İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

5G UYUMLU QC-LDPC KODLAYICININ MODEL TABANLI TASARIM YÖNTEMİ İLE ETKİNLİK ANALİZİ

YÜKSEK LİSANS TEZİ

Hakan TAŞ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

ŞUBAT 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

5G UYUMLU QC-LDPC KODLAYICININ MODEL TABANLI TASARIM YÖNTEMİ İLE ETKİNLİK ANALİZİ

YÜKSEK LİSANS TEZİ

Hakan TAŞ
(504191212)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Sıddıka Berna Örs Yalçın

ŞUBAT 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

EFFECTIVENESS ANALYSIS OF 5G COMPATIBLE QC-LDPC ENCODER WITH MODEL BASED DESIGN APPROACH

M.Sc. THESIS

Hakan TAŞ
(504191212)

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Sıddıka Berna Örs Yalçın

FEBRUARY 2023

İTÜ, Lisansüstü Eğitim Enstitüsü'nün 504191212 numaralı Yüksek Lisans Öğrencisi Hakan TAŞ, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "5G UYUMLU QC-LDPC KODLAYICININ MODEL TABANLI TASARIM YÖNTEMİ İLE ETKİNLİK ANALİZİ" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Sıddıka Berna Örs Yalçın** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Sıddıka Berna ÖRS YALÇIN** İstanbul Teknik Üniversitesi	.....
	Dr. Öğr. Üye. Semiha TEDİK BAŞARAN İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Ali Emre PUSANE Boğaziçi Üniversitesi	.....

Teslim Tarihi : **30 Aralık 2022**
Savunma Tarihi : **13 Şubat 2023**

Sevgili Aileme...

ÖNSÖZ

Yüksek lisans çalışmalarım kapsamında danışmanlığımı üstelenen ve çalışmalarımda benimle tercübe, bilgi ve deneyimlerini paylaşan değerli hocam Prof. Dr. Sıddıka Berna Örs Yalçın'a sonsuz teşekkürlerimi sunarım.

Beni her zaman destekleyen ve zorlu koşullarda dahi her zaman yanımda olan, bu zorlukların üstesinden beraber geldiğimiz sevgili eşim Sıla Döngül Taş'a sonsuz sevgimi sunarım.

Çalışmalarımda benimle teknik konularda tecrübelerini paylaşıp, desteklerini esirgemeyen Ömer Faruk Aygen ve Oğuzhan Çik'a teşekkürlerimi sunarım.

Son olarak beni bu günlere getiren ve her konuda destek olan aileme saygı, sevgi ve minnetlerimle teşekkür ederim.

Şubat 2023

Hakan TAŞ
(Elektronik ve Haberleşme Mühendisi)

İÇİNDEKİLER

KISALTMALAR

FPGA	**:** Alanda Programlanabilen Kapı Dizileri
ASIC	**:** Uygulamaya Özel Tümleşik Devre
HDL	**:** Donanım Tanımlama Dili
IDE	**:** Tümleşik Geliştirme Ortamı
ILA	**:** Entegre Mantık Analizörü
PL	**:** Programlanabilen Lojik
PS	**:** İşlemci Sistemi
SoC	**:** System on Chip)
VHDL	**:** Çok Geniş Ölçekli Entegre Devreler
LDPC	**:** Düşük Yoğunluklu Eşlik Bitleri
QC-LDPC	**:** Yarı Döngüsel Düşük Yoğunluklu Eşlik Bitleri
DMA	**:** Direct Memory Access
AI	**:** Artificial Intelligence
CAM	**:** Content Addressable Memory
RAM	**:** Random Access Memory
ROM	**:** Read Only Memory
BRAM	**:** Block RAM
XOR	**:** Exclusive OR
SIMD	**:** Single Instrution Multiple Data
VLIW	**:** Very Long Instruction Word

SEMBOLLER

Z_c	**:** Yükseltgenme faktörü
K_b	**:** Kod blok boyutu
R	**:** Kod oranı
s	**:** Sistematik bitler
c	**:** Kod sözcğü
p_1	**:** İlk parite vektörü
p_2	**:** İkinci parite vektörü
m_b	**:** Temel matris satır boyutu
n_b	**:** Temel matris sütun boyutu

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

5G UYUMLU QC-LDPC KODLAYICININ MODEL TABANLI TASARIM YÖNTEMİ İLE ETKİNLİK ANALİZİ

ÖZET

Mobil haberleşme sistemleri insanların gündelik hayatının önemli bir parçasını oluşturmaktadır. Kullanılan pos cihazınlarından telefona, televizyon yayınlarından araç iletişimine kadar bir çok gündelik cihazlarda kullanılmaktadır. Bu cihazlar kullanılırken yapılan yayından dolayı cihazların bir birleri ile etkileşimi artmaktadır. Hava ortamında yapılan yayınlarda bozulmalar olabilmektedir. Karşılıklı yayın yapan iki cihaz arasında bir çok etken cisim veya cisimler bulunmaktadır. Bu cisim veya cisimler yayın kalitesine doğrudan etki edebilmektedir ve haberleşmede hatalar oluşabilmektedir. Shannon'nın kanal kapasite teorimine göre teorik kanal kapasitesi düşük oranlarda iletim yapıldıgı sürece düşük hata oranlarında veri iletimi mümkündür. Veri iletimi esnasında hata oranlarının düşürülmesi için kanal kodlama teknikleri kullanılmaktadır.

