITAL.v">
    <design_technology type="cmos"/>
    <port type="input" prefix="D" size="1"/>
    <port type="output" prefix="Q" size="1"/>
    <port type="clock" prefix="prog_clk" lib_name="SAAT"
       size="1" is_global="true"/
</circuit_model>
<configuration_protocol>
    <organization type="scan_chain" circuit_model_name="DFF
       "/>
</configuration_protocol>
```

Figure 4.2. OpenFPGA architecture file content for DFF.

As the configuration protocol changes in the generated netlist, the details of the instantiated cells in the memory.v are shown in Figure 4.3. Port ccff_head is the input and ccff_tail port is the output port of the chain. In the Verilog module we can see the hierarchical scan chain piece generated for a submodule. A long FF chain goes around every part of the circuit. The layout of the DFF cell is shown in Figure 4.4. Cell's dimensions are $8.32 \times 16.64\,\mu m^2$, totaling $138.44\,\mu m^2$.

```
DFF DFF_0_ (
    .SAAT(prog_clk),
    .D(ccff_head),
    .Q(mem_out[0]));

DFF DFF_1_ (
    .SAAT(prog_clk),
    .D(mem_out[0]),
    .Q(mem_out[1]));

DFF DFF_2_ (
    .SAAT(prog_clk),
    .D(mem_out[1]),
    .Q(mem_out[2]));

DFF DFF_3_ (
    .SAAT(prog_clk),
    .D(mem_out[2]),
    .Q(mem_out[3]));
```

Figure 4.3. Verilog implementation of scan chain with DFF.

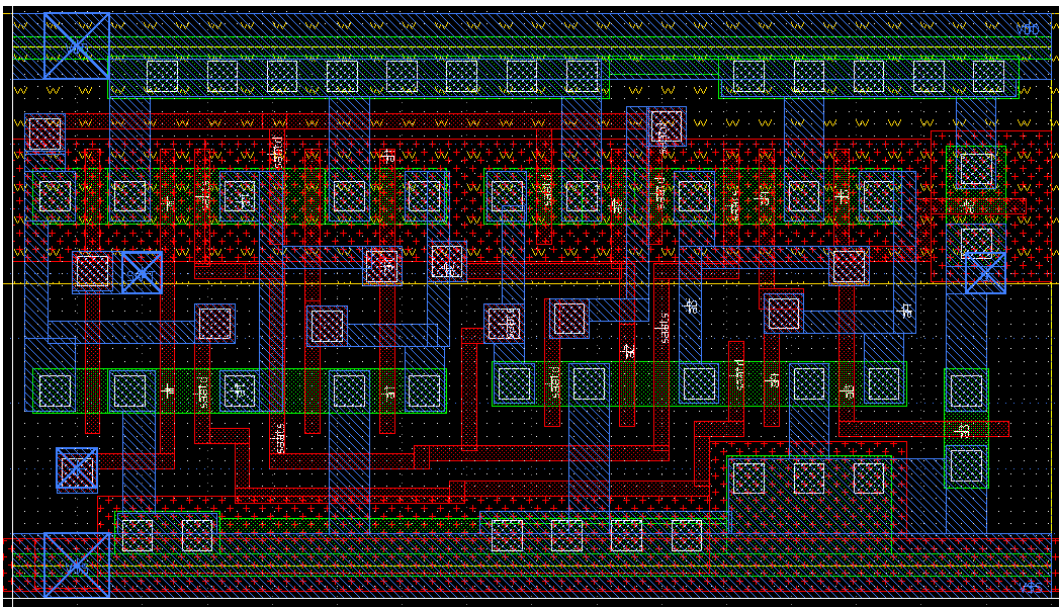

Figure 4.4. Layout of DFF cell.

The sizes of all blocks from the synthesis results of a 2x2 grid-sized FPGA are listed in Table 4.1. Accordingly, area of a 20x20 grid-sized FPGA can be calculated by scaling.

Table 4.1. Table of sizes with DFF cell.

CLB	**CX10**	**CX11**	**CX12**	**CY01**	**CY11**	**CY21**	**SB00**
86404.25	24798	13587	13780.5	15286.5	13681	23266.5	30319
SB01	**SB02**	**SB10**	**SB11**	**SB12**	**SB20**	**SB21**	**SB22**
40884	26287	40885	54586	40131	34524	41481	30267
	logic	**routing**	**total**	**size**			
20x20	34561700	34835652	69397352	8.330507			

The total area cost of a 20x20 grid sized FPGA is $69.4e6\,\mu m^2$, excluding IO pads, and $8.3\,mm^2$ when aspect ratio 1 is taken. It should be noted that these values are values that do not take into account the utilization ratio. A more space-optimized DFF design will improve the area cost of the FPGA, ignoring the speed performance of the flip-flop used. Although the configuration frequency will decrease somewhat as the new DFF speed decreases, this will not affect the operating frequency of the design mapped in the FPGA.

To reduce the area overhead of the conventional design, the two feedback transmission gates can be eliminated as shown in Figure 4.5. The design has been optimized by adding a pull-down structure to compensate for the performance and power losses due to the removal of feedback transmission gates. Minimum sized transistors are used in the design.

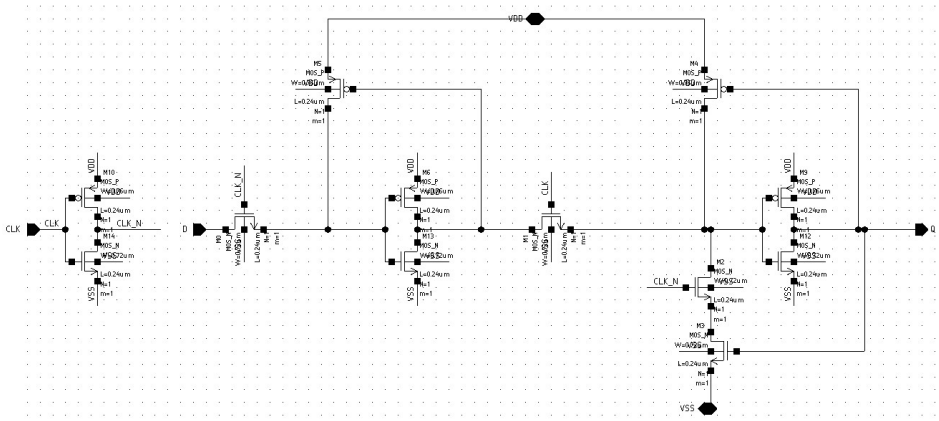

Figure 4.5. Schematic of new DFF cell.

The layout of the circuit is shown in Figure 4.6. In the new design obtained, the area of the circuit is $8.32 \times 11.44\,\mu m^2$, a total of $95.18\,\mu m^2$.

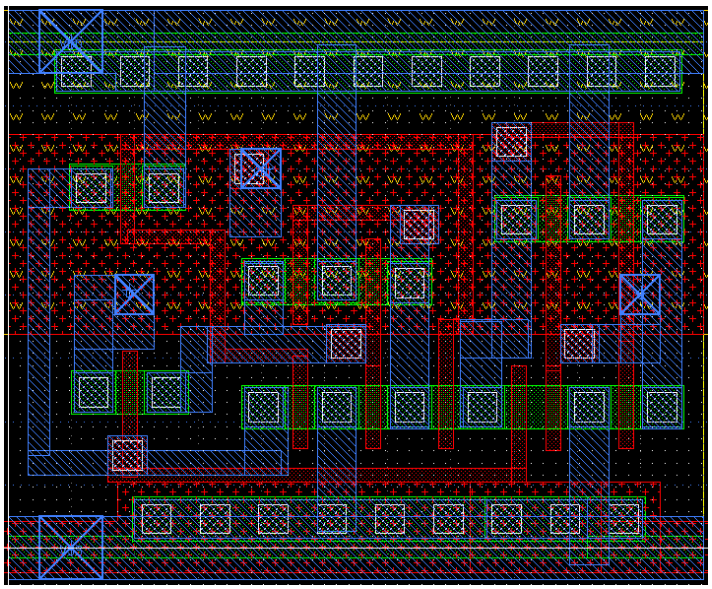

Figure 4.6. Layout of new DFF cell.

The resulting area difference of 43.26 μm^2 results in an area gain of approximately $657 \times 657\,\mu m^2$ for 10000 flip-flops. Below, the related architecture definition part of OpenFPGA for the newly designed flip-flop cell is shown in Figure 4.7 .

