ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

SYSTEM-ON-CHIP DESIGN WITH OPEN-SOURCE FPGA IP

M.Sc. THESIS

Yunus Emre ERYILMAZ

Department of Electronics and Communication Engineering

Electronics Engineering Programme

MARCH 2025

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

SYSTEM-ON-CHIP DESIGN WITH OPEN-SOURCE FPGA IP

M.Sc. THESIS

Yunus Emre ERYILMAZ
(504211243)

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Müştak Erhan YALÇIN
Co-advisor: Dr. Hasan Erdem YANTIR

MARCH 2025

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

AÇIK KAYNAK GÖMÜLÜ FPGA IP'Sİ İLE KIRMIK ÜSTÜ SİSTEM TASARIMI

YÜKSEK LİSANS TEZİ

Yunus Emre ERYILMAZ
(504211243)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Müştak Erhan YALÇIN
Eş Danışman: Dr. Hasan Erdem YANTIR

MART 2025

Yunus Emre ERYILMAZ, a M.Sc. student of ITU Graduate School student ID 504211243 successfully defended the thesis entitled ““SYSTEM-ON-CHIP DESIGN WITH OPEN-SOURCE FPGA IP”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Müştak Erhan YALÇIN** Istanbul Technical University	.....
Co-advisor :	**Dr. Hasan Erdem YANTIR** TÜBİTAK BİLGEM	.....
Jury Members :	**Prof. Dr. Sıddıka Berna ÖRS YALÇIN** Istanbul Technical University	.....
	Prof. Dr. Ece Olcay GÜNEŞ Istanbul Technical University	.....
	Dr. Emre GÖNCÜ Nexustech Electronics, Comm. and Info. Tech.	.....

Date of Submission : **10 January 2025**
Date of Defense : **7 March 2025**

To my family,

FOREWORD

First of all, I would like to thank my advisors, Prof. Dr. Müştak Erhan YALÇIN and Dr. Hasan Erdem YANTIR for their guidance and support in every part of this study.

I also would like to thank my family for their endless support throughout my career and education.

I would like to thank IC Design and Education Laboratory (TÜTEL) of TÜBİTAK BİLGEM for their help with experiences on OpenFPGA, and EDA and PDK support. I also want to thank Mustafa ARSLAN and Mehmet Said EROĞLU for their helps on FPGA research in TÜBİTAK.

March 2025

Yunus Emre ERYILMAZ
(Electronics and Communication Engineer)

TABLE OF CONTENTS

ABBREVIATIONS

AMBA	**:** Advanced Microcontroller Bus Architecture
ASIC	**:** Application Specific Integrated Circuit
AXI	**:** Advanced eXtensible Interface
AI	**:** Artificial Intelligence
EDA	**:** Electronic Design Automation
FPGA	**:** Field Programmable Gate Array
FIR	**:** Finite Impulse Response
IC	**:** Integrated Circuit
IP	**:** Intellectual Property
LUT	**:** Look-up Table
IoT	**:** Internet of Things
RTL	**:** Register Transfer Level
SoC	**:** System-on-chip
UART	**:** Universal Asynchronous Receiver-Transmitter

LIST OF TABLES

Page

LIST OF FIGURES

Page

SYSTEM-ON-CHIP DESIGN WITH OPEN-SOURCE FPGA IP

SUMMARY

In recent years, the demand for computing power has increased due to the increasing number of Internet of Things (IoT) devices and artificial intelligence applications. Field programmable gate arrays (FPGA) are frequently used to meet this demand because of their simultaneous computation, reconfigurability, and high bandwidth. However, integrating FPGAs into the system is challenging. Since they use multiple voltage levels and consume lots of power, producing a suitable printed circuit board (PCB) takes time to develop and increases design costs. At the same time, FPGA packages are large, and the price per piece is higher compared to many integrated circuits, so the procurement costs of products containing FPGAs are also high. Embedded FPGAs (eFPGA) aim to solve these problems by integrating FPGA fabrics into the system-on-chips (SoC). EFPGA vendors can produce FPGA fabrics with fewer look-up tables (LUT), i.e. fabrics with less space and power consumption, to satisfy customer requirements. There are two different design methods for embedded FPGAs: hard and soft. Hard eFPGAs are designed at the transistor level, similar to discrete FPGAs, and are specific to a semiconductor manufacturing technology. Soft eFPGAs are generated as RTL code, and since they are independent of the manufacturing technology, the architectures can be easily fine-tuned and manufactured using different technologies. In this study, a system-on-chip system with soft eFPGA intellectual property (IP) is designed. There are similar studies on this topic, but the aim is to show that it is possible to design an SoC with open-source tools and designs. In addition to the embedded FPGA IP; the processor, the memory that stores the program data for the processor and bitstream files for the FPGA, and two UART elements, one for the processor to use and one for loading the data of the memory element. The Advanced eXtensible Interface 4 (AXI4) protocol of Advanced Microcontroller Bus Architecture (AMBA) standard provides the on-chip communication. The system is mostly prepared with open-source design tools or taken from open-source projects. The processor is CVA6, previously developed in the PULP Platform group of ETH Zürich and now maintained by Open HW Group. It is a 64-bit processor, has open-source RISC-V architecture and supports I, M, C and A extensions. It is a parametric core, the optimum performance can be obtained by changing the parameters which define the core. The embedded FPGA IP is generated with an open-source FPGA fabric generator called OpenFPGA. OpenFPGA can produce RTL codes for FPGA in Verilog format, verification environment, Synopsys timing constraint commands and bitstream files suitable for the desired architecture with Yosys open source RTL synthesizer and Versatile Place-and-Route (VPR), FPGA placement and routing program. The FPGA prepared in this study includes 1960 six-input LUTs, 1960 flip-flops, 50 input-output

cells and a register interface. The logic blocks in the FPGA consist of ten LUTs, ten flip-flops and local routing multiplexers. This method gives the lowest value in terms of area-delay multiplication. Local routing multiplexers are reduced to 50% and reduce the delays in the logic block. Thus, the critical paths and the area covered by multiplexers are reduced without damaging the logic block functionality. The switching block is the Wilton style, which is the best in terms of routability and area, and its flexibility coefficient is 3. Multiplexers are selected for the switches in the FPGA because they cover a smaller area than tri-state buffers and can be optimized in digital implementation tools. Due to the small size of the routing architecture, L4 segments were used; longer ones were not preferred. Input-output (IO) blocks are used as standard input-output blocks of the preferred production technology, and vertical and horizontal IO blocks are made of separate cells to be compatible with the grid in the technology. The register interface provides the AXI interface to which the processor and other blocks are connected to communicate with the design inside the FPGA. The interface consists of two control and status registers and six 64-bit data registers so that the status of the eFPGA can be learnt and a total of 384-bit data can be transferred to the FPGA simultaneously. For programming the embedded FPGA IP, the memory bank was selected from five programming protocols in OpenFPGA. It takes up less space than other protocols because it allows the latch structure for programmable memory. Inside the system, up to three bitstream files can be stored and read from them to reprogram during runtime. A configuration circuit is designed to program the IP according to the preferred protocol in this study. Each configurable element is controlled through bit line (BL) and word line (WL) signals. The data to be written to the elements are loaded into BL, and the address information is loaded into the WL shift registers. When the loading to the shift registers is finished, an enable signal is sent to WL to load the data to selected latches. Each element in FPGA is programmed by following this sequence in 38849 clock cycles.

The system was tested with four different applications in behavioural simulation on Cadence Xcelium. The eFPGA-augmented and software versions, which run solely on the processor, are compared in terms of performance. When supported by embedded FPGA, a performance increase of 3.14 times in Advanced Encryption Standard (AES) encryption, 8 times in CRC-32, 2.8 times in Finite Impulse Response (FIR) filter and 1.36 times in 2D convolution operation was observed. Finally, the physical design of the system was made in Synopsys Design Compiler (DC) and IC Compiler II (ICC2). The physical design is divided into three sections since the flows take long times: logic block, FPGA and the system-on-chip. The logic block is synthesized in 100 MHz clock frequency and its layout is made on a square with a side length of 60 *mu*m. Optimizations which can add buffers or change the net connections are turned off in placement, clock tree synthesis, and routing. Also, timings in the latch outputs which do not show any switching activity are disabled to increase the performance of ICC2. The power meshes are created in the top two metal layers, which have the highest conductivity and the standard cell rails are created in the metal layers nearest to the standard cells. After working the logic block layout and characterization are finished, the FPGA is synthesized with the same clock frequency with logic blocks. Aside from the settings used before, switch block outputs are turned off to break combinational loops. Macro placement for the FPGA is done on a square with a side length of 1.5

mm, according to the placement plan taken from the OpenFPGA flow. After the FPGA is hardened, the system-on-chip flow is run. Since the potential combinational loops are in the FPGA and the optimizations are important for the processor performance, the settings used before were not preferred in the SoC. In the floorplanning stage, the FPGA is placed near to the center and SRAM blocks of the bitstreams and the program memory are placed at the edge of the design to complete flow without issues. Thus, the SoC is place-and-routed in 4.84 mm^2 area for 100 MHz clock frequency.

As a result, a system-on-chip system with embedded FPGAs can be built for various applications with the open-source tools and designs we use. However, features such as custom switch block models, loop-free FPGA fabric, and clock tree regions are not supported in OpenFPGA. So, designs that compete with commercial FPGAs cannot be obtained. In addition, the physical design of the system was made without using OpenFPGA tools, except for timing constraints. If the community and developers contribute more to the project, the open-source tools used can be used in wider areas.

AÇIK KAYNAK GÖMÜLÜ FPGA IP'Sİ İLE KIRMIK ÜSTÜ SİSTEM TASARIMI

ÖZET

Son yıllarda, gerek nesnelerin interneti gerek yapay zeka uygulamaları sebebiyle hesaplama gücüne talep artmıştır. Bu talebi karşılamak için alanda programlanabilir kapı dizilerine (Field Programmable Gate Arrays, FPGA) eşzamanlı hesaplama, tekrar tekrar programlanabilme ve yüksek bant genişliğinde veri üretebildikleri için sıklıkla başvurulur. Ancak, FPGA'leri sisteme entegre etmek zordur. Birden fazla gerilim seviyesi kullandığı için ve çok güç tükettiği için uygun baskı devre kartının üretilmesi geliştirme süresi zaman almakta ve tasarım maliyetlerini arttırmaktadır. Aynı zamanda FPGA paketleri büyük ve parça başı fiyatı birçok tümdevreye göre pahalıdır, bu yüzden FPGA içeren ürünlerin tedarik maliyetleri de yüksektir. Bu sorunları, kırmık üstü sisteme bağlanan gömülü FPGA'ler çözmeyi hedeflemekte. Gömülü FPGA üreticileri, müşterilerin isterlerine göre daha az sayıda başvuru tablosu (lookup table - LUT) içeren, bir diğer deyişle daha az alan kaplayan ve güç tüketen FPGA'ler üretebilir. Gömülü FPGA'lerde sabit ve esnek diye iki farklı tasarım yöntemi vardır. Sabit gömülü FPGA'ler, ayrı FPGA'lere benzer şekilde transistör seviyesi tasarlanır, bir yarı iletken üretim teknolojisine mahsustur. Esnek ise kayıtçı transfer seviyesi (register transfer level - RTL) kodu halinde bulunur, üretim teknolojisinden bağımsız oldukları için mimarilere kolaylıkla ince ayarlamalar yapılabilir, farklı teknolojilerde üretilebilir. Bu çalışmada, esnek gömülü FPGA IP'si içeren bir kırmık üstü sistem tasarlandı. Literatürde benzer çalışmalar bulunmasına karşın, bu çalışmada açık kaynak araçlar ve tasarımlar kullanarak böyle bir kırmık üstü sistemin nasıl tasarlanabileceğini göstermek hedeflenmektedir. Gömülü FPGA IP'sinin yanı sıra, işlemci, FPGA için bit akış dosyası ve işlemci için program verilerini kaydeden hafıza elemanı ve biri işlemcinin kullanması, biri de hafıza elemanının verilerini yüklemesi için iki Universal Asynchronous Receiver-Transmitter (UART) elemanını içerir. Kırmık içi haberleşme Advanced Microcontroller Bus Architecture (AMBA) standardının Advanced eXtensible Interface 4 (AXI4) protokolü ile sağlanmaktadır. Sistem çoğunlukla açık kaynak tasarım araçlarıyla veya açık kaynak projelerden alınarak hazırlandı. İşlemci, önceden ETH Zürich'in PULP Platform grubunda geliştirilen, şimdi Open HW Group sürdürülen CVA6'dır. 64-bit ve açık kaynaklı RISC-V mimarisinde olup I, M, C ve A uzantılarını destekler. Parametrik bir işlemcidir, oluşturan parametreler değiştirilerek en iyi performansı sağlayan kombinasyon elde edilebilir. Gömülü FPGA IP'si, OpenFPGA açık kaynaklı FPGA üreticisiyle üretildi. OpenFPGA, Yosys açık kaynak RTL sentezleyicisini ve VPR, FPGA yerleştirme ve rotalama programını içeren istenilen mimariye uygun Verilog formatında FPGA için RTL kodlarını, doğrulama ortamını, Synopsys zamanlama kısıtlama komutlarını

ve FPGA'ye uygun bit akış dosyalarını üretebilmektedir. Bu çalışmada hazırlanan FPGA 1960 adet altı girişli LUT, 1960 adet flip-flop, 50 giriş-çıkış hücresi ve kayıtçı arayüzü içermektedir. FPGA içindeki lojik bloklar onar adet LUT, onar adet flip-flop ve yerel rotalama çoklayıcılarından oluşmuştur. Bu yöntem alan-gecikme çarpımı bakımından en düşük değeri vermektedir. Yerel rotalama çoklayıcılarının arasındaki bağlantı %50 oranında azaltılmıştır. Böylece lojik bloğun işlevselliğine zarar vermeden kritik yol sayısı ve çoklayıcıların kapladığı alan azaldı, performans arttı. Anahtarlama bloğu rotalanabilirlik ve alan açısından en iyisi olan Wilton stili'dir ve esneklik katsayısı üçtür. FPGA içindeki anahtarlarda dijital tasarım araçlarında üç durumlu buffera göre daha az alan kapladığı ve dijital tasarım araçları içinde daha iyi optimize edilebildiği için çoklayıcı kullanılmıştır. Rotalama mimarisinde FPGA boyutunun küçük olması sebebiyle L4 segment kullanılmış, daha uzunları tercih edilmemiştir. Giriş-çıkış (GÇ) blokları tercih edilen üretim teknolojisinin standart giriş-çıkış blokları kullanıldı ve teknolojideki ızgara sistemine uygun olması için dikey ve yatay GÇ blokları olarak iki türe ayrıldı. Kayıt arayüzü, işlemci ve diğer blokların bağlandığı AXI arayüzünü FPGA içindeki tasarımla haberleşmesini sağlar. Gömülü FPGA'yi kontrol etmeyi sağlayan arayüz, iki adet kontrol ve durum kayıtçısına ve altı adet 64-bitlik veri kayıtçısını oluşturur. Kayıtçılar aracılığıyla IP'nin durumu öğrenilebilir ve toplam 384-bitlik veriyi aynı anda FPGA'ye aktarabilir. Gömülü FPGA'yi programlama için OpenFPGA'deki beş programlama protokolünden QL hafıza bankası seçildi. Programlanabilir hafıza için latch yapısına ve kaydırma kayıtçısı kullanılabilmesine izin verdiğinden dolayı diğer protokollerden daha az alan kaplamaktadır. Sistem içindeki hafızada üçe kadar bit akışı dosyaları saklanabilir, bu verilerle çalışma sırasında FPGA tekrar programlanabilmektedir. Tercih edilen protokole uygun FPGA'yi programlamayı sağlayan bir devre de bu çalışmada tasarlandı. Programlanabilir her eleman word line (WL) ve bit line (BL) sinyalleriyle kontrol edilir. Elemanlara yazılacak veri BL, yazılacak elemanları seçen adres bilgisi WL kaydırma kayıtçılarına yüklenir. Kaydırma kayıtçılarına yükleme bittikten sonra WL için etkinleştirme sinyalinin verilmesiyle istenen adresteki latchlere yükleme yapılır. Bütün elemanlar bu yöntem izlenerek 38849 saat vuruşu içinde programlanabilmekte.