Mobil haberleşme sistemlerinde standart kuruluşları ortak algoritmaların kullanılması için standart belirlemektedirler. 3GPP (3rd Generation Partnership Project) mobil haberleşme sistemlerinde standartlar üzerine çalışmalar yapmaktadır. Bu çalışmalar kapsamında şirketlere davetler göndererek algoritma kararı verilmesi üzerine çalışmalarda bulunmaktadırlar. Üçüncü ve dördüncü nesil mobil haberleşme standartlarında kanal kodlama algoritması için turbo kodlarının kullanılmasına karar verilmiştir. LDPC (Low-density Parity Check) kodlama ilk olarak 1962 yılında Robert G. Gallager tarafından tasarlanmıştır. Ancak üçüncü ve dördüncü nesil mobil haberleşmede LDPC kodlarının kullanılması mümkün olmamıştır. Algoritmanın mevcut sistemlere uygulanılabilirliği açısından donanımsal zorlukları olduğundan 1990'lara kadar haberleşme sistemlerinde kullanımı çok mümkün olmamıştır. Son olarak 3GPP (3rd Generation Partnership Project) üyesi firmalar tarafından beşinci nesil mobil haberleşmede veri kanallarının kodlanmasında standart olarak kabul edilip kullanılmaya başlanmıştır. Kanal Kodlama yöntemi olarak 3GPP tarafından daha donanım dostu bir algoritmik yapısı bulunan QC-LDPC yöntemi seçilmiştir.

Tez çalışması kapsamında QC-LDPC kodlayıcı blok tasarımı model tabanlı olarak Xilinx Model Composer ve Xilinx AI Engine üzerinde yapılmıştır. Model Composer ile yapılan tasarım, donanım tanımlama dillerinden (HDL) Verilog'a çevirilerek IP üretilmiştir. Model tabanlı tasarımın sayesinde blok diagram tasarımından Verilog tasarımına kolaylıkla geçilebilmektedir. Bu çalışmalar kapsamında model tabanlı olarak tasarlanan QC-LDPC bloğun testlerinde MATLAB ortamı kullanılmıştır. MATLAB betikleri ile üretilen test verileri sisteme verilerek testler yapılmıştır. Simülasyon sonuçları ile donanım üzerinde çalışan sistemin sonuçlarının aynı olduğu gözlemlenmiştir. Simülasyon ortamında yapılan testler tasarım süreçlerini hızlandırmaktadır. Bu hızlanmanın nedeni tasarım ve testlerin aynı ortamda

yapılmasıdır. Model tabanlı yapılmayan tasarımlar, HDL dillerinin kullanıldığı geleneksel tasarımlara kıyasla daha hızlı tamamlanmaktadır. Bunun nedeni teorik çalışmaların ortam değiştirmeden, aynı ortamda bloklar şeklinde tasarlanabilmesidir. Kısalan tasarım süresi ve test süresi sayesinde nihai tasarım daha hızlı elde edilebilmektedir.

Bu çalışmada, 5G standartlarına uygun LDPC kodlayıcı bloğu tasarımı yapılması amaçlanmıştır. Bu amaç doğrultusunda model tabanlı LDPC kodlayıcı bloğu tasarımı oluşturulmuştur. 3GPP standartında 2 farklı LDPC tablosu olmasına rağmen tasarımda sadece birinci LDPC tablosuna uygun tasarım yapılmıştır. Gelen konfigürasyon bilgisinde bulunan yükseltgenme faktörü bilgisine göre LDPC tablosu kullanılmaktadır. Bu çalışamada hem model tabanlı FPGA tasarımının yapılması hem de yeni bir mimarisi olan Xilinx'in AI Engine ile 5G standartlarına uygun QC-LDPC kodlayıcı tasarımının ve FPGA uygulamasının yapılması amaçlanmıştır.

EFFECTIVENESS ANALYSIS OF 5G COMPATIBLE QC-LDPC ENCODER WITH MODEL BASED DESIGN APPROACH

SUMMARY

Mobile communication systems constitute an important part of people's daily life. It is used in many everyday devices from POS devices to telephone, from television broadcasts to vehicle communication. Due to the broadcasts made while using these devices, the interaction of the devices with each other increases. There may be many active object(s) between two mutually broadcasting devices. There may be corruption in broadcasts made in the air environment. These object(s) can directly affect the communication quality and errors may occur in communication. According to Shannon's channel capacity theorem, data transmission at low error rates is possible if the theoretical channel capacity is transmitted at low rates. Channel coding techniques are used to reduce error rates during data transmission.