```
<circuit_model type="ccff" name="DFF_PS" prefix="DFF_PS"
   verilog_netlist="/verilog/dff_YITAL.v">
    <design_technology type="cmos"/>
    <port type="input" prefix="D" size="1"/>
    <port type="output" prefix="Q" size="1"/>
    <port type="clock" prefix="prog_clk" lib_name="CLK"
       size="1" is_global="true"/>
</circuit_model>
<configuration_protocol>
    <organization type="scan_chain" circuit_model_name="
       DFF_PS"/>
</configuration_protocol>
```

Figure 4.7. OpenFPGA architecture file content for new DFF.

The sizes of all blocks from the synthesis results of a 2x2 grid sized FPGA are listed in Table 4.2. Accordingly, area of a 20x20 grid sized FPGA can be calculated by scaling.

Table 4.2. Table of sizes with new DFF cell.

CLB	**CX10**	**CX11**	**CX12**	**CY01**	**CY11**	**CY21**	**SB00**
77085	22604.5	12531	12777	14023	12686	21397.5	24089
SB01	**SB02**	**SB10**	**SB11**	**SB12**	**SB20**	**SB21**	**SB22**
34100	20818	34231	48252	33486	27567	34845	24071
	logic	**routing**	**total**	**size**			
20x20	30834000	31110595	61944595	7.870489			

Although the use of D flip-flop chain as a configuration protocol has several advantages, it can still cause area overhead. The results of other protocol alternatives should also be observed. For the Standalone protocol, OpenFPGA developers do not recommend its use during FPGA chip design, since every SRAM cell requires a separate I/O pad. However, during the eFPGA design, it will be possible for the designer to design a new interface between the processor and the FPGA. Therefore, the standalone protocol was excluded in this work.

The ql_memory_bank protocol, on the other hand, makes it possible to use SRAM macros synthesized from the memory compiler in theory, as it generates netlists with BL/WL lines. However, not every compiler supports the synthesis of a wide variety of small sizes (2x4, 3x2 etc.) memory macros for each submodule.

When a standalone SRAM cell is designed with the frame protocol, it is possible to examine the synthesis results of this protocol. We can compare the local decoder overhead that will configure the SRAM cells in each submodule to the use of the DFF chain by obtaining the synthesis results.

The commonly used SRAM cell consists of 6 transistors (6T). It consists of 2 inverters where the output of each inverter is fed as input into other and two access MOSFETs. In order to use the standalone SRAM cell, WE input that enables the Access transistors should be added in the design. During the design, attention was paid to the sizing of the MOSFETs, because transistors must satisfy certain dimensional limitations to ensure both read and write stability. Additionally, in order to attain good layout density, transistors must be designed to be as small as possible. In general, driver transistors must be stronger than access transistors (readability) and access transistors should prevail against pull-up transistors (writability). SRAM cell schematic and layout are shown in the Figure 4.8 and Figure 4.9 below.

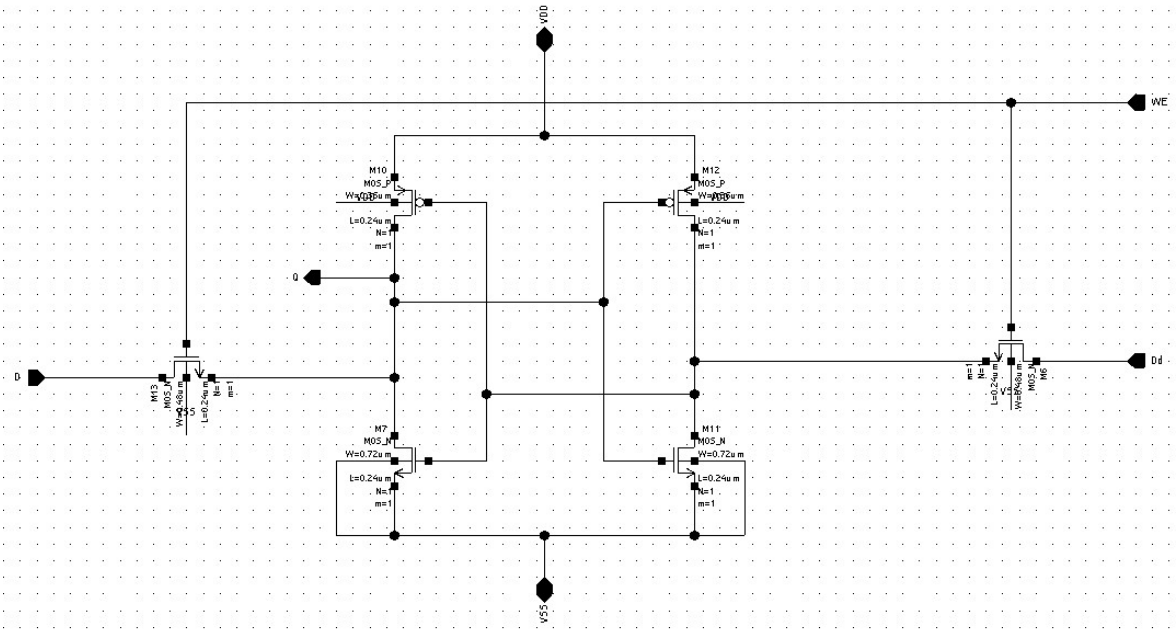

Figure 4.8. Schematic of new SRAM cell.

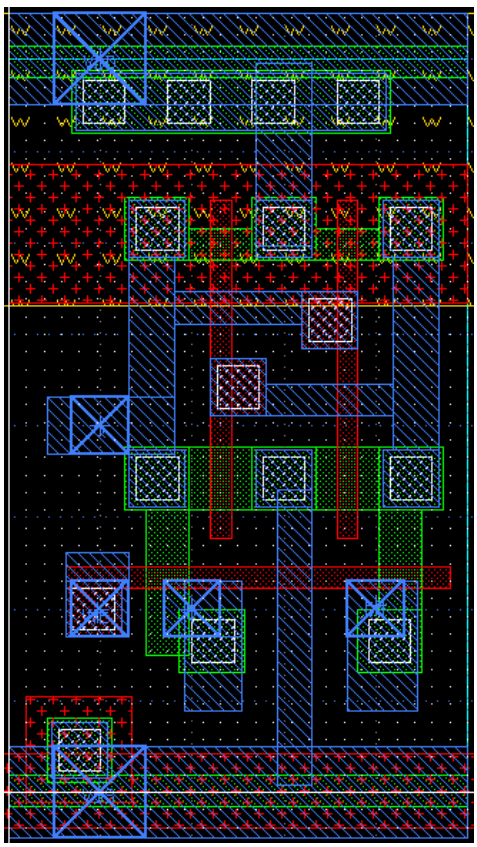

Figure 4.9. Layout of new SRAM cell.

The area occupied by the SRAM cell is 43.26 μm^2. There is an area gain compared to flip flops. However, since a chain can not be created one after the other like DFFs, it should be considered together with the area loss that will be brought by the decoder structures required for the configuration of SRAM cells. Related OpenFPGA architecture file part for frame protocol with SRAM is shown in Figure 4.10.