Tasarlanan sistem, Cadence Xcelium'da davranışsal simülasyonda dört farklı uygulamayla test edildi. Uygulamalar; gömülü FPGA beraber kullanılmış versiyonlarıyla tamamı işlemcide çalıştırılan, yazılım versiyonları performans açısından karşılaştırıldı. Gömülü FPGA'den destek alındığı takdirde Advanced Encryption Standard (AES) şifrelemede 3,14 kat, CRC-32'de 8 kat, Finite Impulse Response (FIR) filtrede 2,8 kat, 2 boyutlu konvolüsyon işleminde 1,36 kat performans artışı gözlemlendi. Son olarak yapılan sistemin Synopsys Design Compiler (DC) ve IC Compiler II (ICC2)'de fiziksel tasarımı yapıldı. Fiziksel tasarım; yerleştirme ve rotalama akışları uzun sürdüğü için lojik blok, FPGA ve kırmık üstü sistem şeklinde üçe aşamaya ayrılmıştır. CLB, DC'de 100 MHz saat sinyali için sentezlendi, ICC2'de 60 μm kenar uzunluğuna sahip bir kare şeklinde yerleşim planına serimi yapılmıştır. Bu sırada tasarımdaki homojenlik bozulmaması için yerleştirme, saat ağacı sentezi ve rotalama aşamalarında buffer ekleyen ve bloklar arasındaki bağlantıları değiştirebilecek optimizasyonlar devre dışı bırakıldı. Kritik yolları azaltarak ICC2'nin performansını iyileştirmek için herhangi bir anahtarlama aktivitesi göstermeyen latch çıkışlarından yapılan

zamanlama analizler kapatıldı. Güç hatları, ızgara şeklinde en yüksek iletkenliğe sahip olan en üstteki iki arka yüz katmanında, standart hücreleri besleyen hatlar ise hücrelere en yakın katmanda oluşturuldu. Lojik bloğun serimi bittikten ve karakterize edildikten sonra FPGA, lojik blok ile aynı saat frekansında sentezlendi. ICC2'de lojik bloklara uygulanan aynı ayarların kullanılmasının yanı sıra anahtarlama bloklarındaki çıkışlar kombinazonel döngüler oluşturmaması için kapatılmıştır. FPGA için makro yerleştirmesi OpenFPGA akışından alınan yerleşim planına uygun şekilde 1.5 mm kenar uzunluğunda bir kare alana yerleştirildi. FPGA de sabitlendikten sonra kırmık üstü sistem için akış koşuldu. Tasarımda potansiyel kombinezonal döngüler FPGA içinde kaldığından ve optimizasyonlar işlemcinin performansına katkıda bulunduğundan önceden tercih edilen zamanlama analizlerinin ve optimizasyonları kapatılmasına ihtiyaç duyulmamıştır. Yerleşim planlaması sırasında akışın sorunsuz geçmesi için FPGA ortaya yakın, program hafızası ve bit akış dosyaları için SRAM blokları tasarımın kenarlarına yerleştirilmiştir. Böylece, tüm tasarım 100 MHz saat frekansı için 4.84 mm^2 alana yerleştirilebildi.

Sonuç olarak, kullandığımız açık kaynak araçlarla ve tasarımlarla çeşitli uygulamalar için gömülü FPGA içeren bir kırmık üstü sistem yapılabilir. Ancak ticari FPGA'lerdeki farklı anahtarlama bloğu modelleri, kombinasyonel döngü oluşturmayan RTL kodu, saat ağacı bölgeleri gibi özellikler OpenFPGA'de desteklenmediği için ticari FPGA'lerle yarışacak tasarımlar elde edilememektedir. Bunun yanında sistemin fiziksel tasarımı, zamanlama kısıtlamaları haricinde OpenFPGA'in araçları kullanılmadan yapılmıştır. Topluluğunun ve geliştiricilerin projeye daha fazla katkı vermesi durumunda, kullanılan açık kaynak araçlar daha geniş kullanım alanları elde edebilir.

1. INTRODUCTION

There has been increasing demand for computing power in recent years due to the increasing number of Internet of Things (IoT) devices and artificial intelligence (AI) usage. According to Dario Amodei et al., the computing power usage for AI training has doubled every 3.4 months since 2012 [1]. Figure 1.1 shows the graph for the usage of computation in AI training.

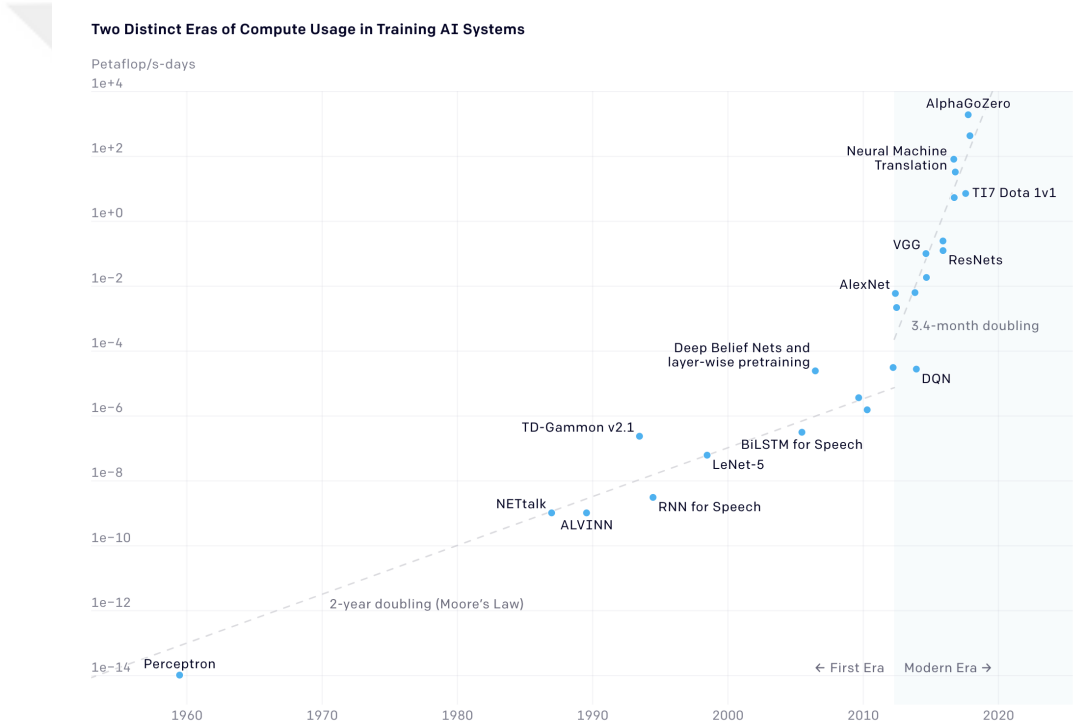


Figure 1.1 : Computing power demand over the years between 1960 and 2020 in AI training [1].

Field programmable gate arrays (FPGA) are well-known solutions to this demand with their high bandwidth, concurrent computing capability, and reconfigurability features. However, FPGA represents a challenging device from multiple perspectives. FPGAs require multiple power levels to work with and consume large amounts of power when compared to microcontrollers (MCU) and application-specific integrated

circuits (ASIC); this results in extra cost and development time to design an adequate high-speed printed circuit board (PCB). As an integrated circuit (IC), they have a large footprint, take up a lot of space, and their price per unit is very high, making them expensive to use in a high-volume project. Embedded FPGA (eFPGA) aims to solve these issues by integrating the fabric inside system-on-chips (SoC). Vendors can create smaller, less power-consuming fabrics to meet customer needs. Furthermore, eFPGAs have higher bandwidths than discreet ones because they are placed in the same die. EFPGAs can be hard intellectual properties (IP), as most FPGAs, or soft IPs. Hard IP eFPGAs are custom-made for a specific PPA and technology node. Soft IPs are made by RTL code, can be tuned to different architectures, and are ported to different technology nodes more easily than hard IPs.

In this study, a system-on-chip design with soft-core eFPGA IP is presented. There are similar studies in the literature, but this study intends to show how an eFPGA IP is designed with open-source tools and designs. Aside from the eFPGA IP, the SoC consists of a processor, memory, and UART. The processor is CVA6, an open-source 64-bit RISC-V core. The eFPGA IP was generated by OpenFPGA, an open-source FPGA fabric and bitstream generator. Advanced eXtensible Interface 4 (AXI4) of Advanced Microcontroller Bus Architecture (AMBA) protocol was used for on-chip communication. After that, the four applications were run on the designed system in a behavioural simulation environment and compared between implementations. Finally, the synthesis and physical design of the SoC were done with commercial 16-nm standard cells.

The thesis is structured as follows: Chapter 2 gives background information about eFPGA, FPGA architecture, and open-source tools used in the thesis. Chapter 3 describes the details of the proposed SoC. The benchmark and physical design results are described in Chapter 4. The thesis concludes in Chapter 5.

2. BACKGROUND

This section explains what an embedded FPGA is, its architecture and the purposes of the open-source FPGA tools used in this study.

2.1 Embedded FPGA Architecture

Embedded FPGA (eFPGA) is an IP core that contains programmable logic blocks. By integrating eFPGA into the system on chips (SoC), the user can implement digital circuits on it at any time and use it with the other blocks of SoC. They have a wide range of applications, such as wireless communication, artificial intelligence, and edge computing. A block diagram for eFPGA in an SoC is shown in Figure 2.1.

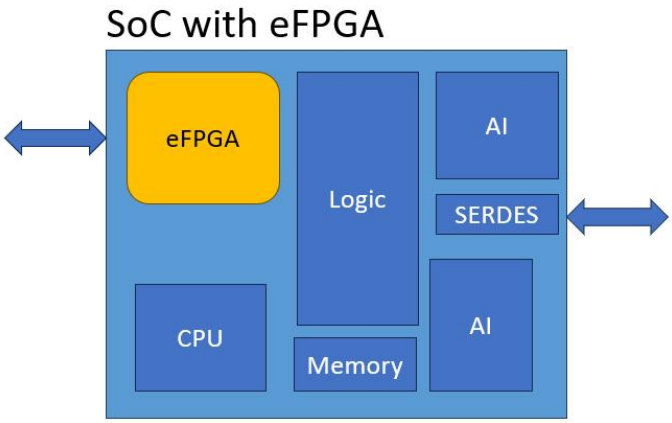


Figure 2.1 : Embedded FPGA in an SoC [2].

EFPGAs have advantages in performance and cost. Since the eFPGA IP is connected to the bus interface directly, FPGA-ASIC data transfers are faster than the performance of the system with discrete FPGA and ASIC. Some algorithms such as multiply-accumulate, cryptography, and error correction coding (ECC) perform better if they are implemented concurrently; thus, implementing these algorithms on FPGA rather than processors decreases latency and increase throughput in the

system-on-chips. Discreet FPGA chips require extra circuitry and specialized PCBs to supply power and run with a processor flawlessly, but the extra circuitry will not be needed if eFPGA is used in the SoC, and the cost and the time-to-market will decrease. Additionally, using an eFPGA can be more efficient than using multiple accelerators in terms of area because every accelerator covers a space in die area, using one eFPGA and reconfiguring for the tasks can be a cheaper solution.

Using eFPGA has disadvantages too. Integration can be hard in the physical design stage; eFPGAs need different clock domains and voltage domains from the other blocks. Also, eFPGA with abundant resources such as lookup tables (LUT), digital signal processor blocks (DSP), and block RAMs can consume more power in return for high performance. Thus, designers should be aware of PPA metrics and decide whether power or performance is more important.

Embedded FPGA architecture is similar to discreet FPGA's. There are two most used architectures by their interconnects, hierarchical and island style. In hierarchical architecture, logical blocks are connected to each other by the switch boxes, the switch boxes are connected to each other with a greater switch box, such as can be seen in Figure 2.2. Routing between two near logical boxes requires routing through a switch box with small wires. On the other hand, routing between two distant logical blocks requires a connection that traverses between different hierarchies with long wires. Due to the long routing costs and high resistivity of interconnect wires in advanced semiconductor technology nodes, the hierarchical architecture is not preferred in FPGAs over 9K look-up tables (LUTs) but can be preferred in smaller FPGAs.

In the island style; logic blocks are laid as in a two-dimensional array, the connection boxes are placed between the logic blocks and the switch boxes are placed between the connection boxes. The routing is made through switch boxes, connection boxes, and various lengths of wires. Since there are no hierarchies between blocks, the routing can use fewer wires than the hierarchical architecture. Also, the placement engine in FPGA computer-aided design (CAD) software helps to minimize routing delay costs by placing involved logic blocks in the chip. Because of this, the island style is more used in the current FPGAs. It comprises configurable logic blocks, switch

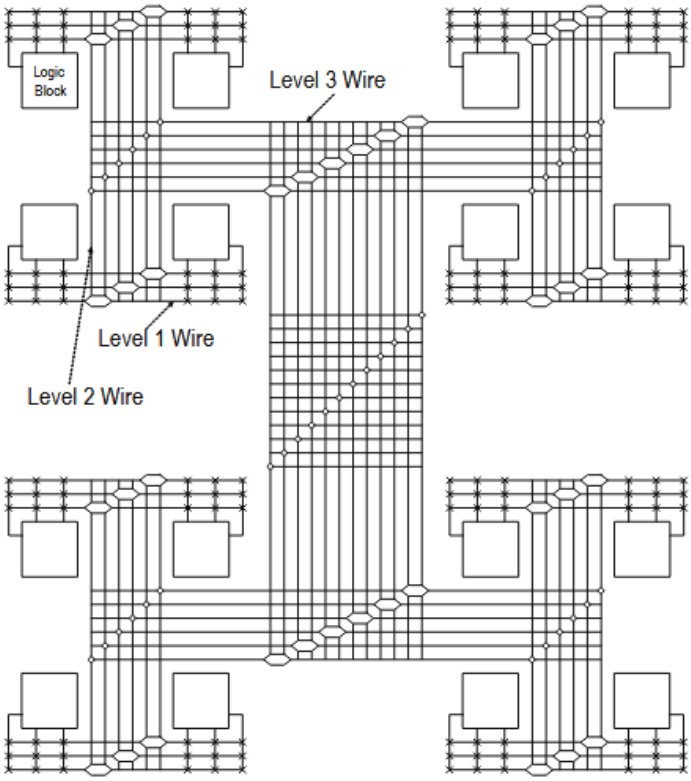


Figure 2.2 : Hierarchical FPGA architecture [3].

boxes, connection boxes, routing segments, input-output blocks, and a configuration circuit. An exemplary island-style FPGA is shown in Figure 2.3. The components of the island-style architecture will be presented in the subsequent subsections.