Standardization organizations, set standard for the use of common algorithms in mobile communication systems. 3GPP (3rd Generation Partnership Project) works on standardization in mobile communication systems. They send the invitation for the working groups. In these group they discuss about the algorithms and their efficiency. Many companies present about their opinions and their works during their meetings. Channel coding is one of the topics which is discuss in the meetings. According to studies and the presentations about channel coding, third and fourth generation mobile communication system use the Turbo coding for the channel coding algorithm. In some companies suggest the LDPC (Low-Density Parity Check) coding. However, LDPC algorithm implementation in the hardware were hard at these days. LDPC coding was first designed by Robert G. Gallager in 1962. But these algorithm is not hardware-friendly. This means hardware implementation is very hard. Until 1990s, the algorithm was not able to be used in communication systems due to hardware difficulties in terms of applicability to existing system. 3GPP working groups discuss about the channel coding for the fifth-generation mobile communication system and they decide the QC-LDPC for the data channels. The biggest reason for them to take this decision is that the QC-LDPC (Quasi-Cyclic Low-Density Parity Check) algorithm has a hardware-friendly algorithm.

In this thesis, the QC-LDPC encoder block design was made on a model-based on Xilinx Model Composer. These design test and debugging are also made on the Xilinx Model Composer. After this test and debugging parts design is implementation on the Xilinx UltraScale+ RFSoC. Xilinx production ZCU670 development board is used for this implementation. Also, software design was made for the testing these design on the FPGA implementation. For these software design Petalinux was used. This software is used for DMA (Direct Memory Access). This design can be work with

software and hardware cooperate. Also, QC-LDPC encoder block design was made on Xilinx AI Engine. These design test and debugging are made on Xilinx model-based design platform, Model Composer. After this test and debugging parts design is implementation on the Xilinx Versal ACAP. Xilinx production VCK190 development board is used for this implementation.

In this duration, it was also learned to run the development environments required for the programming of FPGA chips produced by Xilinx company, high speed FPGA design, monitoring of signals at run time using System ILA, using Petalinux for the software design, design and debugging Xilinx SDK on Linux.

This thesis was written with six main chapters. Chapter 1 is the introduction of the thesis. In this chapter, purpose of the thesis is explained. Also, two main design environment is explained briefly.

Second chapter is preliminary. In this chapter LDPC was explained. It is also mentioned in the Tanner Diagram when describing LDPC. Also aim of the thesis, 5G NR (New Radio) channel codding LDPC and QC-LDPC are briefly explained. How to decide the base graph, lifting factor are explained in this chapter.

Third chapter is example work with Xilinx AI Engine blocks. In this chapter, the AI Engine was described. During the explanation of the AI Engine, matrix multiplication example is briefly explained. Aim of this example matrix multiplication is to understand the AI Engine's working principle and how to generate block is designed. Firstly, how to write AI Engine codes were described. After that,e AI Kernel and AI Graph were explained. How to connect these two structures were explained. After then, Vitis Tools Flow explained and how to use this flow. Extensible design was described. After then, how to design in Vivado dwas described. Test design in FPGA and monitor with ILA.

Fourth chapter is explained model-based design of the QC-LDPC encoder. In this chapter firstly, three design environment are compared according to resource consumption and design time. As a result of comparison, the design was made with the Model Composer. In the Xilinx Model Composer library, matrix digital signal processing blocks are not enough for the generation QC-LDPC encoder design. For this reason some blocks of the design sub blocks are generate from Xilinx Vitis HLS tools. In the Xilinx Model Composer design LDPC encoder design was made for base graph one which is define in 3GPP standard. Firstly, the Design accepts the control massage for the decision of which base graph will be used and lifting factor after the index information decision. When the base graph and lifting factor is determined, modulo operation will be used in the base graph. Every element in the base matrix is compared with the lifting factor value. If they are greater than the lifting factor value, subtract a lifting factor value from base matrix element. When the base matrix operation is done the system can accept the systematic bits from user. The system can accept a maximum of $22 \times Z_c$ bits for the base graph 1 and generate the $66 \times Z_c$ bits for the output. Actually, design generate the $68 \times Z_c$ which are systematic bits and parite bits. However, $2 \times Z_c$ bits remove in front of the stream. When the design is finished, testing and debugging the design is made in the Model Composer. When the testing finished IP generated and is used in the FPGA.

Fifth chapter is explained AI Engine design of the QC-LDPC encoder. In this chapter which operation were used in the design was explained. Model Composer design explained in this chapter. Also testing methodology is explained.

In the last chapter, based on the test results obtained, resource consumption, cycle time and design evaluation were made. Suggestions were made on how to make the design more effective. Information about the designs that can be derived from this study is given.

According to implementations, QC-LDPC encoder design can be made in model based on Xilinx Model Composer Design Suit. Also, can be designed in Xilinx AI Engine's.