```
<circuit_model type="sram" name="SRAM" prefix="SRAM"
   verilog_netlist="/verilog/sram.v">
    <design_technology type="cmos"/>
    <port type="bl" prefix="bl" lib_name="D" size="1"/>
    <port type="wl" prefix="wl" lib_name="WE" size="1"/>
    <port type="output" prefix="out" lib_name="Q"size="1"/>
</circuit_model>
<configuration_protocol>
    <organization type="frame_based" circuit_model_name="
       SRAM"/>
</configuration_protocol>
```

Figure 4.10. OpenFPGA architecture file content for SRAM.

As the configuration protocol changes in the generated netlist, the details of the instantiated cell and configuration memory structure details in the memory.v sub module can be examined in Figure 4.11.

```
decoder2to4 decoder2to4_0_ (
    .enable(enable),
    .address(address[0:1]),
    .data_out(decoder2to4_0_data_out[0:3]));

SRAM SRAM_0_ (
    .D(data_in),.Dd(~data_in),
    .WE(decoder2to4_0_data_out[0]),
    .Q(mem_out[0])
    );

SRAM SRAM_1_ (
    .D(data_in),.Dd(~data_in),
    .WE(decoder2to4_0_data_out[1]),
    .Q(mem_out[1])
    );

SRAM SRAM_2_ (
    .D(data_in),.Dd(~data_in),
    .WE(decoder2to4_0_data_out[2]),
    .Q(mem_out[2])
    );