2.1.1 Configurable logic blocks

Configurable logic blocks (CLB) are where logical functions are implemented in an FPGA fabric. CLBs are compromised of look-up tables (LUTs), flip-flops (FF) and local routing multiplexers. A K-input LUT can implement a K-input Boolean function by storing the truth table values in its SRAM cells. An LUT can implement 2^K different Boolean functions. When an FF and a LUT are combined, it's called basic logic element (BLE) which can be seen in Figure 2.4. If a combinational design is being implemented, only the LUT is used and the multiplexer connects the output of the LUT to the output of the BLE. On the other hand; if a sequential design is being implemented, the output of the LUT is connected to the input of the FF and the output of the FF is used as the output of the BLE. Local routing multiplexers connect an output

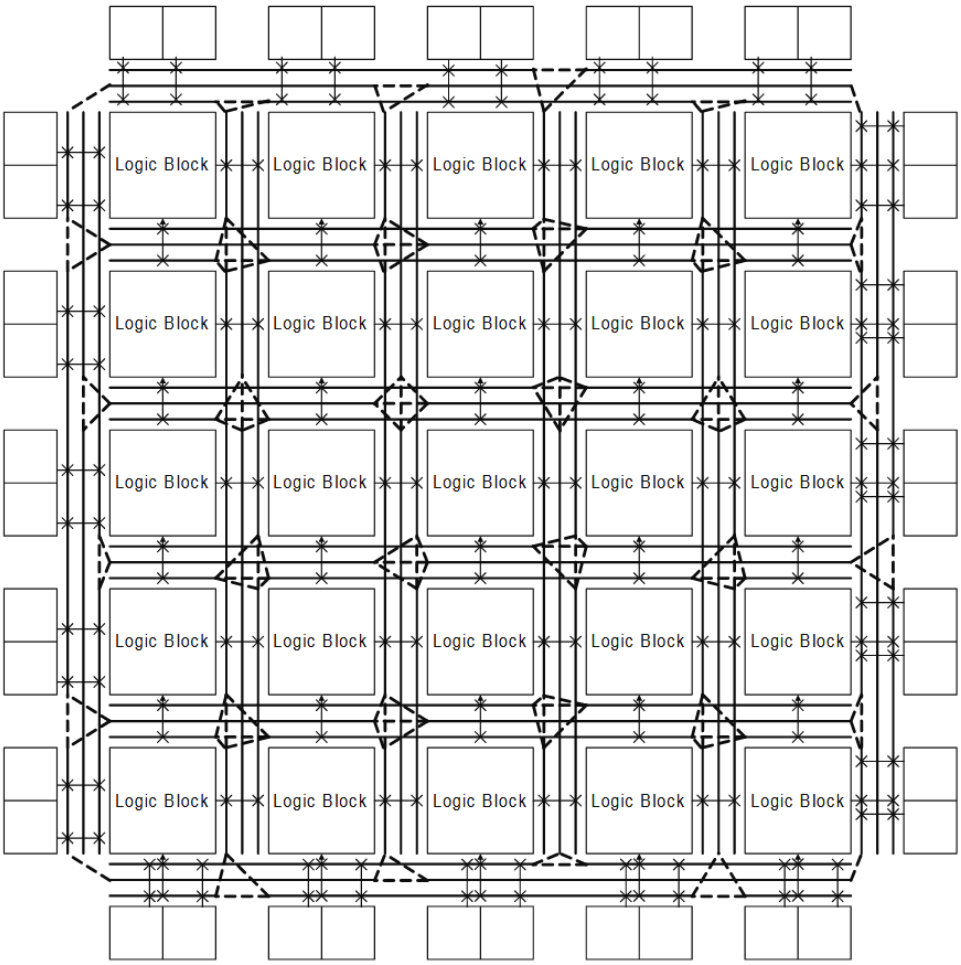


Figure 2.3 : Island style FPGA architecture [3].

of a BLE to the inputs of the other BLEs to route inside CLB without wasting routing architecture resources. Using local routing multiplexers increases performance and functionality because the path through the switch and connection boxes will be longer than the path inside the CLB.

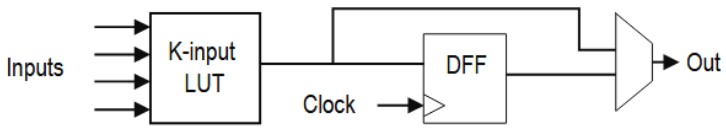


Figure 2.4 : Basic logic element [3].

Every CLB in an FPGA has N amounts of BLE with K-input LUTs. As the need for performance increases, the numbers K and N increase. If K is increased, a LUT can implement functions with more inputs. Since the LUT requires fewer routes for

implementation, the path delays decrease and the performance increases. However, the gate count increases with the power of K and it causes the performance to fall linearly. The effect of the number N and its outcomes for local routing muxes are similar to case for the number K. Ahmet and Rose found that a LUT size of 4-6 and BLE count of 3-10 deliver the best area-delay product and the relationship between input number of the CLB (I), N and K should be to $I = \frac{K}{2} * (N+1)$ to achieve the 98% utilization in the CLB [9]. An example design of CLB with local routing muxes can be seen in Figure 2.5.

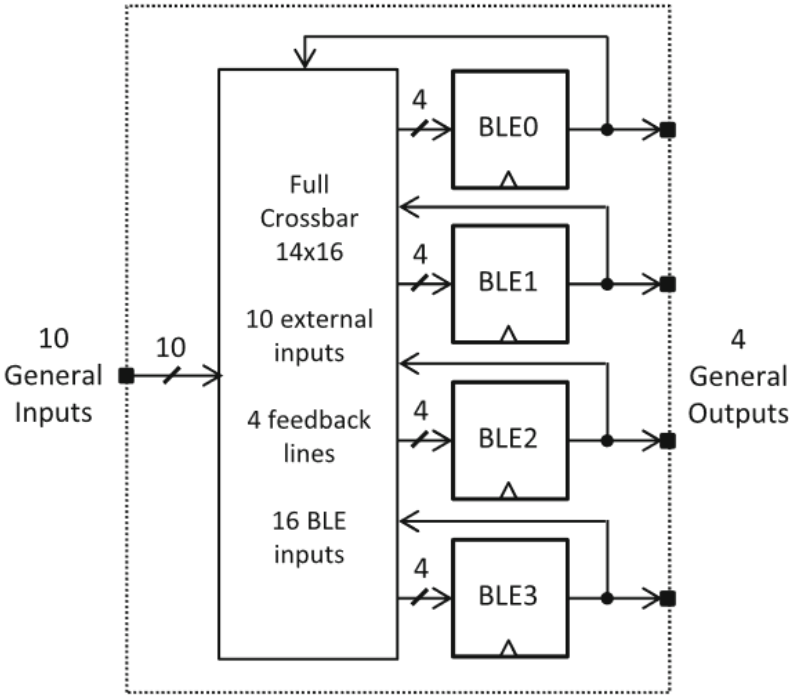


Figure 2.5 : Configurable logic block [4].

2.1.2 Switch and connection boxes

A switch box is a block that diverts signals from horizontal and vertical connection boxes, thus logic blocks can reach each other. The number of possible connections in a switch box is attributed to switch block flexibility, F_s. There are two common structures for switch boxes. The first one is disjoint, which connects wires with the same index only. The disjoint switch box with $F_s = 3$ can be seen in Figure 2.6a, the wires with index 2 can only connect with the other index 2 cables. As a result of this, signals can not reach other indexes and it will limit the routing capability in the

fabric. The other switch box is Wilton; unlike the disjoint style, it can connect to the other wires with different indexes. For example, the wire with index 3 on the left side can connect to the one with index 1 at the top, index 2 at the right, and index 0 at the bottom, as can be seen in Figure 2.6b. For this reason, the Wilton-style has more routing capability than the disjoint with the same F_s.

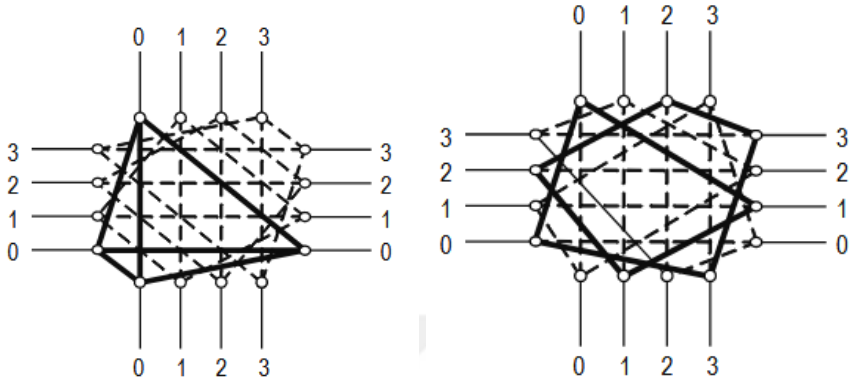


Figure 2.6 : Block diagram of (a) Disjoint and (b) Wilton [3].

There are also different types of electrical switches that can be used in switch boxes. The pass transistor is one of the design choices in early FPGAs [10], it is the smallest switch possible in CMOS processes, but every pass transistor put in the wire causes an increase in delay exponentially because of the large output capacitance. A tri-state buffer is used in later designs; it is faster than a pass transistor, but it covers more area. Nowadays, buffered multiplexers are used; they are built from buffers that cannot be tri-stated. The buffered multiplexers are superior to tri-state buffers and pass transistors in terms of area and speed. The aforementioned switch types are shown in Figure 2.7.

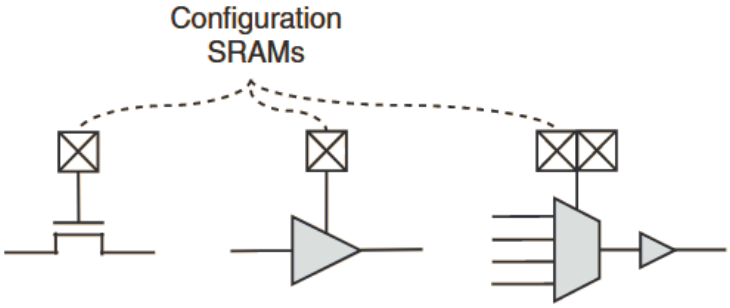


Figure 2.7 : Switch types used in switch boxes [3].

Connection boxes connect logic blocks to switch boxes. The connection boxes which connect to inputs of the logic blocks are called the input connection boxes; their routing capability is measured with the fraction of the number of switches connected to the logic blocks to the channel number and indicated with $F_{c,in}$. The connection boxes that connect to outputs of the logic blocks are called output connection boxes, their routing capability is measured similarly to $F_{c,in}$ and indicated with $F_{c,out}$. The connection boxes do not have any different architectures than switch boxes, they are merged with the switch boxes in modern FPGAs as a routing driver block [3].

2.1.3 Routing architecture

The wires that traverse around the fabric are called segments, and the number of segments is indicated by W. If W is increased, the routability and die area increase. The segments are called by their lengths. If a segment covers one logic block, it is called length-1 (L1); if a segment covers two logic blocks, it is called length-2 (L2) vice versa. An exemplary routing architecture is shown in Figure 2.8.

FPGA requires different lengths of segments for a better area-delay product. If there are too many short wires, short wires combine to form long connections and it will degrade the speed. If there are too many long wires, long wires are used to form short connections, and it will degrade the speed and waste routing resources. So, different lengths of segments should be used to optimize the fabric. Betz and Rose stated that 50% L4 with pass transistor and 50% L4 with tri-state buffers has a better performance than 100% L4 with tri-state buffers [3]. Since segments have different capacitance and resistance, using different kinds of switches can improve critical paths.

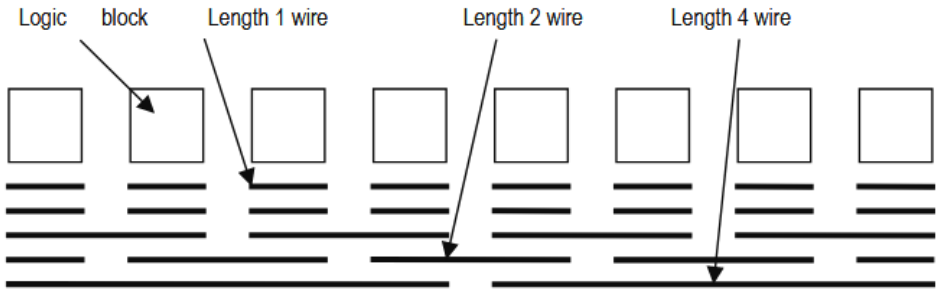


Figure 2.8 : Wire segments in the routing architecture [3].

2.1.4 Input-output blocks

FPGAs can be used in different areas of application, so they require various IO types to communicate with systems such as microcontrollers, double data rate (DDR) memories, and other FPGAs. However, supporting many I/O standards is challenging because timing, voltage and electrical specifications vary from each other. If an I/O block supports all standards, the I/O block will be enormous and will encounter implementation difficulties. So, FPGA I/Os are divided into banks with multiple purposes. The banking increases I/O counts but makes FPGA more efficient.

2.1.5 Programming architecture

FPGA programming architecture defines how SRAMs in the fabric are loaded. The programming architecture of commercial FPGAs is kept secret since they are considered as property. Thus, the research on them relies on reverse engineering and bitstream manipulation.

Reverse engineering of Xilinx 7-series configuration is made by F4PGA project, an open source FPGA EDA project [11]. The bitstreams consist of the header containing various information such as bitstream length and device model, special data to synchronize bus and data width, special instructions to reset programming circuit, and the bitstream data to be written into SRAM cells. When the bitstream is sent from the PC, the programming circuit checks if the device model is correct, and the integrity of the configuration data is checked via CRC. If there is no discrepancy, the clock of the configuration circuit is set, and the FPGA routing architecture is set to open circuit. Finally, the bitstream data started to write in adequate addresses. The SRAM data is stored with their destination row and column address.

In the 7-series, the FPGA fabric is composed of tiles; they are the base unit of an FPGA and serve as a logical resource such as CLB, DSP block or block RAM. The tiles are divided into rows according to different clock trees, and every row is divided into top and bottom with a virtual line. The row address is represented by 6-bit, one bit for the top or half and 5 bits for the row address. Within the row, tiles can be divided into the busses depending on the purpose of the tile, such as block RAM contents or CLB tiles.