1. GİRİŞ

Bu bölümde tez çalışmasında kullanılan önemli kavramlar ve bu çalışmanın motivasyonu açıklanmıştır. Çalışma hakkında derinlemesine açıklamalar yapılmadan çalışma hakkında temel bilgiler aktarılmıştır.

1.1 Amaç

Haberleşme alanında her geçen gün kullanıcı/cihaz sayısı artmaktadır. Bu nedenden dolayı haberleşmede kullanılan sistemlerin hızlı olması ve tüm kullanıcı/cihazlara da hızlı bir şekilde cevap vermesi gerekmektedir.

Günümüz haberleşme sistemlerinde teknolojinin sürekli değişmesi ile birlikte kullanılan standartlar da değişmekte ve güncellenmektedir. ASIC çiplerinin kullanılması güncellenen standartlar için yeni çiplerin yapılması gerekliliğini doğurmaktadır. Bu nedenle haberleşme sistemlerinin FPGA (Field Programmable Gate Array - Sahada Programlanabilen Kapı Dizisi) ile geliştirilmesinin en büyük avantajı sürekli güncellenebilmesidir. FPGA ile yapılan standart tasarımlarda blokların birbirine bağlılığı, tasarım ve test sürelerinin uzun olmasından dolayı FPGA tasarım süreleri de uzamaktadır. Model tabanlı yapılan tasarımlar bu sürenin kısalmasını ve ürünleşmenin hızlanmasını sağlamaktadır. FPGA firmalarının çıkarmış oldukları model tabanlı tasarım yöntemleri kendi cihazları ile daha uyumlu çalışmaktadır. Aynı zamanda bu uyumluluktan dolayı kaynaklarını daha etkin bir şekilde kullanması beklenmektedir.

Bu çalışmanın amacı 5G haberleşme sisteminde bulunan LDPC kodlayıcı bloğunun Xilinx Model Composer [17] ile model tabanlı tasarlanması ve Xilinx AI Engine ile tasarımının anlatılmasıdır. Ayrıca yapılan çalışmaların, Verilog dili ile yapılan tasarımlarla kaynak tüketimi ve performans açısından karşılaştırılmasının yapılmasıdır.

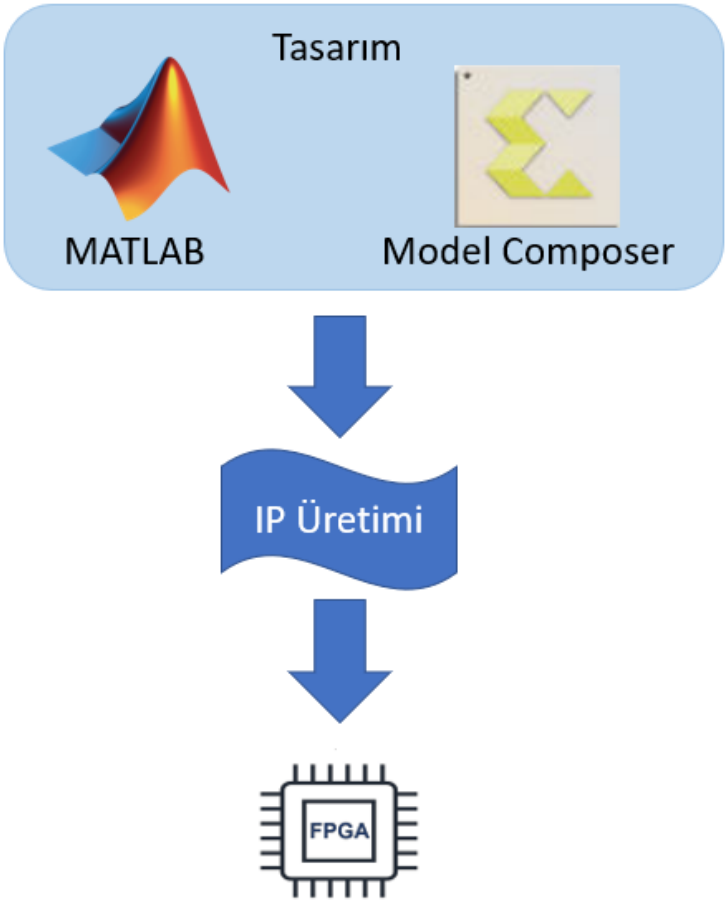
1.2 Model Tabanlı Tasarım

Donanım tanımlama dili (HDL), mantık devresinin tanımlanması ve gerçeklemesinde kullanılan bilgisayar dillerini ifade etmek için kullanılan bir kavramdır. Bu bilgisayar dilleri ile yapılması istenilen işin anlamlı bir biçimde metin tabanlı olarak ifade edilmesine olanak sağlamaktadır. HDL dilleri ile tasarımcı, tasarlamak istediği devrenin çalışma tasarımına ilişkin metin tabanlı bir model oluşturabilmektedir.