SRAM SRAM_3_ (
    .D(data_in),.Dd(~data_in),
    .WE(decoder2to4_0_data_out[3]),
    .Q(mem_out[3])
    );
```

Figure 4.11. Verilog implementation of frame protocol with new SRAM cell.

The sizes of all blocks from the synthesis results of a 2x2 grid sized FPGA for the Frame protocol are listed in Table 4.3. Accordingly, area of a 20x20 grid sized FPGA can be calculated by scaling.

Table 4.3. Table of sizes with new SRAM cell.

CLB	**CX10**	**CX11**	**CX12**	**CY01**	**CY11**	**CY21**	**SB00**
90009.25	24875.5	13668.5	13841.5	15520	13910.5	23578	28397
SB01	**SB02**	**SB10**	**SB11**	**SB12**	**SB20**	**SB21**	**SB22**
39274	24400	40313	54957	39032	32464	40469	28320
	logic	**routing**	**total**	**size**			
20x20	36003700	35012050	71015750	8.427084			

The decoder in the design and the inverter cells used for the inverted data input of the SRAM cell make the area cost disadvantageous compared to the scan chain. With the advantage of the DFT feature and area gain, scan chain protocol with the newly designed optimized DFF cell was the preferred protocol in this study.

4.2.2. Multiplexer Design

In this section, design of multiplexers, which are the most important elements that make up the FPGA is described. Traditionally, larger multiplexers are built using trees of smaller multiplexers, illustrated in Figure 4.12.

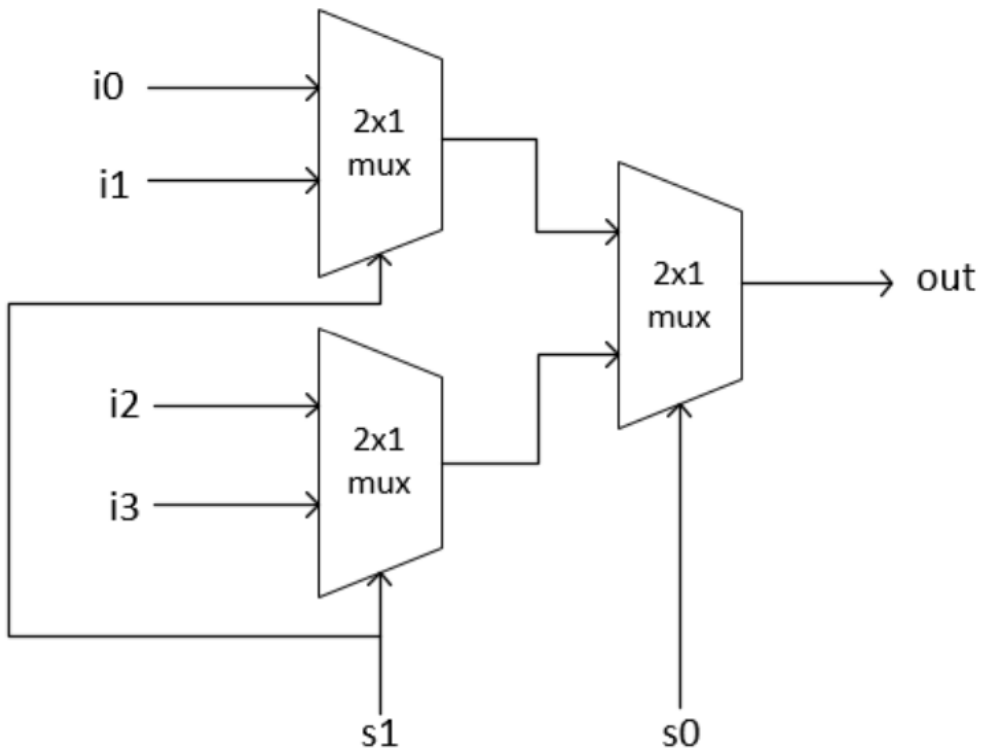

Figure 4.12. Multiplexer tree.

Multiplexer trees lead to large power and timing constraints that limit FPGA performance. FPGA fabrics use complementary pass gate logic (CPL) to replace multiplexer trees with transmission gate derived multiplexers, illustrated in Figure 4.13.

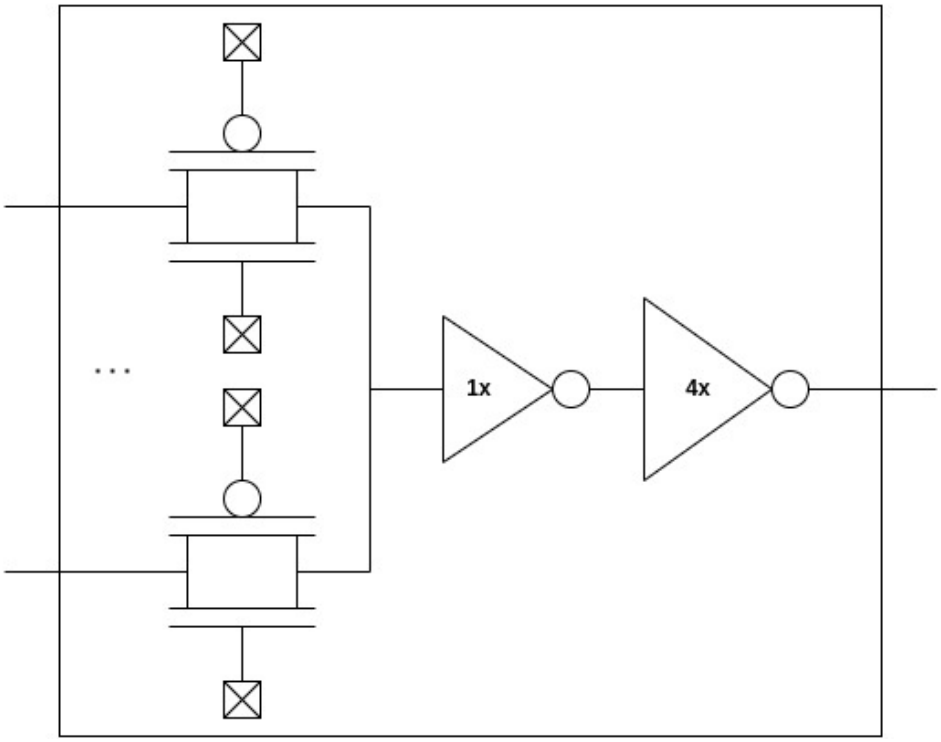

Figure 4.13. CPL multiplexer.

Single level multiplexers are controlled through configuration cells that enable high impedance connections throughout the multiplexer hierarchy, thereby removing the need for hierarchical designs of multiplexers. Therefore, the CPL multiplexers enable increased performance and reduced power consumption throughout FPGA fabrics. Standard cells required for CPL multiplexers are not commonly included in PDKs, thereby requiring the need for custom cell creation to enable FPGA multiplexer hierarchies.

In the CPL multiplexer structure, the inverted state of the enable signal is also required. Due to tansmission gate structure, enable_bar pin is needed for pMOSFET besides nMOSFET enable pin. It is possible to use local encoder structures to optimize the number of configuration flip-flops required for multiplexers. The area overhead of the CPL multiplexer can be eliminated by adding inverted output to the local encoder.

It is experimentally determined that CPL multiplexer primitives of different sizes from 2 to 8 are required for all FPGA sizes. Custom cell designs are required for all

these dimensions. The schematic and layout of the designed 4-input CPL multiplexer are shown in the Figures 4.14 and 4.15 below.

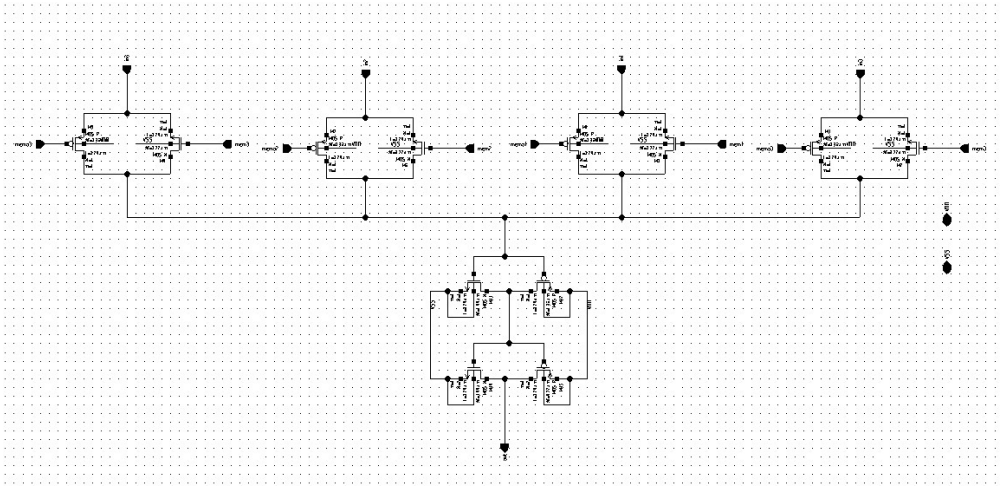

Figure 4.14. Schematic of 4-input CPL multiplexer.

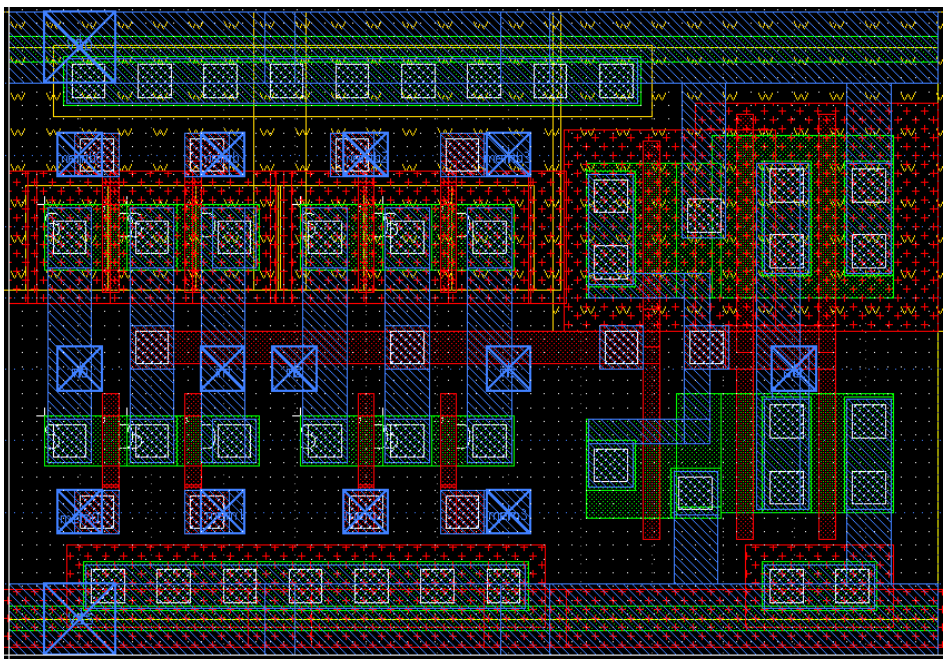

Figure 4.15. Layout of 4-input CPL multiplexer.

In order to observe the speed and area performance of CPL multiplexers, it would be appropriate to compare the synthesis results of local routing multiplexers in the CLB in k4n6 architecture. Local routing structures that maximize the CLB utilization

rate by crossing the LUT outputs and CLB inputs in the form of a crossbar are the largest sized multiplexers used in FPGA. Here, the performance comparison of CPL multiplexers and multiplexer trees will be more obvious. A multiplexer with 26 inputs is used for K4N6 and 14 CLB input numbers.

A hypothetical 1ns input to output combinational delay constraint is provided during synthesis. When the area reports in Figures 4.16 and 4.17 are examined, it seems that 2level_mux is more advantageous in terms of area, although it also contains a local_encoder. When the timing reports in Figures 4.18 and 4.19 are examined, it seems that the input to output delay was 430ps for 2level_mux, but mux_tree caught 1ns timing constraint hardly. This obvious difference in terms of area and speed shows that it is an inevitable design choice for multiplexer blocks, which are used extensively in every part of the FPGA.

```
****************************************
Design : mux_2level_size26
****************************************
BIR                                17.305599
local_encoder3to6_0               475.903999
local_encoder3to6_1               475.903999
mux_2level_basis_input2_mem2       77.879997
mux_2level_basis_input6_mem6_0    155.745602
mux_2level_basis_input6_mem6_1    138.440002
mux_2level_basis_input6_mem6_2    138.440002
mux_2level_basis_input6_mem6_3    138.440002
mux_2level_basis_input6_mem6_4    138.440002
----------------------------------------
Total 9 references                1756.499207
```

Figure 4.16. Area report of CPL multiplexer with 26 inputs.