The buses are represented by 3-bit. The buses are made of columns, 50 vertical tiles and 2 horizontal tiles. The bitstream data is delivered as 101 words called frames, 1 word for horizontal clock row and 100 words for the tiles and the columns can contain multiple amounts of frames. In bitstream data, frames are represented with 7 bits called minor addresses. Therefore, the bitstream frame address is shown in Table 2.1.

Table 2.1 : Xilinx 7-series bitstream frame address.

Field	Bits
Reserved	31:26
Bus	25:23
Top/Bottom Half	22
Row	21:17
Column	16:7
Minor	6:0

When the bitstream configuration is finished, the clock for tiles and the interconnect switches are activated.

2.2 Open Source Tools for FPGA Research

This section introduces the open-source tools used in this study and their purpose.

2.2.1 Yosys

Yosys is an open-source synthesis tool by Claire Wolf. It can synthesize designs written in Verilog-2005 to BLIF (Berkeley Logic Interchange Format), EDIF (Electronic Design Interchange Format) and structural Verilog files. Yosys supports technological mapping for FPGA chips such as Xilinx 7-series, Lattice iCE40 and Lattice ECP5 and can be used in open-source ASIC design flows. Yosys also contains formal methods algorithms to check properties and logical equivalence of designs [12].

2.2.2 Versatile Place-and-Route

Versatile Place-and-Route (VPR) is a computer-aided design (CAD) tool for research on FPGA architectures. It can perform packing, placement, and routing on given FPGA architectures, as well as static timing analysis on placed and routed designs. VPR takes FPGA architecture description files in XML and synthesized designs in

BLIF formats as inputs and gives reports about utilization, timing, and critical path delay as outputs in text formats. FPGAs with multi-mode logic blocks, configurable memory blocks, custom clock networks, and wire segments with varied lengths can be modelled and analyzed. The VPR graphical user interface (GUI) is shown in Figure 2.9. The tool can be found in the Verilog-to-Routing project [13].

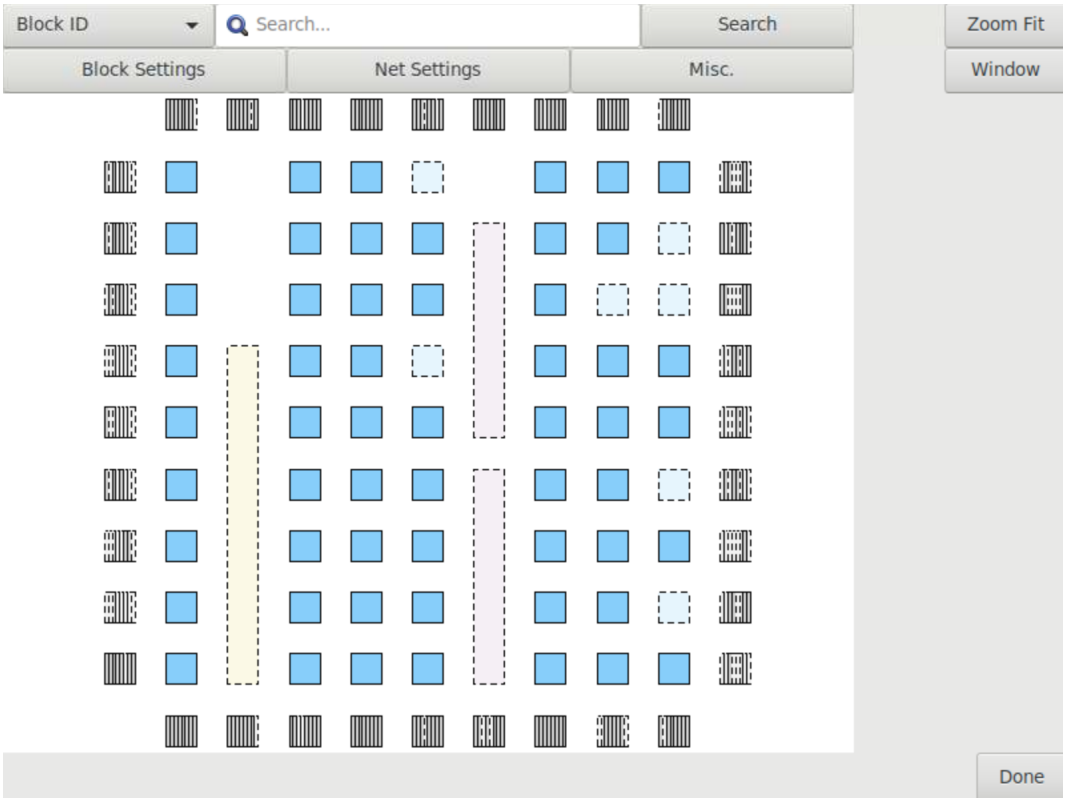


Figure 2.9 : VPR tool user interface [5].

2.2.3 OpenFPGA

OpenFPGA is an open-source FPGA IP generator by the Laboratory for NanoIntegrated Systems of the University of Utah. FPGA development requires tremendous efforts, hundreds of proficient software and hardware engineers and years of development cycles even for current industry vendors. The blocks such as LUT, FF, and multiplexers are still made with custom analog design, which is a hard way to create stable and fast circuits. EDA tools need to be set for every different FPGA architecture. Due to the reasons mentioned above, the vendors prefer to develop general-purpose products rather than application-specific products, which can be more

efficient. OpenFPGA aims to accelerate customizable FPGA design to a process that requires less effort and time than current methodologies.

OpenFPGA has two main flows. The first one is the production flow, which generates design files, testbenches, and constraints for the designed FPGA IP to create a ready-to-tapeout FPGA chip. The second one is end-user flow which generates bitstream files for FPGA architectures designed with the OpenFPGA production flow similar to Xilinx Vivado and Altera Quartus EDA tools. The two flows are shown in Figure 2.10.

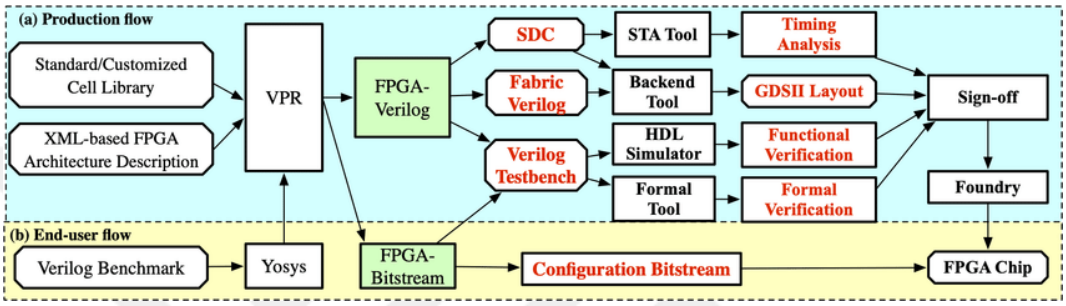


Figure 2.10 : OpenFPGA flows [6].

The production flow takes XML-based VPR and OpenFPGA architecture description files and generates fabric netlists written in Verilog, Verilog testbenches and Synopsys Design Constraints (SDC) files. The VPR architecture file defines the FPGA layout, such as the logic blocks located in the fabric and their properties, such as modes, which clock and reset signal they use, and signal inputs and outputs are defined. The OpenFPGA architecture file defines options about Verilog netlists that will be generated, such as primitive blocks, configuration protocol, reset and clock networks and associates the primitives with the blocks defined in the VPR architecture file. For instance, if there is a digital signal processing (DSP) block called dsp36 in the VPR architecture file; the inputs, outputs and mode options of dsp36 are defined in the OpenFPGA architecture file and associated with the block in the VPR and the OpenFPGA architecture files.

The generated Verilog netlists are structural Verilog files based on the layout defined in the VPR architecture file. The netlists are hierarchical designs that instantiate the primitives defined in the OpenFPGA architecture file so that the synthesis and P&R of the fabric become easier. Besides this, the bitstream configuration circuit is stitched

into the design to send the bitstream data to the SRAMs of LUTs and multiplexers. If it is set up that way, OpenFPGA can generate tile-based architectures which heavily reuse primitives and decrease the number of unique blocks.

The testbenches check if the generated fabric netlists work correctly. There are two types of testbenches: full and formal-oriented. The full testbenches configures the fabric according to its configuration protocol and starts comparing the outputs of the synthesized design and the fabric. If there is a mismatch, error flags are raised and show where the mismatch happened. The formal-oriented testbench does the same compared with the full testbench, but it skips the configuration step, and the bitstream data is instantiated in SRAMs directly. As the fabric gets larger, the bitstream and time required for the configuration phase increase exponentially so using the formal-oriented testbenches is advised to accelerate the verification in large fabrics.

The SDC files are used for synthesis, P&R, and advanced STA tools to constrain the fabric. The constraints help to get identical delays around the fabric to get accurate reports and break the combinational loops, which are not legal in backend tools and result in crashes and problems in performance.

The end-user flow takes the same inputs as the production flow, but it generates fabric-specific bitstreams ready for configuration. The flow can generate bitstreams for any architecture that can be modelled in VPR.

According to the research which is conducted by Tang et al., homogenous 20x20 and heterogenous 32x32 FPGAs whose architecture is similar to Intel Stratix IV are created in OpenFPGA. The layouts using commercial 40-nm and 12-nm technologies are finished in 24h. They stated that OpenFPGA generated fabrics are 60% larger and have 20% longer path delays than Intel Stratix IV FPGAs [6]. Besides the performance and area gap, OpenFPGA can generate unoptimized and unsynthesizable RTL designs, which depend on technology. Many built-in features do not work perfectly and sometimes require manual work. Since fabrics have combinational loops and hierarchical designs, simulations can take time and consume a lot of memory. So, simulations on personal computers can be inefficient.

3. PROPOSED SYSTEM-ON-CHIP ARCHITECTURE

The proposed SoC is based on the processor, the eFPGA IP, the memories containing bitstream and the program memory, and two UARTs. One of the UARTs is used by the processor to communicate off-chip and the other is used to load the programs to the memories. The AXI4 protocol is used to communicate between blocks in the system, the designs regarding AXI4 were taken from PULP Platform's open source available on GitHub [14]. However, only the fabric bitstream data is sent through a different bus to decrease data traffic on the AXI bus. The system was designed to work at a 100 MHz clock frequency due to the absence of a clock generation circuit. Thus, the clock received from standard IO cells is limited to approximately 100 MHz. The SoC diagram is shown in Figure 3.1. The following sections will present the components of the system.

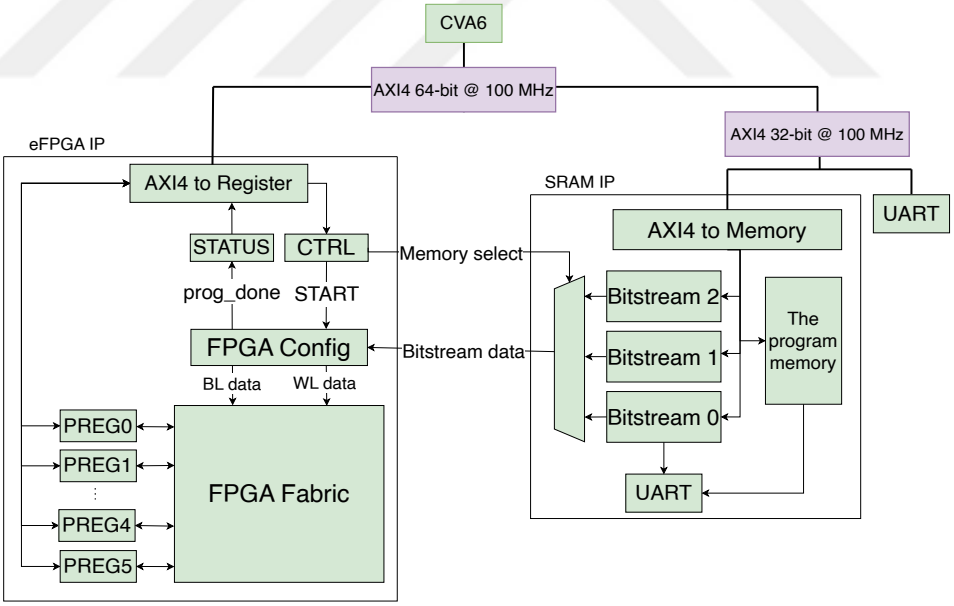


Figure 3.1 : The diagram for the system-on-chip.

3.1 Processor

CVA6 is used as the processor. It is an open-source 6-stage in-order core with RV64IMAC architecture and is capable of running Unix-like operation systems [7].

CVA6 features configurable cache sizes, branch prediction, and a hardware TLB. Since it is a configurable core, it helps to find the optimum performance for the SoC. The core is the only master of the SoC which commands the other blocks. The core stages and the other components are shown in Figure 3.2.

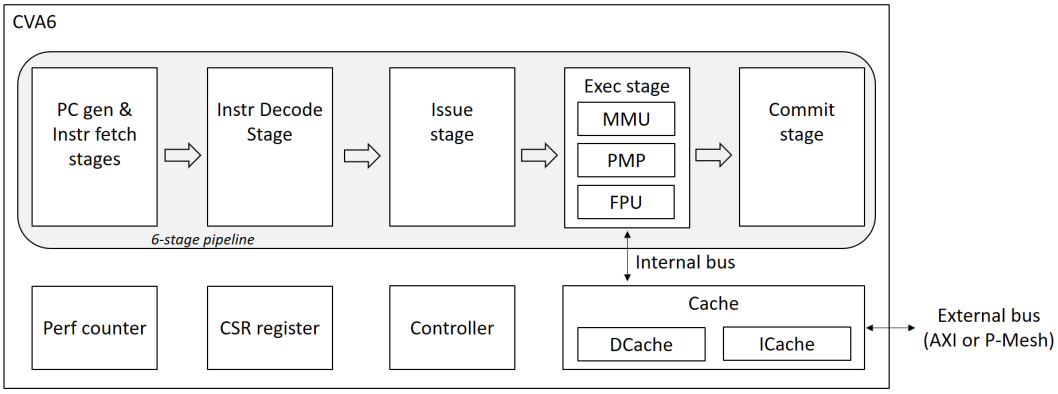


Figure 3.2 : CVA6 core structure [7].

3.2 Embedded FPGA

The embedded FPGA IP is an FPGA fabric with an interface that connects to the other SoC blocks. It can be used for hardware acceleration or IO expansion to increase system flexibility and performance. Fabric bitstreams are in the SRAM program memory and loaded together with CPU program memory. Up to three bitstreams can be stored in the memory and loaded into the fabric at any time during runtime. The IP will be examined in three components: The fabric, the interface and the configuration circuit.