Model tabanlı tasarım, HDL dilleri kullanılmadan üretici firmanın oluşturmuş olduğu blokların ya da AND, OR, NOT gibi alt blokların kullanılmasıyla yapılan tasarımlardır. Bu tasarımlarda, tasarımcı kendine ait alt bloklarda oluşturabilmektedir. Model tabanlı tasarımların en büyük artılarından biri oluşturulmuş olan blokların alt blok olarak başka tasarımlarda ya da aynı tasarımının başka bölümlerinde kullanılmasıdır. Bu sayede tasarım süresinin kısalması sağlanmaktadır.

Xilinx Model Composer, Xilinx firmasının kendi FPGA'leri ile uyumlu olarak çalışan model tabanlı bir tasarım aracıdır [17]. Model Composer ile yapılan tasarımlarda, Xilinx firmasının geliştirmiş olduğu alt blokların kullanılması gerekmektedir. Bu blokların avantajı firmanın geliştirmiş olduğu FPGA mimarilerinin bilgilerini kullanarak daha optimize HDL kod üretilmesine olanak sağlamasıdır. Uygulamalara özel blokların oluşturulmasına da imkan sağlanmaktadır. Bu bloklar Xilinx firmasının HLS [15] dili kullanılarak da üretilmektedir. Bu sayede firmanın geliştirme yapmadığı blokların oluşturulmasına imkan sağlanmaktadır. Model Composer ile tasarımda alt seviye bloklardan başlanarak kapsamlı bir tasarım yapılabilmektedir. Şekil 1.1'de model tabanlı tasarım için MATLAB ve Model Composer kullanılarak IP üretilmesi ve FPGA'de kullanılmasının şeması gösterilmektedir.

MATLAB HDL Coder, MATLAB firmasının Simulink tasarım ortamının içinde HDL kod üretmek için özel oluşturduğu model tabanlı geliştirme ortamıdır [18]. Temel blokların ile genel tasarımlara imkan sağlamaktadır. Sistem içerisinde oluşturulmuş alt bloklar haricinde herhangi bir bloğun kullanılmasına imkan vermemektedir.



Şekil 1.1 : Model Composer ile IP üretimi.

Model tabanlı yapılan tasarımın testi için ayrıca bir ortam oluşturulmasına ihtiyaç duyulmamaktadır. Tasarım yapılan ortam aynı zamanda test ortamı olarak kullanılabilmetedir. MATLAB ile entegre çalışmasının avantajları kullanılarak tasarımının giriş ve çıkışlarının MATLAB ortamında analiz edilebilmesi mümkündür.

1.3 Xilinx AI Engine

Xilinx AI Engine, sinyal işlemeye yönelik VLIW mimarisinde SIMD vektör işlemcileri içermektedir [3]. AI enginelerin işlemcilerden farklı olarak PL (Programmable Logic - Programlanabilen Mantık Devresi) ile aralarında herhangi bir arayüz olmadan erişim sağlanabilmektedir. Her bir AI Engine bloğunun girişinde ve çıkışında veri saklamak için FIFO (Ping Pong Buffer) bulunmaktadır. Bu FIFO'ların olması sistem içerisinde verilerin bekletilebilmesine olanak sağlamaktadır.

Ayrıca AI Engine blokları arası yüksek hızlarda veri trafiğinin oluşabilmesi sistemin hızlı çalışmasına ve performansının artmasına yardımcı olmaktadır. AI enginelerin çalışma prensiblerinden dolayı işlemleri paralleleştirebilme imkanı mevcuttur. Bu paralleleştirme, VLIW ve SIMD mimarilerinin bir sonucu olarak ortaya çıkmaktadır. Verilen bir komutu paralleleştirerek birden fazla alt işlemlere ayırmaktadır. Bu sayede bir AI Engin'deki iş yükü azaltılarak sistemin hızlanması sağlamaktadır.

Düşük maliyetleri ve işlemci gücüyle FPGA tasarımlarının etkinliliğinin artırılmasından dolayı, kırmık üstü tasarım (SoC - System on Chip) uygulamalarda oldukça kullanılmaktadır. FPGA ile yapılan işlere ek olarak işlemci gerektiren tasarımlarda da SoC kullanılmaktadır.

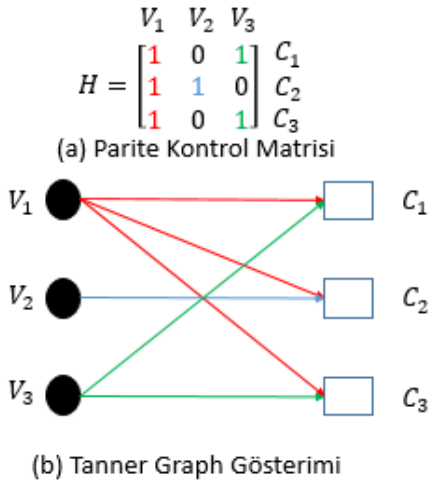
Bu çalışmada 5G sistemlerinde veri kanallarında kullanılan ileri yönde hata düzeltme algoritması olan LDPC'nin kodlayıcı bloğunun tasarımının, model tabanlı olarak Xilinx Model Composer ve Xilinx AI Engine blokları kullanılarak SoC tasarımı ve etkinlik analizi incelenmesi yapılacaktır.