```
****************************************
Design : mux_tree_size26
****************************************
Reference               Unit Area   Count    Total Area
--------------------------------------------------------
MUX2                    60.569599       1     60.569599
SEVIRX1                 25.958401       9    233.625607
VEVEYAd3_X1             43.264000       6    259.584000
VEVEYAd12d              51.916801       2    103.833603
VEVEYAd22               51.916801       9    467.251213
VEYAVEd1d2_X            95.180801       1     95.180801
VEYAVEd1d2_v0           51.916801       2    103.833603
VEYAVEd2dd1_X1          51.916801       3    155.750404
VEYAVEd3_X1             43.264000       2     86.528000
VEYAVEd3_X2             77.875198       1     77.875198
VEYAd2d1_X1             43.264000       1     43.264000
VEYAd2d1_v1             43.264000       4    173.056000
--------------------------------------------------------
                                       1860.352028
```

Figure 4.17. Area report of multiplexer tree with 26 inputs.

```
*************************************
Design : mux_2level_size26
*************************************
  Point                                          Fanout     Trans      Incr       Path
  ------------------------------------------------------------------------------------
  input external delay                                                 0.00       0.00 f
  in[0] (in)                                                  0.00     0.00       0.00 f
  in[0] (net)                                        1                 0.00       0.00 f
  mux_l1_in_0_/in[0] (mux_2level_basis_input6_mem6_4)                  0.00       0.00 f
  mux_l1_in_0_/in[0] (net)                                             0.00       0.00 f
  mux_l1_in_0_/mux_input6_mem6_i/out (mux_input6_mem6)        0.09     0.23 *     0.23 f
  mux_l1_in_0_/out[0] (net)                          1                 0.00       0.23 f
  mux_l1_in_0_/out[0] (mux_2level_basis_input6_mem6_4)                 0.00       0.23 f
  mux_2level_basis_input6_mem6_0_out (net)                             0.00       0.23 f
  mux_l2_in_0_/in[0] (mux_2level_basis_input6_mem6_0)                  0.00       0.23 f
  mux_l2_in_0_/in[0] (net)                                             0.00       0.23 f
  mux_l2_in_0_/mux_input6_mem6_i/out (mux_input6_mem6)        0.06     0.21 *     0.43 f
  mux_l2_in_0_/out[0] (net)                          1                 0.00       0.43 f
  mux_l2_in_0_/out[0] (mux_2level_basis_input6_mem6_0)                 0.00       0.43 f
  out[0] (net)                                                         0.00       0.43 f
  out[0] (out)                                                0.06     0.00 *     0.43 f
  data arrival time                                                                0.43

  max_delay                                                            1.00       1.00
  output external delay                                                0.00       1.00
  data required time                                                               1.00
  ------------------------------------------------------------------------------------
  data required time                                                               1.00
  data arrival time                                                               -0.43
  ------------------------------------------------------------------------------------
  slack (MET)                                                                      0.57
```

Figure 4.18. Timing report of CPL multiplexer with 26 inputs.

```
****************************************
Design : mux_tree_size26
****************************************
  Point                          Fanout     Trans      Incr       Path
  ---------------------------------------------------------------------
  input external delay                                0.00       0.00 f
  in[18] (in)                              0.00       0.00       0.00 f
  in[18] (net)                    1                   0.00       0.00 f
  U46/Y (VEYAd2d1_X1)                      0.28       0.14 *     0.14 r
  n37 (net)                       1                   0.00       0.14 r
  U47/Y (VEVEYAd12d)                       0.30       0.11 *     0.25 f
  n42 (net)                       1                   0.00       0.25 f
  U51/Y (VEYAVEd2dd1_X1)                   0.24       0.28 *     0.53 f
  n44 (net)                       1                   0.00       0.53 f
  U52/Y (MUX2)                             0.12       0.28 *     0.81 f
  n68 (net)                       1                   0.00       0.81 f
  U71/Y (VEYAVEd2dd1_X1)                   0.12       0.18 *     1.00 f
  out[0] (net)                    1                   0.00       1.00 f
  out[0] (out)                             0.12       0.00 *     1.00 f
  data arrival time                                                1.00