3.2.1 Fabric

The FPGA fabric was generated using the production flow of the OpenFPGA framework. The fabric comprises 1960 LUTs, 1960 D flip-flops (DFF), 200 IOs and a register interface. The fabric block diagram taken from VPR GUI is shown in Figure 3.3. The LUTs have six inputs and 10 of the LUTs and the DFFs are packed as a CLB, which is called the K6N10 architecture. This architecture was chosen according to the research in [9] to get the best area-delay product. Besides that, the local routing muxes were depopulated by 50%. The full crossbar creates long critical paths and encumbers the performance but decreasing the connectivity of the crossbar by 50%

does not change the utilization ratio in the CLB, according to the research in [15]. The switch box is Wilton-style with $F_s = 3$ because the disjoint-style switch boxes' routing capability is more limited than Wilton-style. L4 segments and multiplexer switches were used for routing architecture. Although longer segments help to route efficiently, longer than L4 can not be used in this fabric because of its small size. Multiplexer switches improve performance and cover less area than tri-state buffers. In addition, the synthesis tools can optimize multiplexers to achieve better power-performance-area (PPA) results. The standard IO library of the TSMC16 PDK is used for input-output (I/O) blocks. Each cell in the I/O library has horizontal and vertical variations, so I/O blocks are grouped into two according to how they are oriented.

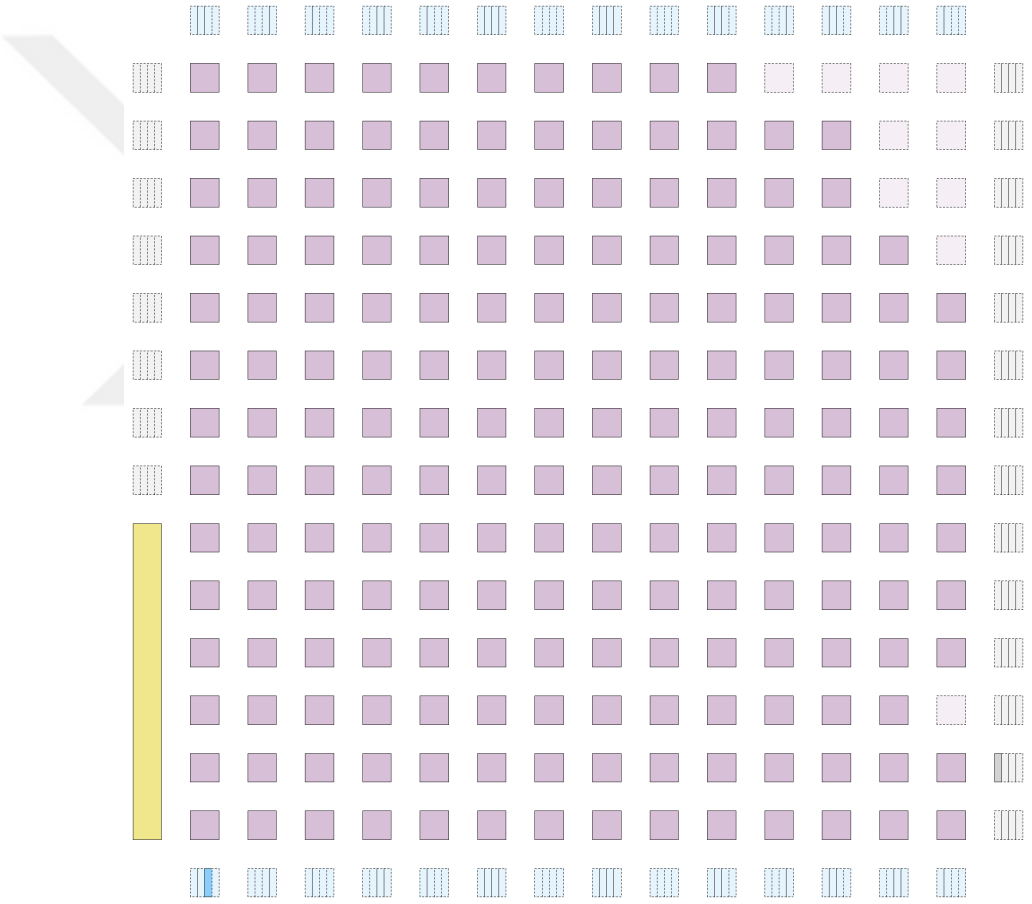


Figure 3.3 : eFPGA fabric.

The register interface is a hard block that connects the AXI4 registers to the fabric. The interface block conveys the registers' 1-bit write enable, 64-bit data write bus, and 64-bit data read bus. Thus, the user can instantiate the block to transfer data between

the fabric and the other SoC components. The block eases the transfer by supporting 384-bit data transfer simultaneously. The diagram for the block is shown in Figure 3.4.

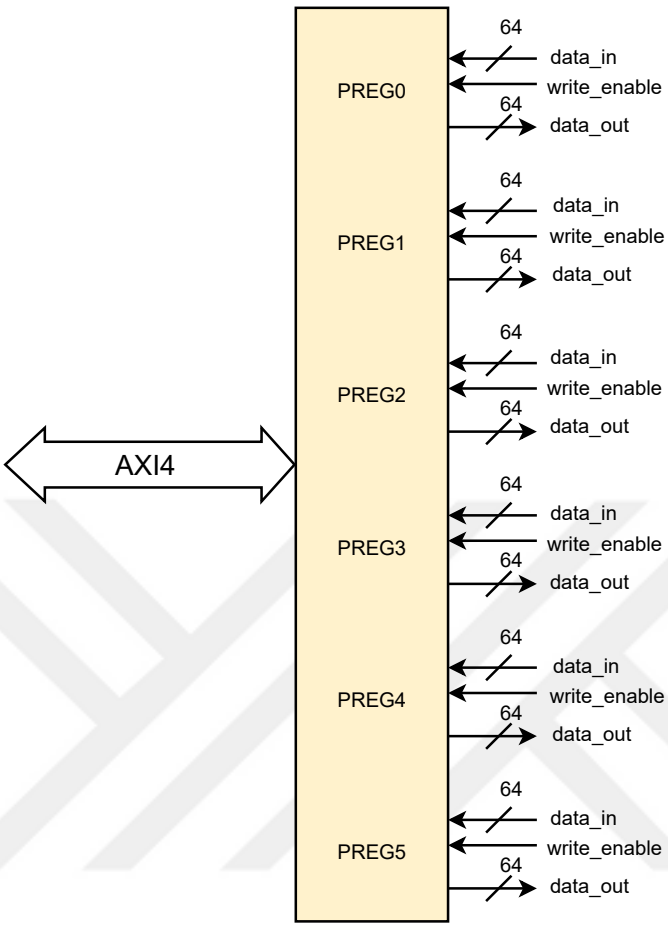


Figure 3.4 : The AXI register interface.

OpenFPGA gives five options for fabric configuration protocol. The first one is the scan chain, which connects configurable elements in LUTs, switch boxes, and connection boxes as a long chain similar to the shift register. The frame decoder protocol organizes the configurable elements as frames and banks, each of them can be reached via a specific address similar to Xilinx 7-series' configuration. The memory bank arranges the configurable elements in an array via bit line(BL)/word line (WL) decoders. The QL memory bank protocol works on the same principle as the memory bank, but the BL/WL decoders can be switched to shift registers or direct connections from the fabric's I/O, and the memories are shared more efficiently for the physical design. The last one is standalone, all configurable elements can be reachable from the IO directly [16]. In this study, the areas they cover have been considered when

selecting the protocol. The frame decoder deploys small decoders on every block in the fabric but the decoders bloat the area. The standalone does not have a protocol, so using it and writing a protocol from scratch is not preferred when others are easy to use. The scan chain covers less area than the frame decoder but the configuration elements must be flip-flops. However, latches are supported in the memory bank and the QL memory bank, covering half of the flip-flops' area. So, the QL memory bank and latches are used for the configuration. Additionally, BL/WL decoders were switched with shift registers to save the area more.

3.2.2 Interface

The embedded FPGA block is controlled by 64-bit eight registers. Two, *CTRL* and *STATUS*, are used for configuration and the others for data. The address space table for the IP is shown in Table 3.1. *CTRL* register chooses which memory will be loaded and gives the start signal to the configuration circuit. The start bit field is set to zero automatically after the start signal. The most significant 60 bits are unused, they do not affect the operation. The *CTRL* register and its bit fields are summarized in Figure 3.5.

Table 3.1 : Address spaces for eFPGA IP.

Register Name	Address
CTRL	0x0
STATUS	0x8
PREG0	0x10
PREG1	0x18
PREG2	0x20
PREG3	0x28
PREG4	0x30
PREG5	0x38

STATUS register specifies the status of the configuration register if it is busy or ready for the operation. *PREGn* for n=0..5 can connect the fabric to write and read data. The documentation for *STATUS* and *PREGn* registers are shown in Figure 3.6 and 3.7.

3.2.3 Configuration circuit

The configuration circuit loads the bitstream to the fabric according to the configuration protocol. The fabric is divided into 32 memory banks to configure faster. If there is

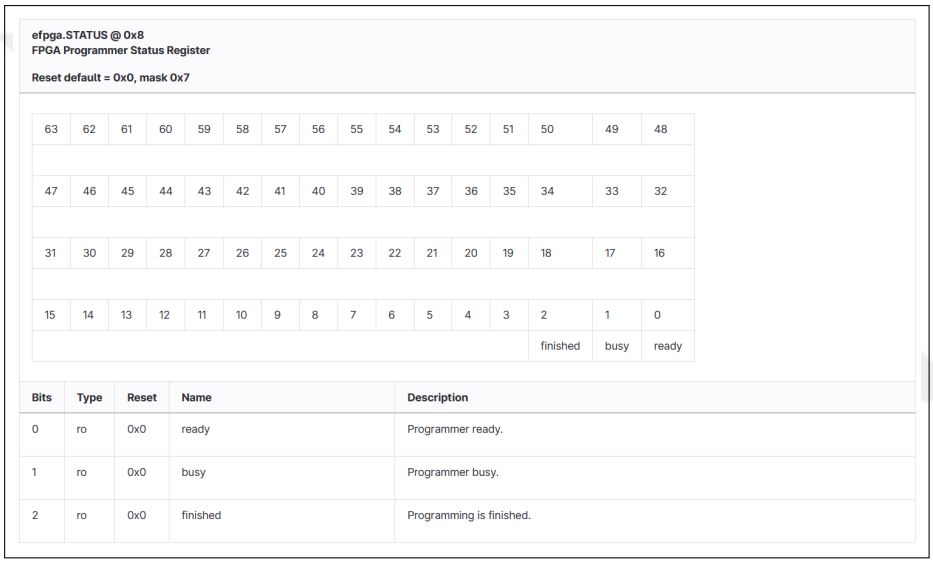
efpga.CTRL @ 0x0
FPGA Programmer Control Register

Reset default = 0x0, mask 0x7

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													program_select		start

Bits	Type	Reset	Name	Description
0	rw	0x0	start	Start programming
2:1	rw	0x0	program_select	Select programs from the memory. Only 0, 1, and 2 is selectable.

Figure 3.5 : *CTRL* register.



efpga.STATUS @ 0x8
FPGA Programmer Status Register

Reset default = 0x0, mask 0x7

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													finished	busy	ready

Bits	Type	Reset	Name	Description
0	ro	0x0	ready	Programmer ready.
1	ro	0x0	busy	Programmer busy.
2	ro	0x0	finished	Programming is finished.

Figure 3.6 : *STATUS* register.

only one memory bank, each configurable element is programmed one by one and the configuration takes longer. The number of memory banks is chosen according to the program memory's data width. The fabric is configured by using word line (WL) and bit line (BL) signals, similar to SRAM cells. The word line decides which memories are to be written to and the bit line sends the data to the memories. WL and BL data are loaded through 22-bit shift registers, each shift register transfers the data to the corresponding memory bank. This section will describe only one memory bank configuration since the other banks are configured in the same way. The diagram for the memory bank is shown in Figure 3.8.

efpga.PREG0 @ 0x10
Programmable registers (0)

Reset default = 0x0, mask 0xffffffffffffffff

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
preg_0...															
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
...preg_0...															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
...preg_0...															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
...preg_0															

Bits	Type	Reset	Name	Description
63:0	rw	0x0	preg_0	Programmable registers (0)

Figure 3.7 : *PREGn* register.

The configuration circuit starts to write BL and WL data bit by bit from the bitstream into the corresponding shift registers and waits to finish loading since 38849 clock cycles are required to load shift registers fully. When both shift registers are loaded, the enable pin of the WL register is turned on to load values in the BL shift register. Thus, the data is loaded to the configurable elements and the next data load is started. The process is the same for each data. Once all the data in the bitstream has been loaded to the fabric, the programming sequence is complete and the done signal is sent to *STATUS* register. The configuration phase wave diagram is shown in Figure 3.9.

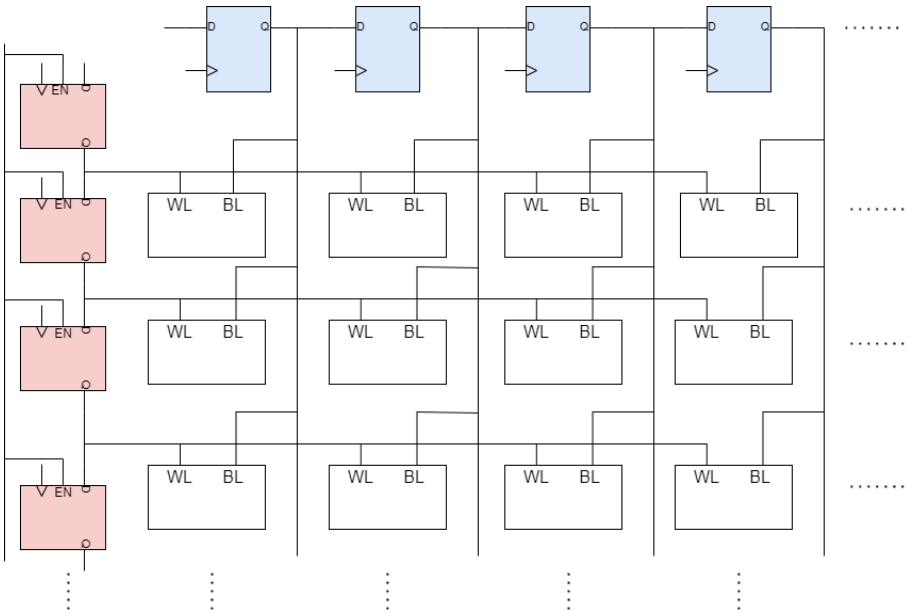


Figure 3.8 : Memory bank.

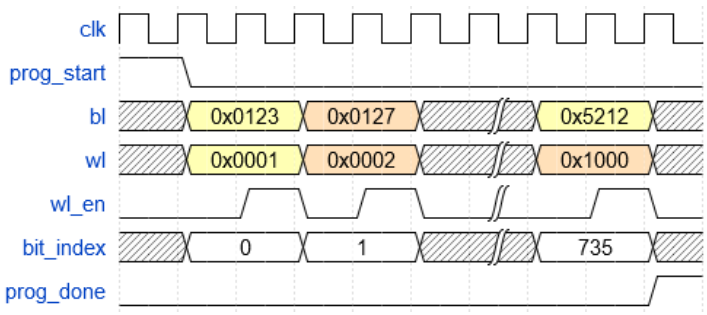


Figure 3.9 : Wavediagram for the configuration.