2. 5G NR LDPC KODLAR VE LDPC KODLAYICILAR

2.1 LDPC

LDPC (Low Density Parity Check) ileri yönde uygulanan hata düzeltme kodlarından bir tanesidir. 5G NR (Yeni Radyo) sistemlerinde yüksek verim ihtiyacından dolayı 4G LTE'de kullanılan Turbo Kodları [9] yerine veri kanallarında LDPC kullanılması 3GPP standartlarında kararlaştırılmıştır [2]. QC-LDPC (Yarı Döngüsel Düşük Yoğunluklu Eşlik Bitleri) Richardson ve Urbanke'nin bulmuş olduğu, LDPC'nin özelleşmiş bir alt kümesidir [16]. QC-LDPC, LDPC ye göre daha az kaynak tükettiğinden dolayı sistemlerin içinde aktif olarak kullanılmasına imkan sağlamıştır.

LDPC iki ana parçanın birleşmesi ile oluşmaktadır. Bunlardan biri Eşlik Kontrol Matrisi (Parity Check Matrix - PCM), diğeri Tanner Grafiğidir.



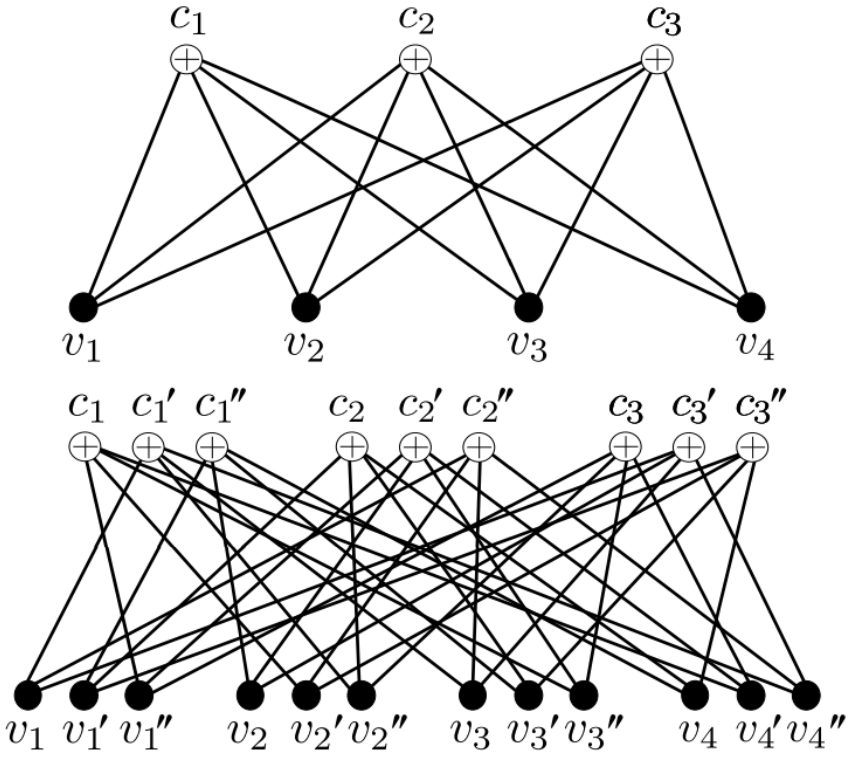
Şekil 2.1 : Parite kontrol matrisi ve Tanner Diagramı.

Şekil 2.1'de görüldüğü üzere PCM matrisi üzerinde bulunan her bir sutun bir düğüme, her bir satır bir kenara denk gelmektedir. Matris içerisinde 1'in anlamı düğüm ile kenar arasında bir bağlantı olduğunu 0 ise düğüm ile kenar arasında bir bağlantı olmadığını ifade etmektedir. Şekil 2.1'ye bakıldığında düğüm birin tüm kenarlar ile bağlantı olduğunu ancak düğüm ikinin sadece ikinci kenar ile bağlı olduğu görülmektedir.

LDPC'de de Tanner Graph şeklinde bir düğüm ve kenar bağlantıları mevcuttur [6]. Her bir bilgi satırına karşılık bir eşlik biti tekabül etmektedir.

2.2 5G'de LDPC Kodlayıcı

Bu çalışmada LDPC Kodlayıcı bloğu üzerinde çalışmalar yapılmaktadır. 5G NR sistemlerinde QC-LDPC kullanılmaktadır [2]. QC-LDPC protograph bir yapıdadır [1]. Protograph, Tanner Grafiği yapısı gibi düğüm ve kenar bağlantıları göstermektedir. Ancak bu yapıda lifting bilgisi dahil olmaktadır. Lifting bilgisi düğüm ve kenar bilgilerinin tekrarını göstermektedir. Şekil 2.2'de gösterilmekte olan üstteki şekil (3,4) boyutunda bir Tanner grafiği, alt tarafta bulunun ise lifting değeri 3 olan bir protographdır.