  max_delay                                           1.00       1.00
  output external delay                               0.00       1.00
  data required time                                             1.00
  ---------------------------------------------------------------------
  data required time                                             1.00
  data arrival time                                             -1.00
  ---------------------------------------------------------------------
  slack (MET)                                                    0.00
```

Figure 4.19. Timing report of multiplexer tree with 26 inputs.

As a result of these studies, the design preferences for the FPGA became clear. Design parameters of FPGA can be seen in Table 4.4.

Table 4.4. Table of FPGA design parameters.

LUT Structure(N)	Fracturable 4-LUT
CLB Size(K)	6
CLB Input Number(I)	14
Multiplexers	CPL 2 Level Multiplexer
Configuration Protocol	Scan Chain D-Flip Flop

4.3. Final Layout Design

Place&Route design of FPGA's tileable submodules was done with IC Compiler tool. The designed new DFF and CPL multiplexers were added to the standard cell library and used during P&R. During the design, CLB module aspect ratio and utilization constraints 1 and 85%, respectively. For other modules, T, L and custom shaped floorplans were designed as shown in Figure 3.3, according to the coordinates of the modules placed. 85% utilization target was achieved for these modules. The layout of the 2x2 grid sized FPGA is shown in Figure 4.20.

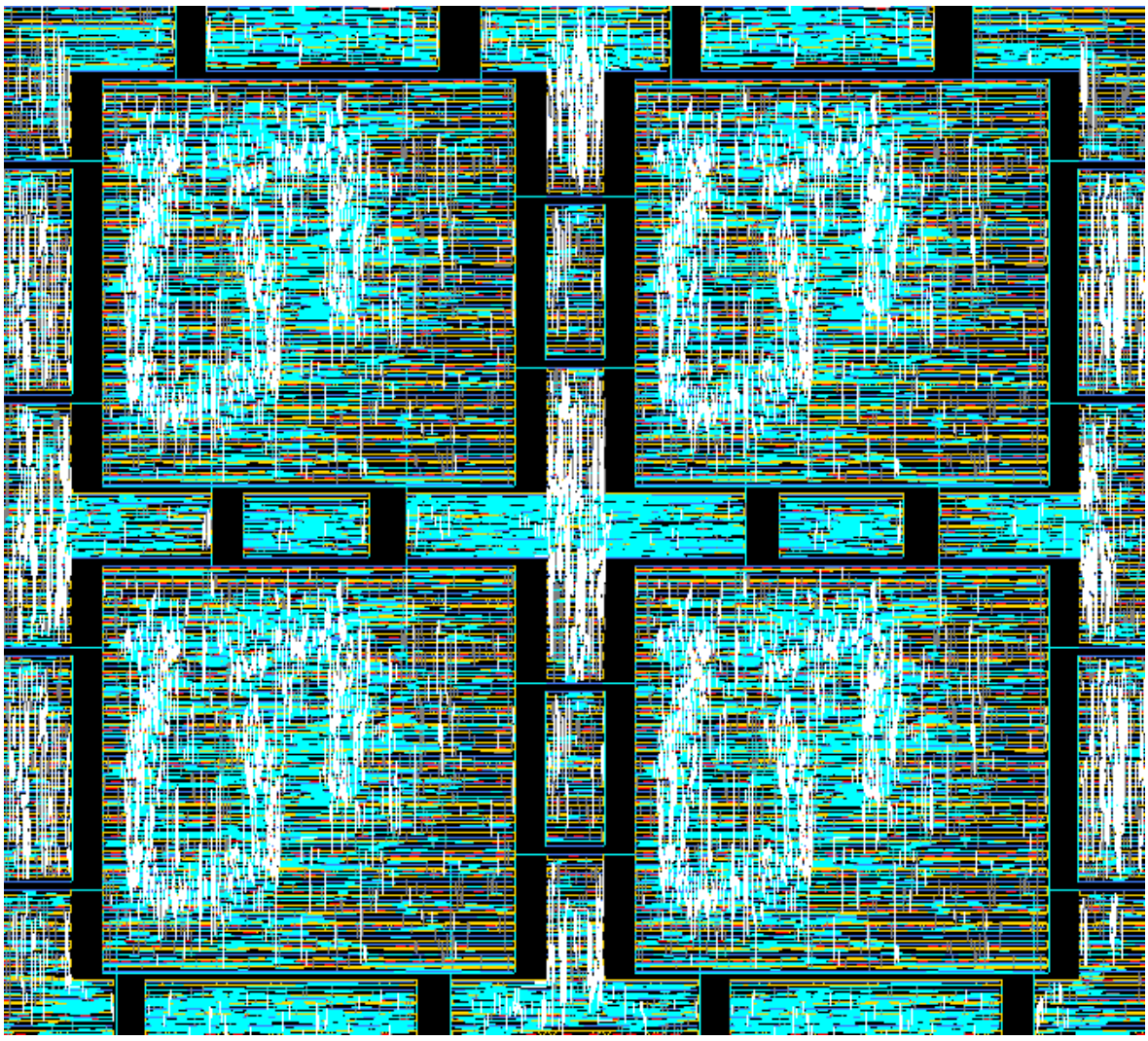

Figure 4.20. Layout of 2x2 FPGA.

Figure 4.21 is the final DRC-LVS clean layout of the 20x20 grid sized FPGA, which is the ultimate goal of this study. For larger grid sized FPGAs, the design was automated by automatic macro placement with the help of a TCL script that is automatically generated by a Python script in macro form of modules, whose design was finalized in FPGA with 2x2 layout. Some distance parameters are determined, and a regular placement methodology is developed. Figure 4.22 is a part of the script where the macro placement commands are automatized.

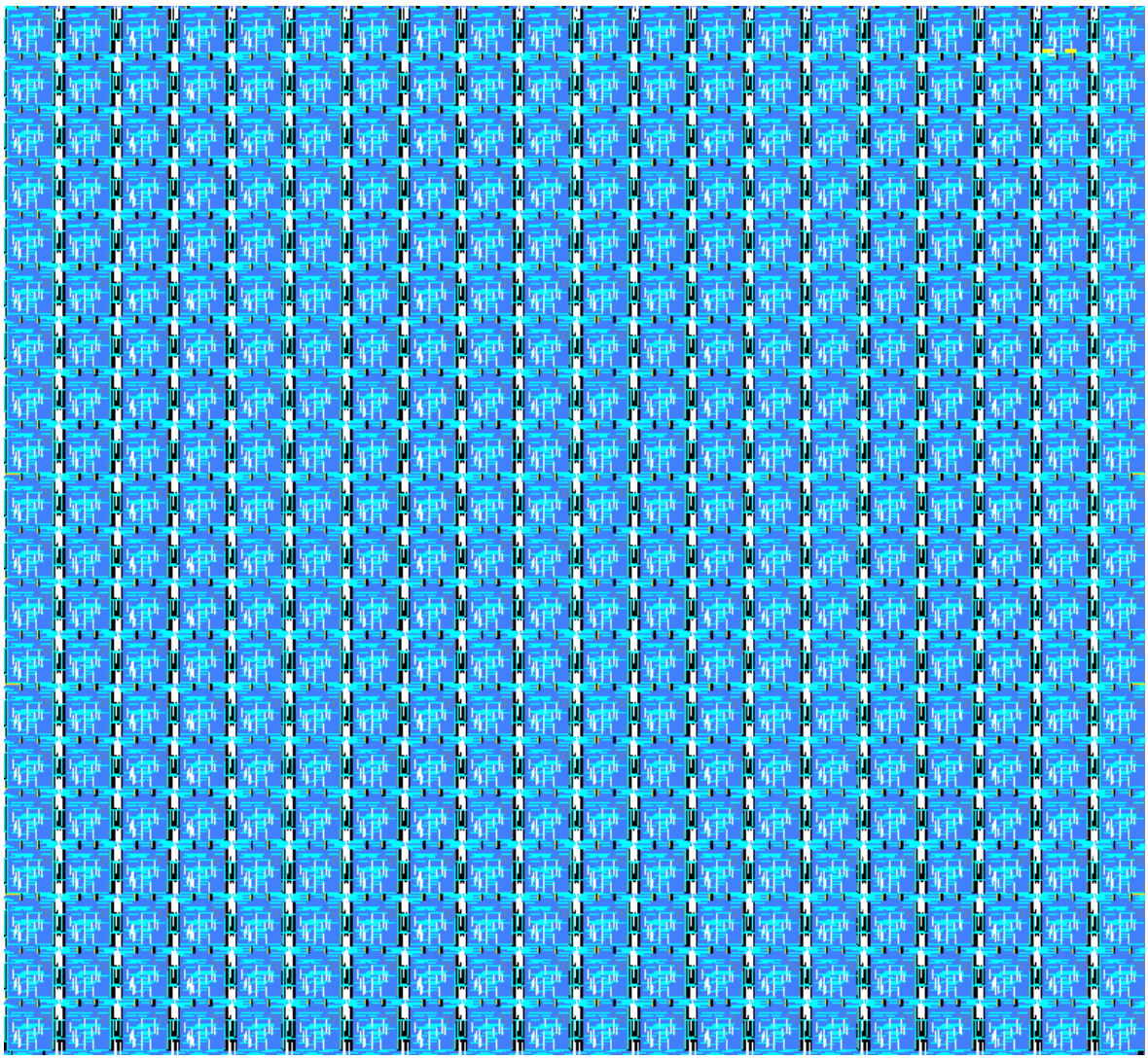

Figure 4.21. Layout of 20x20 FPGA.

```
set_attribute -quiet [get_cells {"cbx_1__0_"} -all] origin {space+bottom_sb00+offset bottom}
set_attribute -quiet [get_cells {"cbx_1__1_"} -all] origin {space bottom+left_grid_clb+left_cx10+2*row}
set_attribute -quiet [get_cells {"cbx_1__2_"} -all] origin {space bottom+2*left_grid_clb+left_cx10+left_cx11+4*row}
set_attribute -quiet [get_cells {"cbx_2__0_"} -all] origin {space+bottom_sb00+bottom_cx10+bottom_sb10+3*offset bottom}
set_attribute -quiet [get_cells {"cbx_2__1_"} -all] origin {0 bottom+left_grid_clb+left_cx10+2*row}
set_attribute -quiet [get_cells {"cbx_2__2_"} -all] origin {0 bottom+2*left_grid_clb+left_cx10+left_cx11+4*row}