4. IMPLEMENTATION & RESULTS

This section discusses simulations, synthesis and place-and-route (P&R) results for the proposed system. A variety of electronic design automation (EDA) tools are used to capture the results. The EDA tools, their versions, vendors and purposes are shown in Table 4.1.

Table 4.1 : EDA tools used in the work.

Vendor	Tool name	Version	Purpose
Cadence	Xcelium	22.03-s005	Logic Simulation
Synopsys	Design Compiler NXT	U-2022.12	Logic Synthesis
Synopsys	IC Compiler II	U-2022.12	Place-and-Route
Siemens	Calibre	2023	Physical Verification

4.1 Simulation

The system-on-chip is tested in a behavioural simulation environment. Four benchmark circuits are loaded into the fabric and compared with their software implementation by clock cycles. Cadence Xcelium 2022.12 and a server are used instead of an open-source simulator and a desktop computer because the fabric contains combinational loops that increase simulation runtime and RAM usage.

4.1.1 AES encryption and decryption

The first benchmark is 640-bit Advanced Encryption Standard (AES) encryption and decryption with the cipher block chaining (CBC) mode. In CBC mode, the plaintext is divided into 128-bit chunks. The first chunk is XORed with an initialization vector (IV), which is random data to randomize the encryption process, and encrypted with a key. Then, the resulting ciphertext is XORed with the second part of the plaintext and encrypted with the same key used in the first encryption. This sequence continues until all parts of the data are encrypted. In the process of decryption, the steps are reversed. The first chunk is decrypted with the key used in the encryption, and the resulting data

are XORed with IV. The other chunks are decrypted, and the resulting plaintexts are XORed with the previous chunk of ciphertext. Figure 4.1 summarises the encryption and decryption processes. The software implementation is taken from *tiny-AES-c*, a C library of cryptographic functions available on GitHub [17].

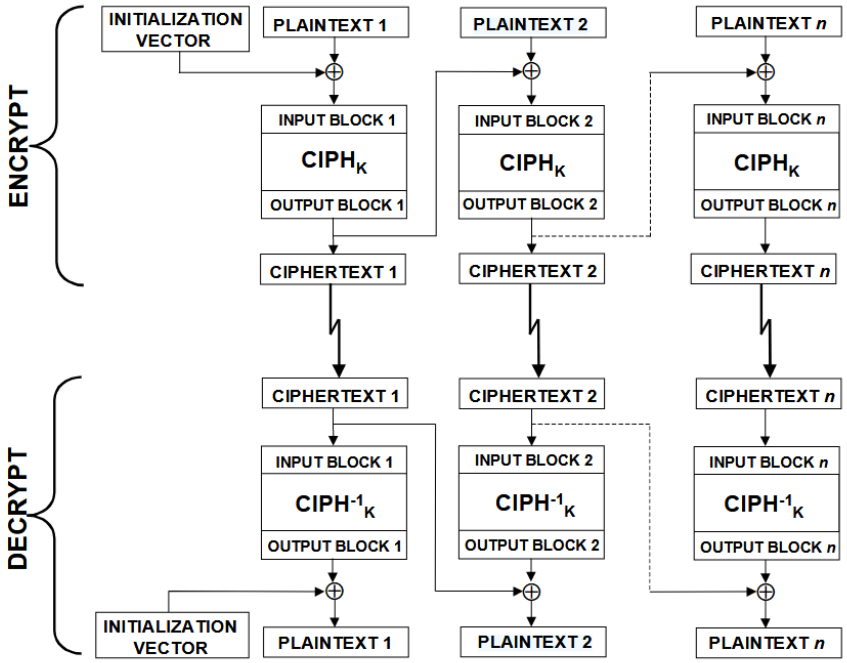


Figure 4.1 : AES encryption and decryption in CBC mode.

The eFPGA-augmented implementation is written over the same code, but a round of AES cipher is on the embedded FPGA. While computing the state, the CPU sends them to *PREG0* and *PREG1* of the eFPGA IP via the AXI bus and reads from *PREG2* and *PREG3*. The input and output require two registers since the AES states are 128-bit regardless of key size. The software implementation takes 68007 clock cycles and the eFPGA takes 21907 clock cycles, using the eFPGA increased performance by 3.14 times. The critical path for the Verilog model is between the *SubBytes* and *ShiftRows* stages, the timing report is given in Figure 4.2.

4.1.2 CRC

The second benchmark is the CRC-32 calculation. The C function for CRC-32 is taken from a CRC-32 code generator [18]. The function has arguments for *crcIn*, the previous chunk input, and *data*, the data to be calculated and returns the checksum.

```
#Path 1
Startpoint: i_aes_cipher.sa03_q[6].Q[0] (.latch clocked by clk_i)
Endpoint  : i_aes_cipher.sa03_sr[3].D[0] (.latch clocked by clk_i)
Path Type : setup

Point                                                                    Incr      Path
-------------------------------------------------------------------------------------
clock clk_i (rise edge)                                                 0.000     0.000
clock source latency                                                    0.000     0.000
clk_i.inpad[0] (.input)                                                 0.000     0.000
i_aes_cipher.sa03_q[6].clk[0] (.latch)                                  6.312     6.312
i_aes_cipher.sa03_q[6].Q[0] (.latch) [clock-to-output]                  0.108     6.420
$abc$183286$new_n1783.in[1] (.names)                                    0.382     6.802
$abc$183286$new_n1783.out[0] (.names)                                   0.149     6.951
$abc$183286$new_n1782.in[2] (.names)                                    0.225     7.176
$abc$183286$new_n1782.out[0] (.names)                                   0.149     7.325
$abc$183286$new_n1858.in[0] (.names)                                    0.519     7.845
$abc$183286$new_n1858.out[0] (.names)                                   0.149     7.994
i_aes_cipher.sa03_sub_d[3].in[2] (.names)                               0.446     8.439
i_aes_cipher.sa03_sub_d[3].out[0] (.names)                              0.149     8.588
i_aes_cipher.sa03_sr[3].D[0] (.latch)                                   0.000     8.588
data arrival time                                                                 8.588

clock clk_i (rise edge)                                                10.000    10.000
clock source latency                                                    0.000    10.000
clk_i.inpad[0] (.input)                                                 0.000    10.000
i_aes_cipher.sa03_sr[3].clk[0] (.latch)                                 6.312    16.312
clock uncertainty                                                       0.000    16.312
cell setup time                                                        -0.014    16.298
data required time                                                               16.298
-------------------------------------------------------------------------------------
data required time                                                               16.298
data arrival time                                                                -8.588
-------------------------------------------------------------------------------------
slack (MET)                                                                       7.710
```

Figure 4.2 : The timing report of the critical path in the AES Verilog model.

The software implementation calculates the CRC-32 of multiple chunks of data. The procedure is similar to the aforementioned AES-CBC, the data is split into chunks by the data size, and each data depends on the previous chunk. In the first round, the *crcIn* value must be -1 according to the standard. The result is stored in a variable to be used in the next byte after the checksum of the first data has been calculated. In the second round, the checksum from the previous round and the current chunk are passed to *crcIn* and *data* inputs, respectively. The result of this round is XORed with the previous checksum and stored. This procedure is continued until the data is fully processed. Finally, the result is bitwise inverted. The software implementation takes 12925 clock cycles in the simulation environment run on Cadence Xcelium.

In the eFPGA-augmented implementation, the procedure is the same, but the CRC-32 function is loaded to the eFPGA IP. *data* is passed from the *PREG0* register, *crcIn* from the *PREG1* and the checksum is read from the *PREG2*. The Verilog module for the function is taken from the same generator. The eFPGA-augmented implementation takes 1552 clock cycles which is 8 times faster than the software implementation in the same environment. The max frequency of the design is 689.66 MHz with the critical

path 1.55 ns, from the 63rd bit of *PREG1* to the 63rd bit of *PREG2*. Figure 4.3 shows the timing report for the critical path. The design also covers 106 LUTs and zero FF because the model is combinational. Although being a combinational circuit provides an advantage in terms of clock cycles in the simulation, its maximum frequency can be higher in pipelined designs.

```
#Path 1
Startpoint: preg0_o[63].preg1_o[57] (_AXI_SLAVE_ clocked by clk_i)
Endpoint  : preg0_o[63].preg2_i[52] (_AXI_SLAVE_ clocked by clk_i)
Path Type : setup

Point                                                              Incr      Path
-------------------------------------------------------------------------------
clock clk_i (rise edge)                                           0.000     0.000
clock source latency                                              0.000     0.000
clk_i.inpad[0] (.input)                                           0.000     0.000
preg0_o[63].clk_i[0] (_AXI_SLAVE_)                                6.312     6.312
preg0_o[63].preg1_o[57] (_AXI_SLAVE_) [clock-to-output]           0.108     6.420
$abc$886$new_n159.in[5] (.names)                                  0.271     6.691
$abc$886$new_n159.out[0] (.names)                                 0.149     6.840
i_crc.crcOut[11].in[3] (.names)                                   0.296     7.136
i_crc.crcOut[11].out[0] (.names)                                  0.149     7.285
preg0_o[63].preg2_i[52] (_AXI_SLAVE_)                             0.396     7.681
data arrival time                                                           7.681

clock clk_i (rise edge)                                          10.000    10.000
clock source latency                                              0.000    10.000
clk_i.inpad[0] (.input)                                           0.000    10.000
preg0_o[63].clk_i[0] (_AXI_SLAVE_)                                6.312    16.312
clock uncertainty                                                 0.000    16.312
cell setup time                                                  -0.014    16.298
data required time                                                          16.298
-------------------------------------------------------------------------------
data required time                                                          16.298
data arrival time                                                           -7.681
-------------------------------------------------------------------------------
slack (MET)                                                                 8.617
```

Figure 4.3 : The timing report of the critical path in the CRC Verilog model.

4.1.3 FIR filter

The third benchmark is an 8-bit FIR filter with 5 taps. The block diagram for the filter is shown in Figure 4.4. In the eFPGA implementation, the CVA6 passes the data to the filter loaded in the eFPGA. The filter has *valid_i*, a valid signal to pass the data correctly, and an 8-bit *data_i* bus as inputs. The inputs are sent via the *PREG0* register since all signals can fit in 64-bit. The output of the filter *data_o* is 8-bit and can be read from *PREG1*. The software implementation has the same functionality, but each computation is done on the CPU. Both HDL and C codes are coded in this thesis.

The software and the hardware implementations are applied to 50 samples of sinus function. While the software implementation takes 7195 clock cycles to finish, the

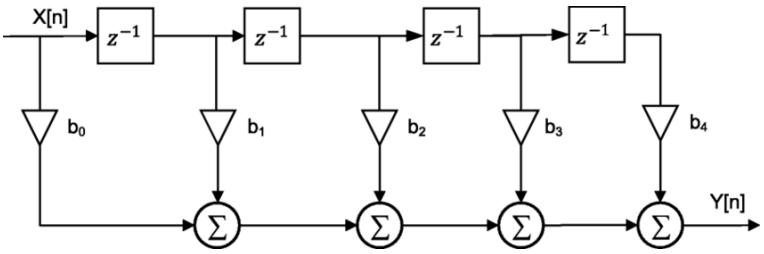


Figure 4.4 : 5-tap FIR filter [8].

eFPGA takes 2569 clock cycles, which is 2.8 times faster than the software. The timing report for the implementation is shown in Figure 4.5.

```
#Path 1
Startpoint: i_fir.reg2[1].Q[0] (.latch clocked by clk_i)
Endpoint  : preg0_o[63].preg1_i[56] (_AXI_SLAVE_ clocked by clk_i)
Path Type : setup

Point                                                              Incr      Path
-------------------------------------------------------------------------------
clock clk_i (rise edge)                                           0.000     0.000
clock source latency                                              0.000     0.000
clk_i.inpad[0] (.input)                                           0.000     0.000
i_fir.reg2[1].clk[0] (.latch)                                     6.312     6.312
i_fir.reg2[1].Q[0] (.latch) [clock-to-output]                     0.108     6.420
$abc$1711$new_n46.in[0] (.names)                                  0.324     6.744
$abc$1711$new_n46.out[0] (.names)                                 0.149     6.893
$abc$1711$new_n50.in[5] (.names)                                  0.312     7.205
$abc$1711$new_n50.out[0] (.names)                                 0.149     7.354
$abc$1711$new_n56.in[1] (.names)                                  0.298     7.652
$abc$1711$new_n56.out[0] (.names)                                 0.149     7.801
$abc$1711$new_n83.in[4] (.names)                                  0.231     8.032
$abc$1711$new_n83.out[0] (.names)                                 0.149     8.181
i_fir.o_out[7].in[1] (.names)                                     0.025     8.206
i_fir.o_out[7].out[0] (.names)                                    0.149     8.355
preg0_o[63].preg1_i[56] (_AXI_SLAVE_)                             0.411     8.766
data arrival time                                                           8.766

clock clk_i (rise edge)                                          10.000    10.000
clock source latency                                              0.000    10.000
clk_i.inpad[0] (.input)                                           0.000    10.000
preg0_o[63].clk_i[0] (_AXI_SLAVE_)                                6.312    16.312
clock uncertainty                                                 0.000    16.312
cell setup time                                                  -0.014    16.298
data required time                                                          16.298
-------------------------------------------------------------------------------
data required time                                                          16.298
data arrival time                                                           -8.766
-------------------------------------------------------------------------------
slack (MET)                                                                  7.532
```

Figure 4.5 : The timing report of the critical path in the FIR filter Verilog model.

4.1.4 2D convolution

The fourth benchmark is the Sobel filter. The edge detection algorithm was applied to a 50 by 50 image in which each pixel holds 8-bit grayscale data. The algorithm fetches a 3x3 frame from the image and applies two-dimensional convolution with two kernels, Gx and Gy shown below. The absolute values of the convolution results are summed and written to the output array. After processing each 3x3 frame in the image, the result is the input image with the edges of the objects highlighted. The input and output images are shown in Figure 4.6.

(a) (b)

Figure 4.6 : (a) The original image and (b) The resulting image after the Sobel filter.