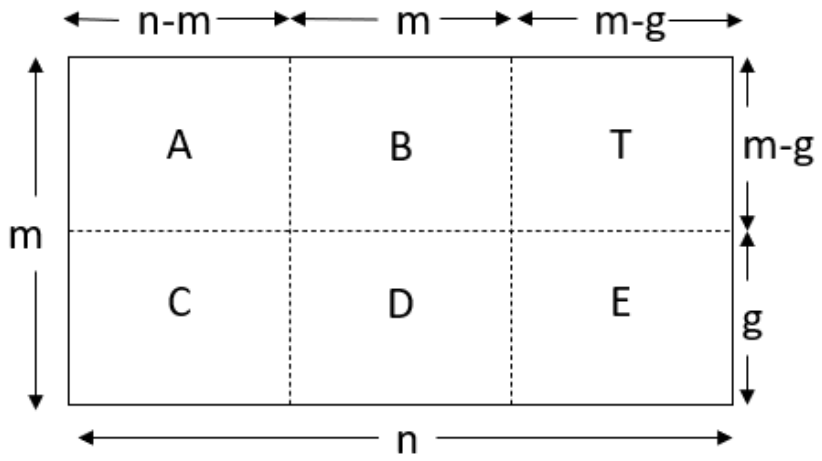


Şekil 2.2 : Boyutu (3,4) olan Tanner grafiği (üstte) ve boyutu (3,4) Lifting kat sayısı 3 olan Protograp (alttaki) [1].

Son yıllarda, LDPC üzerine yapılan araştırmalar, yapılandırılmış LDPC olarak adlandırlan QC-LDPC kodlarına odaklanmıştır [16] [19]. QC-LDPC'nin diğer metotlardan üstün olduğu noktalar; donanım gerçeklemeleri ve basit kaydırma register ile logik devrelerinin kullanılması. QC-LDPC kullanılarak düşük karmaşıklıkta

kodlayıcı bloğu gerçeklmesi parite kontrol matrisinin seyrekliği sayesinde yapılabilmektedir [19]. LDPC kodlayıcı bloğu için donanım karmaşıklığını iyileştirmek için çeşitli yaklaşımlar önerilmiştir [16] [20] [21] [22]. Bunlar arasında en geleneksel yaklaşımlardan biri sistematik kodlamadır. Sistematik kodlamada, Üretici matrisi, Gauss eliminasyon metotuyla parite kontrol matrisinin içerisinden üretiliyor. Bu yöntemin en büyük dezavantajlarından birisi kodlamaya girecek blokların boyutunun artmasıyla beraber depolama ek yükünün önemli bir ölçüde artmasıdır [19]. Richardson ve Urbanke'nin geliştirmiş oldukları QC-LDPC yaklaşımı klasik LDPC'ye kıyasla daha donanım dostu bir algoritmadır [16]. Richardson ve Urbanke'nin yaklaşımının temel amacı, parite kontrol matrisinin yaklaşık bir alt üçgen matrisine dönüştürülmesi ile yalnızca satır veya sütun permutasyonlarının kullanılarak oluşturmasını amaçlamıştır [16].

2.3 QC-LDPC Kodlayıcı Yapısı



Şekil 2.3 : Richardson ve Urbanke parite kontrol matrisi.

Şekil 2.3'de Richardson - Urbanke QC-LDPC kodlama algoritmasında [16] kullanılan parite kontrol matrisi gösterilmektedir.

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} \tag{2.1}$$

QC-LDPC kodlama algoritmasında *c codeword*'ü denklem **2.3**'de belirtildiği gibi hesaplanmaktadır [16]. Denklemde belirtilen *s* parçası sistematik bitlerden

oluşmaktadır. p_1 ve p_2 matrisleri ise parite bitlerinden oluşmaktadır [16]. Algoritmanın amacı

$$Hc^T = 0^T \tag{2.2}$$

denkleminin çözümünü bulmaktır.

Richardson ve Urbanke'nin tasarlamış olduğu algoritma beş aşamadan oluşmaktadır. İlk aşamada As^T hesaplanmaktadır. Sonrasında ise C matrisi ile sistematik bitlerin transpozunun çarpılması gerekmektedir (Cs^T). Daha sonraki adımda E matrisi, T matrisi ve A matrisinin çarpılması gerekmektedir $-ET^{-1}As^T$. Öncelikli çarpma işlemleri yapıldıktan sonra ilk parite matrisinin hesaplanması denklem **2.4**'de gösterilmektedir. Denklemlerde belirtildiği gibi p_1 sadece sistematik bitlerine bağımlıdır. p_2'den bağımsız bir şekilde hesaplanabilmektedir. İlk bulunan parite matrisi kullanılarak ikinci parite matrisi hesaplanması denklem **2.5**'de gösterilmektedir [14]. T matrisinin alt üçgen matris olduğu bilgisi mevcut olduğundan geri doğru yerine koyma algoritması ile p_2 elde edilebilir.