set_attribute -quiet [get_cells {"cby_0__1_"} -all] origin {space bottom+left_sb00+row}
set_attribute -quiet [get_cells {"cby_0__2_"} -all] origin {space bottom+left_sb00+left_cy01+left_sb01+3*row}
set_attribute -quiet [get_cells {"cby_1__1_"} -all] origin {space+bottom_cy01+bottom_grid_clb+2*offset bottom+left_sb10+row}
set_attribute -quiet [get_cells {"cby_1__2_"} -all] origin {space+bottom_cy01+bottom_grid_clb+2*offset bottom+left_sb10+left_cy11+left_sb11+3*row}
set_attribute -quiet [get_cells {"cby_2__1_"} -all] origin {space+bottom_cy01+2*bottom_grid_clb+bottom_cy11+4*offset bottom+left_sb20+row}
set_attribute -quiet [get_cells {"cby_2__2_"} -all] origin {space+bottom_cy01+2*bottom_grid_clb+bottom_cy11+4*offset bottom+left_sb20+left_cy21+left_sb21+3*row}

set_attribute -quiet [get_cells {"grid_clb_1__1_"} -all] origin {space+bottom_cy01+offset bottom+left_cx10+row}
set_attribute -quiet [get_cells {"grid_clb_1__2_"} -all] origin {space+bottom_cy01+offset bottom+left_cx10+left_grid_clb+left_cx11+3*row}
set_attribute -quiet [get_cells {"grid_clb_2__1_"} -all] origin {space+bottom_cy01+bottom_grid_clb+bottom_cy11+3*offset bottom+left_cx10+row}
set_attribute -quiet [get_cells {"grid_clb_2__2_"} -all] origin {space+bottom_cy01+bottom_grid_clb+bottom_cy11+3*offset bottom+left_cx10+left_grid_clb+left_cx11+3*row}