In the eFPGA implementation, the CPU decides which frame is processed and sends them to the fabric where convolution and summation operations are performed. In software implementation, every computation is performed on the CPU. While the software implementation takes 269550 clock cycles, the eFPGA takes 197914 cycles. Using eFPGA accelerated the process by 1.36 times. The critical path in the Verilog model is between the *Gx* and the output, the timing report is shown in Figure 4.7.

```
#Path 1
Startpoint: i_sobel_core.gx_q[5].Q[0] (.latch clocked by clk_i)
Endpoint  : preg0_o[63].preg1_i[62] (_AXI_SLAVE_ clocked by clk_i)
Path Type : setup

Point                                                                  Incr      Path
-------------------------------------------------------------------------------------
clock clk_i (rise edge)                                               0.000     0.000
clock source latency                                                  0.000     0.000
clk_i.inpad[0] (.input)                                               0.000     0.000
i_sobel_core.gx_q[5].clk[0] (.latch)                                  6.312     6.312
i_sobel_core.gx_q[5].Q[0] (.latch) [clock-to-output]                  0.108     6.420
$abc$1443$new_n127.in[1] (.names)                                     0.252     6.672
$abc$1443$new_n127.out[0] (.names)                                    0.149     6.821
$abc$1443$new_n126.in[1] (.names)                                     0.231     7.052
$abc$1443$new_n126.out[0] (.names)                                    0.149     7.201
$abc$1443$new_n124.in[2] (.names)                                     0.232     7.433
$abc$1443$new_n124.out[0] (.names)                                    0.149     7.582
$abc$1443$new_n120.in[3] (.names)                                     0.087     7.669
$abc$1443$new_n120.out[0] (.names)                                    0.149     7.818
$abc$1443$new_n119.in[0] (.names)                                     0.228     8.046
$abc$1443$new_n119.out[0] (.names)                                    0.149     8.195
i_sobel_core.out[1].in[0] (.names)                                    0.295     8.490
i_sobel_core.out[1].out[0] (.names)                                   0.149     8.639
preg0_o[63].preg1_i[62] (_AXI_SLAVE_)                                 0.396     9.034
data arrival time                                                               9.034

clock clk_i (rise edge)                                              10.000    10.000
clock source latency                                                  0.000    10.000
clk_i.inpad[0] (.input)                                               0.000    10.000
preg0_o[63].clk_i[0] (_AXI_SLAVE_)                                    6.312    16.312
clock uncertainty                                                     0.000    16.312
cell setup time                                                      -0.014    16.298
data required time                                                              16.298
-------------------------------------------------------------------------------------
data required time                                                              16.298
data arrival time                                                               -9.034
-------------------------------------------------------------------------------------
slack (MET)                                                                      7.264
```

Figure 4.7 : The timing report of the critical path in the Sobel filter Verilog model.

4.2 Physical design

The physical design of the SoC is made using TSMC 16nm FinFET standard cells and Synopsys tools, Design Compiler for logic synthesis and IC Compiler II for place-and-route flow. The physical design flow is divided into CLB, FPGA fabric and SoC. The CLB is a design instantiated multiple times, so applying P&R flow on CLB and integrating it at the top level decreases the runtime. Also, FPGA fabric contains many combinational loops that increase runtimes during flow. Solving some of the loops beforehand makes top-level integration easier. The FPGA fabric is hardened for the same reason. The design flow is shown in Figure 4.8.

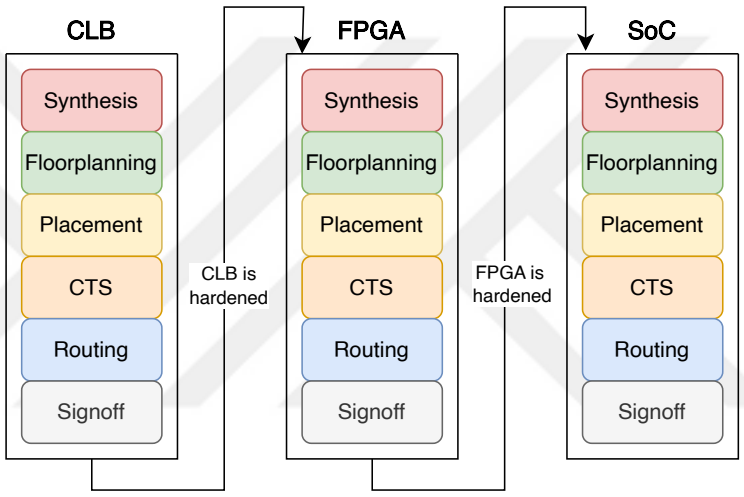


Figure 4.8 : The physical design flow.

4.2.1 Configurable logic block

Firstly, CLB was synthesized at a 100 MHz clock frequency in Synopsys Design Compiler. Afterwards, the synthesized design was opened in IC Compiler II. The design was constrained to analyze in one mode and three Process-Voltage-Temperature (PVT) corners for a total of 3 scenarios. The mode was named *func*, which checks the timing of flip flops in the BLEs while the FPGA works after the configuration and disables the arcs on the configuration circuitry. Three PVT corners are *tt_0p8v_25c*, *ffgnp_0p72v_m40c*, and *ssgnp_0p88v_125c*. The floorplan is a square with sides of 60 μm each. Two μm spaces are left from each side to route ports easily. The ports of the

design were placed according to the places specified in the VPR architecture file. BL and WL are unspecified in the VPR file and placed on the top and the left sides. The mesh grid was chosen for the power plan because it covers less area than the power ring method. The grid was routed on M8 and M9 which have higher conductivity than the other metals. Standard cells have embedded 90nm power rails, but to ensure connectivity, standard cell rails have been routed on M1 with a width of 32nm. It is the widest possible power route in M1 since minimum spacing DRC errors occur between pins in standard cells and wider traces than 32nm. The grid and the standard cell rails are connected by vias from M9 to M1. The placement is made with a 70% utilization ratio to avoid encountering congestion issues or otherwise wasting the area. The congestion map is shown in Figure 4.9.

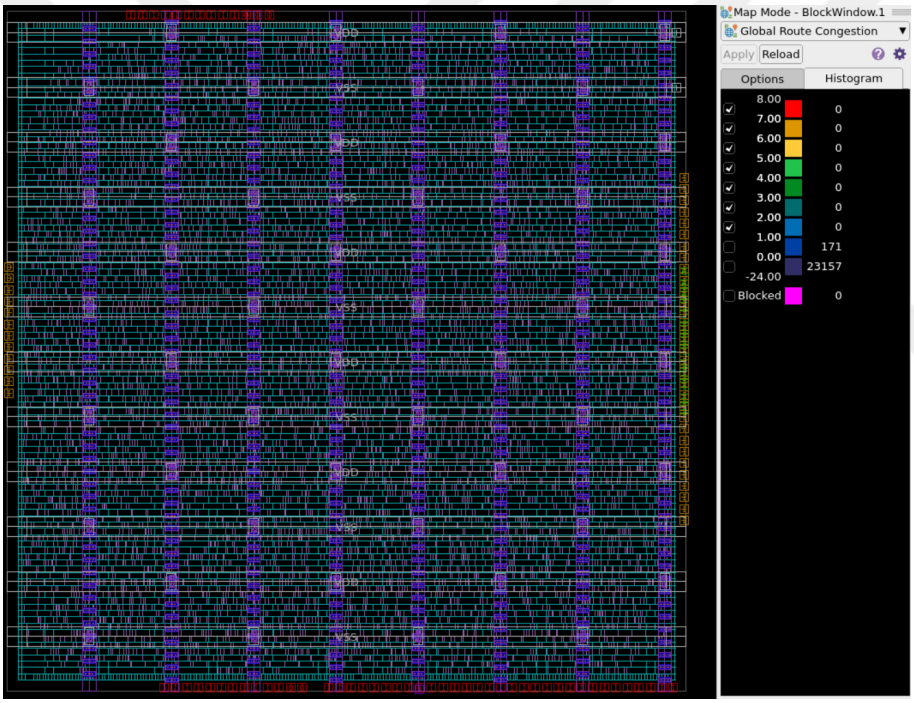


Figure 4.9 : The congestion heat map of the CLB.

In the clock tree synthesis (CTS) stage, the clock and data are synchronized with the concurrent clock and data (CCD) method of Synopsys by changing the driving strengths of flip flops or inserting buffers. However, the FPGA fabric should be homogeneous so positive skew insertion is turned off. Non-default routing rules (NDR) are applied on the clock to decrease crosstalk between the clock and signal nets; the width and spacing of the clock nets are two times larger than default values in M5,

M6, and M7 layers. The NDR is not applied on M2, M3 and M4 since their design rules are stricter and consume more routing resources. In the routing stage, routing is done between the M2 and M7 layers because M1, M8, and M9 are reserved for power. If these layers are used for routing, the Zroute routing engine will try to draw eligible routes that are free from DRC and timing violations, and the runtime will be increased greatly. Besides, routing in the M1 layer is challenging in advanced nodes due to high density and complex design rules. So, standard cell connections are made with only vias from the M2 layer. In the signoff stage, the ECO fusion flow is run if any path is violated, filler cells are inserted and redundant vias are inserted to increase reliability. Finally, DRC and LVS flows are run on Siemens Calibre tools, and no errors are detected. The physical design of the CLB with DRC and LVS checks takes 45 minutes to finish. The final layout is shown in Figure 4.10. Dummy metal, OD, and cut OD insertion are not included in this flow; they will be inserted in the top-level design.

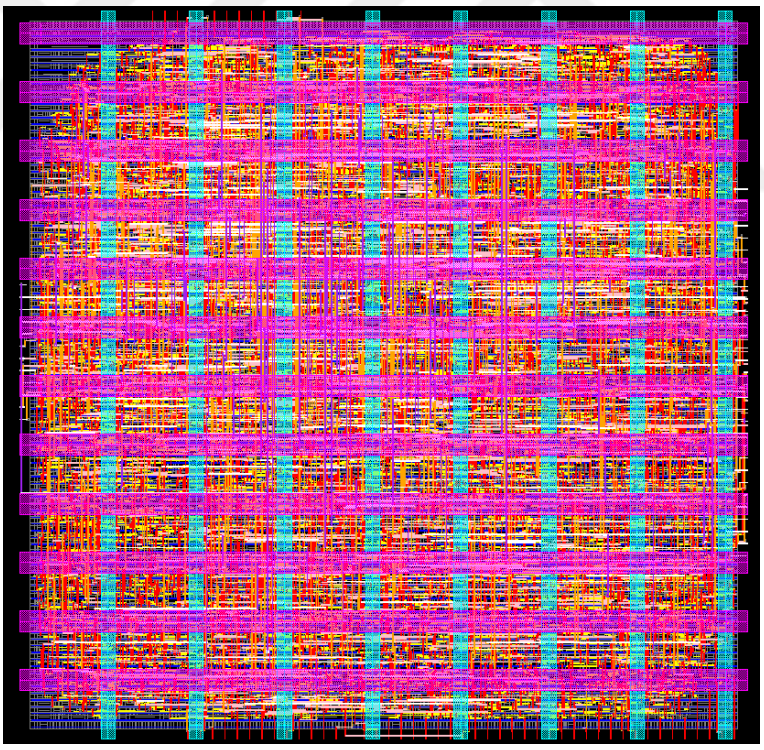


Figure 4.10 : The CLB final layout.

4.2.2 FPGA fabric

After the physical design of the CLB was completed, the FPGA top was implemented. The synthesis is done in 100 MHz. Aside from the timing arcs from latches, the

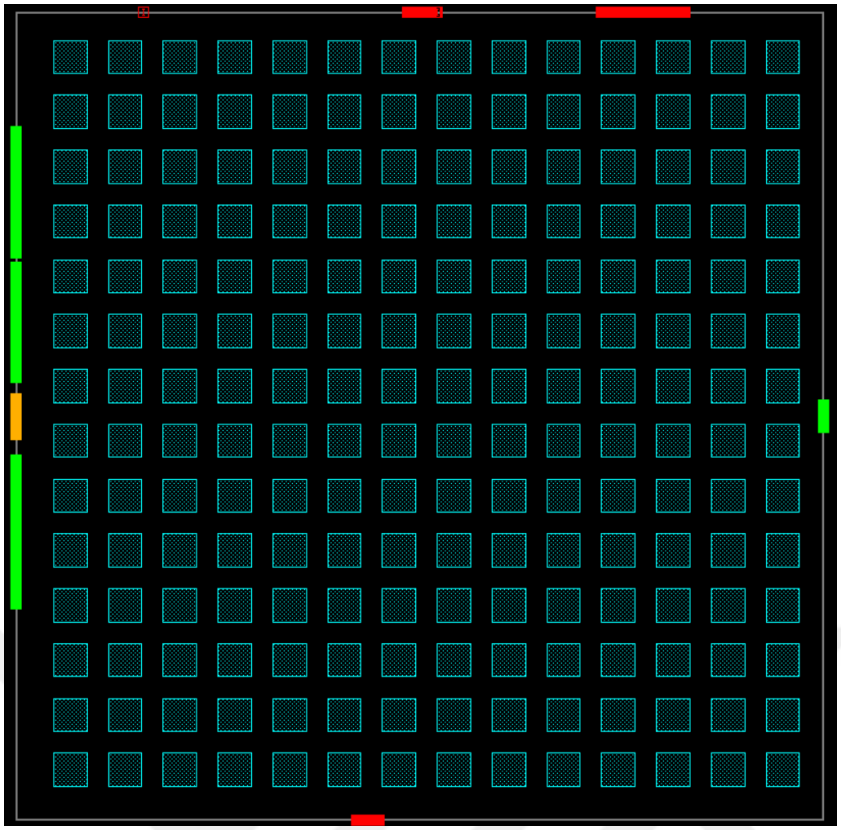


Figure 4.11 : The FPGA top macro placement.

timing arcs in the switch box's outputs are disabled to prevent combinational loops. The difference between FPGA fabric and CLB, there are two clocks in the fabric, which are *prog_clk*, the programming clock for the shift registers and *op_clk*, the clock for flip-flops in CLB. They are constrained in the same frequency and set as logically exclusive, they do not have paths that cross each other. After the synthesis was completed, the implementation was continued on ICC2. The same scenario setup was done as made in the CLB, three PVT corners with one mode. In floorplanning, the CLBs were placed according to their instantiation names in the Verilog design file. For instance, *grid_clb_0_0* should be placed in the bottom left corner of the fabric, the *grid_clb_0_1* should be placed above *grid_clb_0_0*, and *grid_clb_1_0* should be placed to the right of the *grid_clb_0_0*. Each CLB is placed with 40 μm intervals to fit switch and connection boxes between them. The macro placement is shown in Figure 4.11. The same settings for power plan, placement, CTS and routing were applied here to maintain homogeneity. Finally, the FPGA fabric was laid into a 1500 μm by 1500 μm area. The final layout is shown in Figure 4.12.