set_attribute -quiet [get_cells {"sb_0__0_"} -all] origin {space bottom}
set_attribute -quiet [get_cells {"sb_0__1_"} -all] origin {space bottom+left_sb00+left_cy01+2*row}
set_attribute -quiet [get_cells {"sb_0__2_"} -all] origin {space bottom+left_sb00+2*left_cy01+left_sb01+4*row}
set_attribute -quiet [get_cells {"sb_1__0_"} -all] origin {space+bottom_cy01+bottom_grid_clb-top_left_sb10+2*offset bottom}
set_attribute -quiet [get_cells {"sb_1__1_"} -all] origin {space+bottom_cy01+bottom_grid_clb-bottom_left_sb11+2*offset bottom+left_sb10+left_cy11+2*row}
set_attribute -quiet [get_cells {"sb_1__2_"} -all] origin {space+bottom_cy01+bottom_grid_clb-bottom_left_sb12+2*offset bottom+left_sb10+2*left_cy11+left_sb11+4*row}
set_attribute -quiet [get_cells {"sb_2__0_"} -all] origin {space+bottom_cy01+2*bottom_grid_clb+bottom_cy11-top_left_sb20+4*offset bottom}
set_attribute -quiet [get_cells {"sb_2__1_"} -all] origin {space+bottom_cy01+2*bottom_grid_clb+bottom_cy11-bottom_left_sb21+4*offset bottom+left_sb20+row}
set_attribute -quiet [get_cells {"sb_2__2_"} -all] origin {space+bottom_cy01+2*bottom_grid_clb+bottom_cy11-bottom_left_sb22+4*offset bottom+left_sb20+2*left_cy21+left_sb21+4*row}
```

Figure 4.22. Macro placement TCL script.

The sizes of all blocks from the layout implementation results of implemented FPGA are listed in Table 4.5. The area occupied by 20x20 layout is 9.3x9.3 mm^2. The maximum die size supported by the technology is 10x10 mm^2. Larger die size is not preferred as it will negatively affect the fabrication yield. In this thesis, preferring performance-oriented structures while staying within the die-size limits set by the technology was possible.

Table 4.5. Table of sizes of 20x20 FPGA.

CLB	**CX10**	**CX11**	**CX12**	**CY01**	**CY11**	**CY21**	**SB00**
105290.8	31598.55	17428.25	17393.18	19223.98	17393.75	29911.5	38985
SB01	**SB02**	**SB10**	**SB11**	**SB12**	**SB20**	**SB21**	**SB22**
53882.1	34089.45	55145.95	70998.7	53852.2	44199.1	55156.3	38945.9
	total	**size**					
20x20	87240684	9.340272					

4.4. Verification Results

In order to verify the design, it is sufficient to run any Verilog design on the FPGA as the entire FPGA is configured during the operation. The operating frequency of a design implemented on the FPGA depends upon various factors:

- Architecture, whether or not the design will be implemented using LUTs, soft logic, or hard logic
- Synthesis script used in FPGA CAD flow (if it can derive design logic and implement it on any of the above-mentioned specialized structures)
- Structure's timing delays for design's critical path

During architecture exploration, all synthesized netlists were consisting of multiplexer tree based structures. Operating period of UART design was around $40ns$ (see Table 3.5). This time UART operating period is $29ns$ and combinational speeds of the design are at least $5ns$ faster. UART RX waveforms can be seen in Figure 4.23 and Figure 4.24.

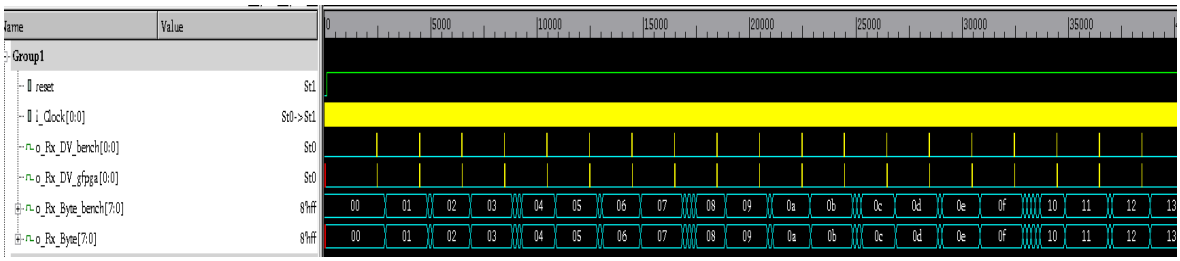

Figure 4.23. UART RX post-layout simulation results - 1.

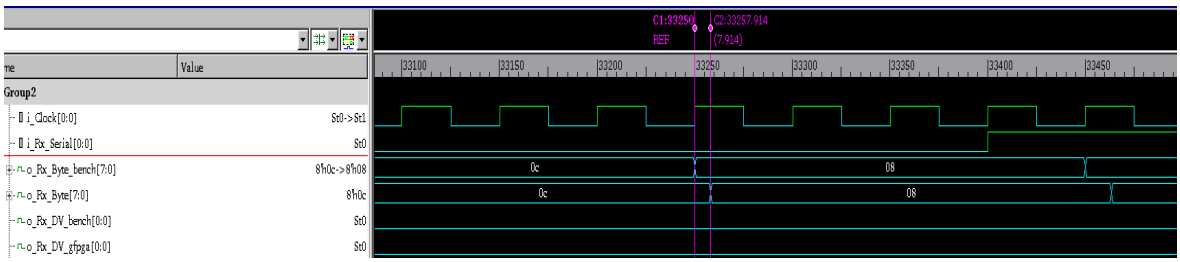

Figure 4.24. UART RX post-layout simulation results - 2.

When waveforms are observed signal names ending with _bench and _gfpga can be seen. The signals with _bench are behavioral waveforms of the design. The signals with _gfpga are FPGA's related I/Os that correspond to the same signal with _bench. The behavioral signal rises/falls at the edge of the clock signal, but FPGA signal comes with a delay. From these waveforms, it can be understood that the design is successfully mapped onto FPGA because it acts the same as its behavioral counterpart, and sdf delays can be viewed with the help of a cursor. UART TX waveforms can be seen in Figure 4.25 and Figure 4.26.

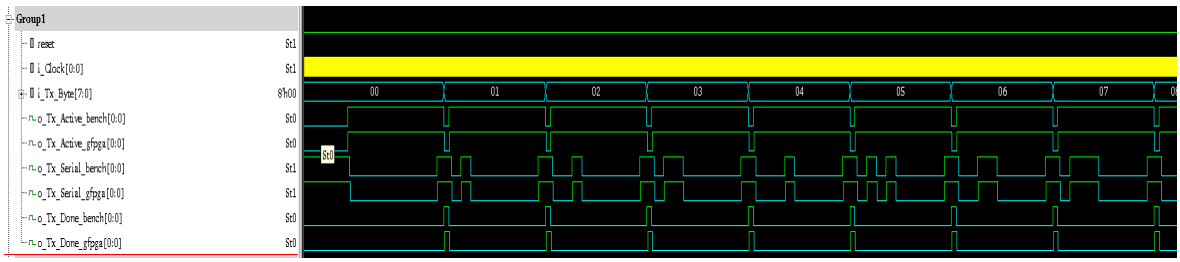

Figure 4.25. UART TX post-layout simulation results - 1.

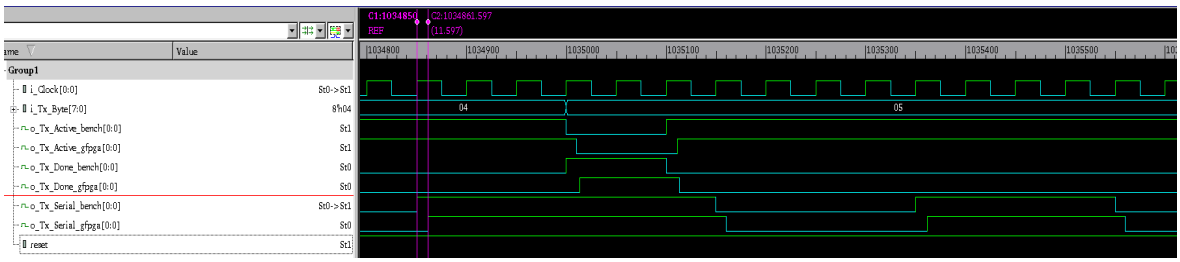

Figure 4.26. UART TX post-layout simulation results - 2.

5. CONCLUSION

In this thesis, we designed a low-cost FPGA using 250nm CMOS technology. With this thesis study, and thanks to new Open Source FPGA researches such as VTR and OpenFPGA, there may no longer be a need for multi-person teams that require high effort to design a small eFPGA for SoC.

To summarize the thesis progress, we focused on choosing the best area-speed FPGA architecture by choosing between FPGA architecture options, CLB sizes, and channel width numbers in the architectural design chapter. We explored the ideal design parameters, including Look-up Table (LUT) and Configurable Logic Block (CLB) sizes, channel width number. For this, we consistently compared architectures with iteratively synthesized MCNC benchmark circuits with The Verilog-to-Routing (VTR) tool and physical synthesis of small FPGAs with the Design Compiler.

In Chapter 4, the focus was on the physical design of the FPGA, whose design parameters were clarified. We used Synopsys IC Compiler 2 as the physical implementation tool. We made several design efforts to determine the best configuration protocol and multiplexer type, the two primary structures that form the FPGA. An area-optimized D-Flip flop for the configuration protocol has been added to the standard cell library. With the CPL architecture for multiplexer structures, we improved the operating frequency of the FPGA significantly. As a result of the clarification of these design improvements, which is the ultimate goal of the study, we successfully implemented the 20x20 FPGA within the die-size limits of the technology.

This research demonstrates that despite the challenges posed by the nature of FPGAs and the limitations of an older technology node, significant progress can be made using design techniques which are described throughout the thesis.

However, some design issues still exist that OpenFPGA still needs to solve fully by enhancing the capability of its automatically generated netlists. The first design issue is that generated netlists are functionally correct but need to be more clock network aware. Some modifications that will be done on netlists before synthesis would give better clock skew results. Another issue is that all SRAM cells are spread in the submodules in the frame protocol. Developing a way to gather SRAM cells on the tile level and make the SRAM cells replaceable with SRAM memory macros will give better area results. Although OpenFPGA toolset is open source, implementation tools which is used during layout phase are commercial. In a future work, designing an open source FPGA with entirely open source tool from RTL to GDS should be cared.

Moreover, open source FPGA design using Turkey's domestic production technology will eliminate Turkey's foreign dependency in this field to some extent. This study can play an important role in eliminating the periodic problems experienced in the supply of such widely used technologies, such as the Covid pandemic.

In summary, this thesis has provided insights into optimizing area and speed in low-cost FPGA designs. By leveraging Open Source FPGA frameworks and a 250nm CMOS technology, we have explored architectural and physical design stages to achieve efficient FPGA implementations. As FPGA technology evolves, further advancements and research in this field will undoubtedly lead to even more efficient and cost-effective solutions for digital designs. By continually exploring architectures, we can unlock the full potential of FPGAs and benefit them for a wide range of applications.

REFERENCES

1. Murray, K. E., 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 and V. Betz, "VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling", *Association for Computer Machinery Transactions on Reconfigurable Technology and Systems*, Vol. 13, pp. 1–55, 2020.

2. Tang, X., E. Giacomin, B. Chauviere, A. Alacchi and P.-E. Gaillardon, "OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs", *IEEE Micro*, Vol. 40, No. 4, pp. 41–48, 2020.

3. Clos, C., "A Study of Non-blocking Switching Networks", *The Bell System Technical Journal*, Vol. 32, No. 2, pp. 406–424, 1953.

4. Lemieux, G., E. Lee, M. Tom and A. Yu, "Directional and Single-driver Wires in FPGA Interconnect", *Proceedings. IEEE International Conference on Field-Programmable Technology (IEEE Cat. No.04EX921)*, Brisbane, Australia, 2004.

5. Ahmed, E. and J. Rose, "The Effect of LUT and Cluster Size on Deep-submicron FPGA Performance and Density", *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 12, No. 3, pp. 288–298, 2004.

6. Tang, X. and L. Wang, "The Effect of LUT Size on Nanometer FPGA Architecture", *IEEE 11th International Conference on Solid-State and Integrated Circuit Technology*, Xi'an, China, 2012.

7. Yang, S., "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0", `api.semanticscholar.org/CorpusID:61228243`, accessed on May 21, 2023.

8. Farooq, U., Z. Marrakchi and H. Mehrez, *Tree-Based Heterogeneous FPGA Architectures*, Springer, New York, USA, 2012.

9. Gore, G., X. Tang and P.-E. Gaillardon, "A Scalable and Robust Hierarchical Floorplanning to Enable 24-hour Prototyping for 100k-LUT FPGAs", *Association for Computing Machinery International Symposium on Physical Design*, pp. 135–142, 2021.

10. Gore, G., "SOFA eFPGAs", 2020, `https://skywater-openfpga.readthedocs.io`, accessed on May 21, 2023.

11. Tuan, T. and B. Lai, "Leakage Power Analysis of a 90nm FPGA", *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Jose, USA, 2003.

12. Tang, X., "Circuit Design, Architecture and CAD for RRAM-based FPGAs", Lausanne, 2017, `https://api.semanticscholar.org/CorpusID:116121394`, accessed on May 21, 2023.