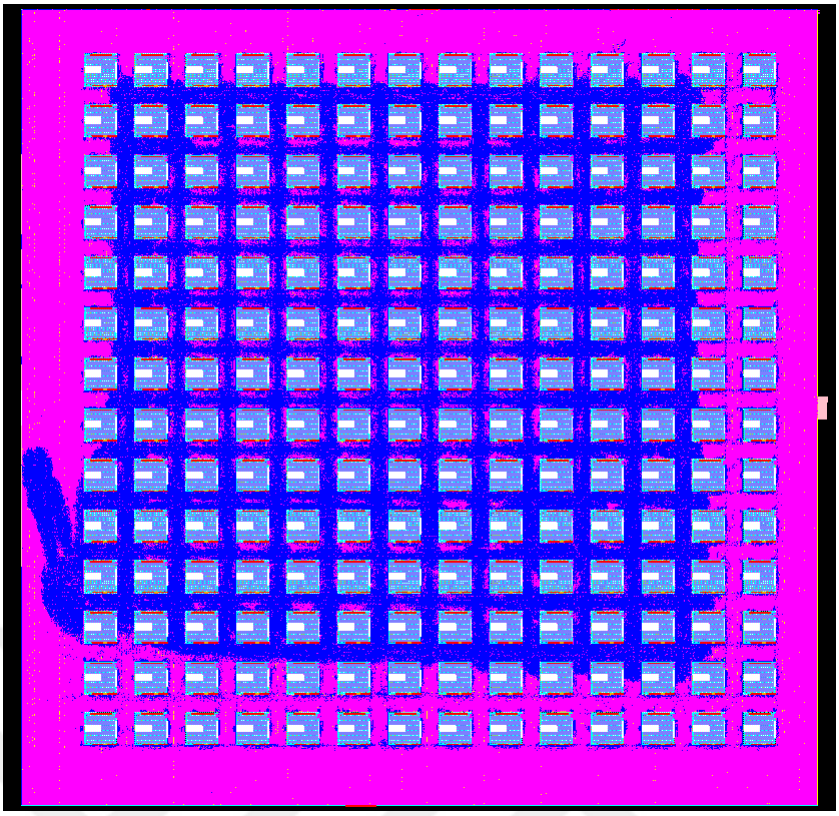


Figure 4.12 : The FPGA top final layout.

4.2.3 Top level design

After the physical design of the FPGA fabric was completed, the top module was implemented. The fabric and SRAM memories are used as black boxes in the top module because the fabric takes too much time to finish, and the memories increase the runtime and waste die area. The black-box modules were replaced in the RTL netlist, and the Liberty files were included in the libraries to overcome the issues. The design was synthesized with a 100 MHz clock frequency. Since there is one clock, and no possible combinational loops are present, *set_disable_timing* commands are not added. In IC Compiler II, the same modes and corners are set up. The fabric was placed in the center of the core area. The IOBs were instantiated in the IO ring according to their places relative to the fabric. The SRAM blocks were positioned at the left side of the core area with their pins facing the core area. Placement blockages are added around the hard macros to hinder from overlapping implant layers of standard cells and the CLB. The floorplan for the top module is shown in Figure 4.13.

The power ring and meshes distribute the power around the die. The power pads are connected to the power ring, and the ring is connected to the meshes. The mesh width, pitch and spacing are the same as the CLB's sizes, so placing it in the mesh is sufficient to ensure connectivity. Additional meshes were compiled for the SRAM blocks since they require power and ground from the M6 layer. The turned-off options in the P&R steps were turned on in this design because local skewing and high fanout net breaking helps the optimization, and homogeneity was not sought on the top design. The same CTS NDR rules used in the CLB design are applied to the top module. After routing and signoff steps, the chip covers around 4.84 mm^2 with a 100 MHz clock frequency. The final layout is shown in Figure 4.14.

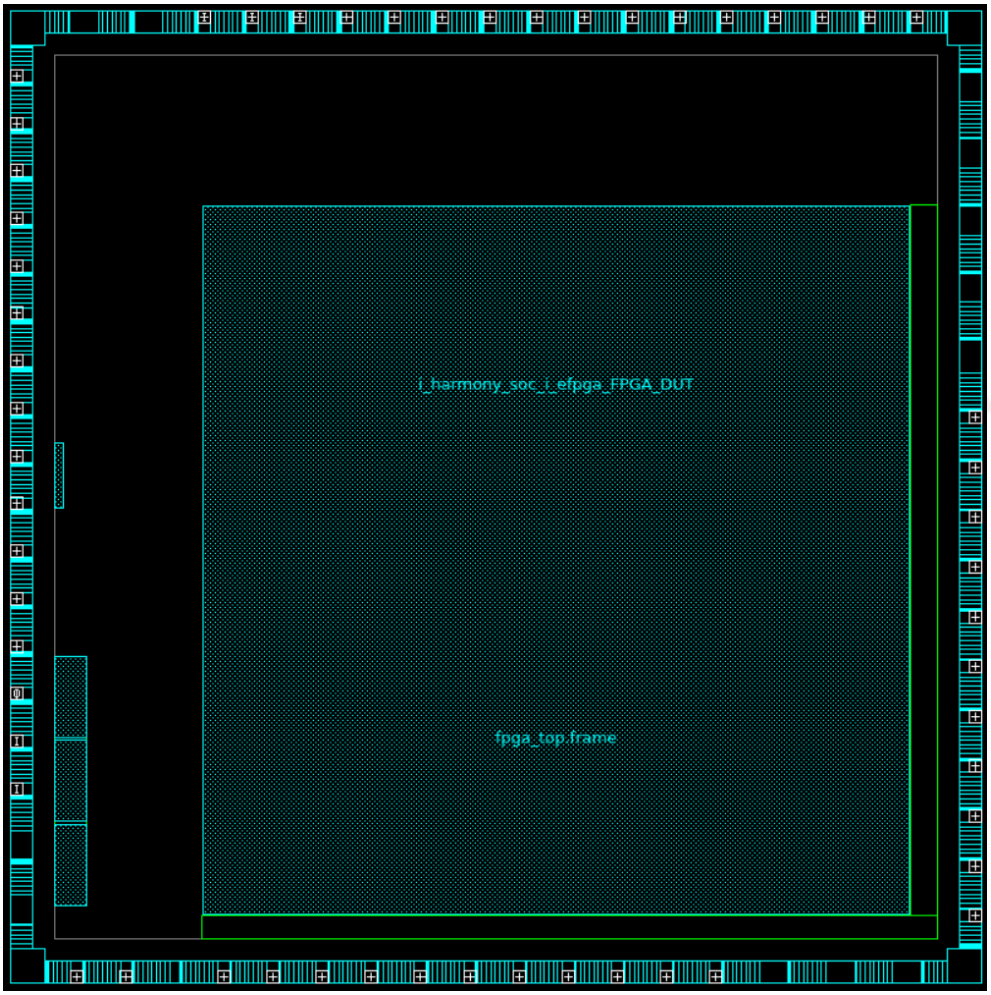


Figure 4.13 : The SoC top floorplan.

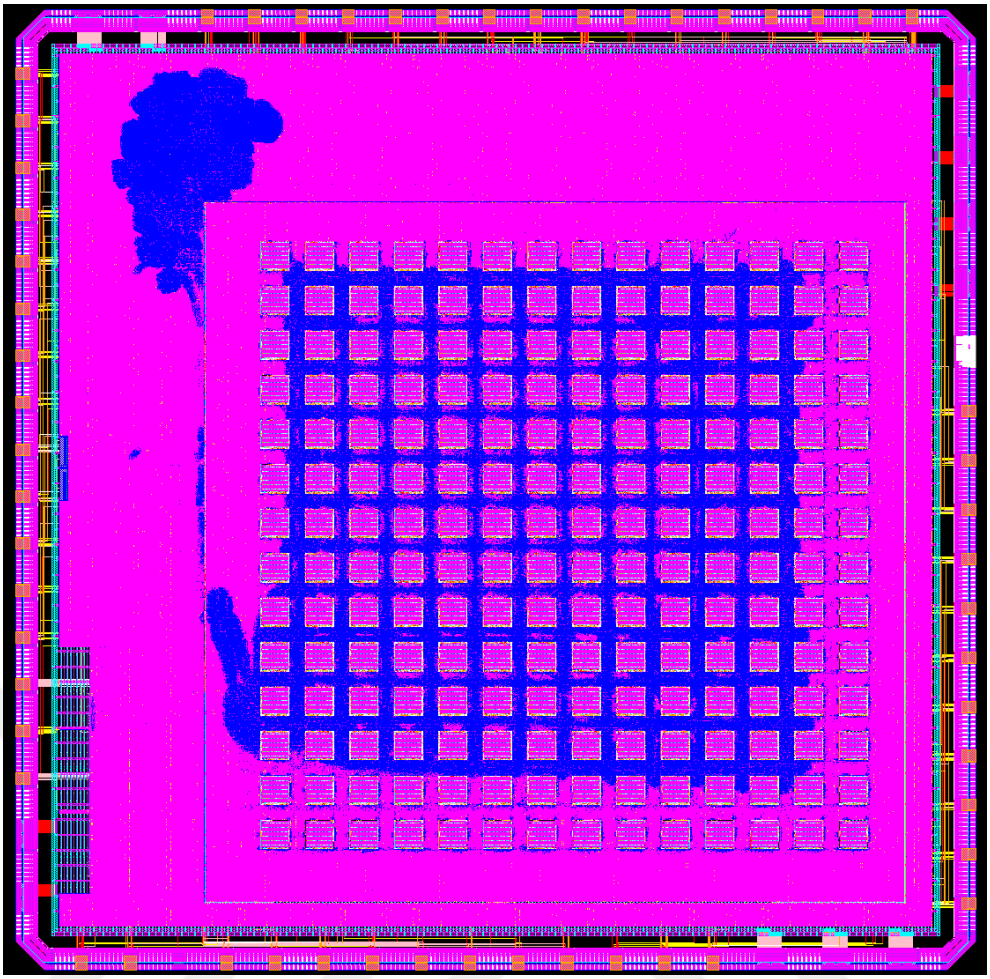


Figure 4.14 : The SoC layout.

5. CONCLUSION

As a result, FPGAs are important devices for their concurrency, reconfigurability, and high bandwidth to increase efficiency in recent years, but they are hard to integrate into systems due to their complicated power requirements. Embedded FPGA aims to solve these issues by instantiating a smaller fabric than discrete FPGAs to the SoC. There are two kinds of eFPGA IP, called hard and soft. Hard IPs are designed at the transistor level and cannot be ported to another technology or architecture. Soft IPs are found in RTL code form and they are flexible in architecture and technology. This appeals to the designers since their development time is shorter. Thus, an SoC was designed with soft-core eFPGA IP in this thesis. System-on-chip mainly consists of CVA6, an open-source 64-bit RISC-V processor and the eFPGA IP generated by OpenFPGA, open-source FPGA IP generator. The on-chip communication was done using the AXI4 protocol. The eFPGA IP is a homogenous FPGA fabric with 1960 LUTs, 1960 flip-flops and the AXI interface block. The interface block has six 64-bit registers and the processor can write or read through the registers. eFPGA also supports reconfigurability during runtime. The designed system was tested with four applications, the software and eFPGA-augmented implementations were compared. According to behavioural simulations made on Xcelium, the eFPGA-augmented implementations were an average of 3.83 times faster than the software. The physical design was also made with a commercial 16-nm technology node and Synopsys EDA tools. The clock frequency is 100 MHz and the die size is $4\,\mathrm{mm}^2$. With the methods we followed, a system-on-chip with eFPGA IP can be made and usable in various applications. However, some of the structures in commercial FPGAs, such as loop-free custom switch boxes, clock regions and chains between CLBs cannot be produced or not optimized as commercial FPGAs in OpenFPGA. So, adding these kinds of structures was avoided in this thesis. Also, the promised features of OpenFPGA except timing constraints were not supportive in physical design. The physical design settings

and methodology for how it is made is mostly decided without OpenFPGA's utilities. If the community and developers develop it further, the open-source tools used in this work can gain wider application areas.

REFERENCES

[1] *AI and Compute*, https://openai.com/index/ai-and-compute/.

[2] *The Ultimate Guide eFPGA - AnySilicon*, https://anysilicon.com/efpga/.

[3] **Kuon, I.**, **Tessier, R. and Rose, J.** (2008). FPGA Architecture: Survey and Challenges, *Foundations and Trends® in Electronic Design Automation*, *2*(2), 135–253.

[4] **Amano, H.** (2018). *Principles and Structures of FPGAs*, Springer, Singapore.

[5] *Graphics — Verilog-to-Routing 8.1.0-Dev Documentation*, https://docs.verilogtorouting.org/en/latest/vpr/graphics/.

[6] **Tang, X.**, **Giacomin, E.**, **Chauviere, B.**, **Alacchi, A. and Gaillardon, P.E.** (2020). OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs, *IEEE Micro*, *40*(4), 41–48.

[7] **Zaruba, F. and Benini, L.** (2019). The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-Nm FDSOI Technology, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *27*(11), 2629–2640.

[8] **Alzaq, H. and Üstündağ, B.** (2018). An Optimized Two-Level Discrete Wavelet Implementation Using Residue Number System, *EURASIP Journal on Advances in Signal Processing*, *2018*.

[9] **Ahmed, E. and Rose, J.** (2004). The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *12*(3), 288–298.

[10] **Boutros, A. and Betz, V.** (2021). FPGA Architecture: Principles and Progression, *IEEE Circuits and Systems Magazine*, *21*(2), 4–29.

[11] *Configuration — Project X-Ray 0.0-3807-G72e6371b Documentation*, https://f4pga.readthedocs.io/projects/prjxray/en/latest/architecture/configuration.html.

[12] **Wolf, C. and Glaser, J.** Yosys - A Free Verilog Synthesis Suite.

[13] **Murray, K.E.**, **Petelin, O.**, **Zhong, S.**, **Wang, J.M.**, **Eldafrawy, M.**, **Legault, J.P.**, **Sha, E.**, **Graham, A.G.**, **Wu, J.**, **Walker, M.J.P.**, **Zeng, H.**, **Patros, P.**, **Luu, J.**, **Kent, K.B. and Betz, V.** (2020). VTR 8: High-performance

CAD and Customizable FPGA Architecture Modelling, *ACM Trans. Reconfigurable Technol. Syst.*, *13*(2), 9:1–9:55.

[14] (2025). *Pulp-Platform/Axi*, pulp-platform.

[15] **Betz, V. and Rose, J.** (1998). How Much Logic Should Go in an FPGA Logic Block, *IEEE Design & Test of Computers*, *15*(1), 10–15.

[16] *Configuration Protocol — OpenFPGA 1.2.2942 Documentation*, https://openfpga.readthedocs.io/en/master/manual/arch_lang/config_protocol/.

[17] **kokke** (2025). *Kokke/Tiny-AES-c*.

[18] *Generator for CRC HDL Code*, https://bues.ch/cms/hacking/crcgen.html.

CURRICULUM VITAE

Name Surname: Yunus Emre ERYILMAZ

EDUCATION:

- **B.Sc.:** 2022, Istanbul Technical University, Faculty of Electric and Electronics Engineering, Department of Electronics and Communication Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2018 Intern at Penta Electronic Production
- 2019 Intern at TÜBİTAK BİLGEM
- 2020 Intern at PAVOTEK
- 2020-2022 Part-time Researcher at TÜBİTAK BİLGEM
- 2022- Researcher at TÜBİTAK BİLGEM

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Eryilmaz, Y. E.**, Yantir, H. E., & Yalçin, M. E. (2023). An Open-Source eFPGA-based SoC Design for Computation Acceleration. *ICECS 2023 - 2023 30th IEEE International Conference on Electronics, Circuits and Systems: Technosapiens for Saving Humanity (ICECS 2023 - 2023 30th IEEE International Conference on Electronics, Circuits and Systems: Technosapiens for Saving Humanity)*, December 4-7, 2023 Istanbul, Turkey. (Presentation Instance)