$$c = \begin{bmatrix} s & p_1 & p_2 \end{bmatrix} \tag{2.3}$$

$$p_1^T = \left[ET^{-1}As^T + Cs^T\right] \tag{2.4}$$

$$p_2^T = \left[-T^{-1}(As^T + Bp_1^T)\right] \tag{2.5}$$

2.4 5G QC-LDPC Kodlayıcı Yapısı

QC-LDPC, 5G (New Radio - Yeni Radyo) veri kanallarında 3GPP 38.212 [2]'da standart haline gelmiştir. Yapılan toplantılarda 3GPP, 2 tane hız uyumluluğu olan temel matris BGN1 ve BGN2'nin kullanılmasını kararlaştırmıştır [2]. 3GPP TS 38.212'de belirtildiği üzere, birinci temel matris (BGN1) büyük kod blok boyutlarında ($500 \leqslant K \leqslant 8448$) ve yüksek kod oranlarında ($1/3 \leqslant R \leqslant 8/9$) kullanılırken ikinci temel matris (BGN2) küçük kod blok boyutlarında ($40 \leqslant K \leqslant 2560$) ve düşük kod

oranlarında ($1/5 \leqslant R \leqslant 2/3$) kullanılmaktadır [2]. Kod oranı hesapları sisteme giren bit sayısı ile sistemden çıkan bit sayısı oranı ile gösterilmektedir. Ne kadar çok parite biti eklenirse kod oranı o kadar düşmektedir. Kod oranı hesabı Denklem **2.6**'de gösterilmektedir. Denklem **2.6**'de $K = Z_c \times k_b$, $N = (Z_c \times n_b) - 2$ ile gösterilmektedir.

$$R = K/N \tag{2.6}$$

5G NR (New Radio - Yeni Radyo) veri kanallarında kanal kodlaması olarak QC-LDPC kodları kullanılmadır. 3GPP 38.212 standartında [2] belirtildiği gibi ilk olarak sistemde kullanılacak temel parametrelerin hesaplanması gerekmektedir.

$$0 \leqslant P_{ij} \leqslant Z \tag{2.7}$$

Sistemde dolanım matrisinin boyutunu belirtmek için Z değeri, kaydırma değeri olarakta P_{ij} kullanılacak. P_{ij} değeri denklem **2.7**'de 0 ile Z arasında olması gerektiği belirtilmektedir. Boyutu $Z \times Z$ olan dolanım matrisi aslında birim matristir. Kaydırma değeri baz matrisin (i, j) elemanıdır ve birim matrisin dairesel olarak ne kadar döneceğini belirtir. "-1" değeri için boyutu $Z \times Z$ olan 0 matrisi yerleştirilir. Denklem **2.8**'de P_{ij} değeri 2 olduğu zamanki permutasyon matrisi görülebilmektedir.

$$Q(2) = \begin{bmatrix} 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{2.8}$$

QC-LDPC kodlayıcı bloğunda parite control matrisi oluşturulurken dolanım matrisinden yararlanılmaktadır [16]. Denklem **2.9**'de parite kontrol matrisinin nasıl oluşturulacağı gösterilmektedir. m_b ve n_b değeri positif tam sayılardır. Parite kontrol matrisinde gösterilen her Q_{ij} elemanı $Z \times Z$ boyutundadır. H matrisi

Z_c	2	3	5	7	9	11	13	15
0	2	3	5	7	9	11	13	15
1	2	4	10	14	18	22	26	30
2	8	12	20	28	36	44	52	60
3	16	24	40	56	72	88	104	120
4	32	48	80	112	144	176	208	240
5	64	96	160	224	288	352		
6	128	192	320					
7	256	383						

$Z_c = a * (2^j)$

Şekil 2.4 : Z_c hesaplanması.

$$H = \begin{bmatrix} Q(P_{11}) & Q(P_{12}) & \cdots & Q(P_{1n_b}) \\ Q(P_{21}) & Q(P_{22}) & \cdots & Q(P_{2n_b}) \\ \vdots & \vdots & \ddots & \vdots \\ Q(P_{m_b1}) & Q(P_{m_b2}) & \cdots & Q(P_{m_bn_b}) \end{bmatrix} \quad (\mathbf{2.9})$$

QC-LDPC 5G veri kanallarında standard haline gelirken yapılan toplantılarda 3GPP, 2 tane hız uyumluluğu olan temel matrisler BGN1 ve BGN2'nin kullanılmasını kararlaştırmıştır [2]. 3GPP TS 38.212'de belirtildiği üzere, birinci temel matris (BGN1) büyük kod blok boyutlarında ($500 \leqslant K \leqslant 8448$) ve yüksek kod oranlarında ($1/3 \leqslant R \leqslant 8/9$) kullanılırken ikinci temel matris (BGN2) küçük kod blok boyutlarında ($40 \leqslant K \leqslant 2560$) ve düşük kod oranlarında ($1/5 \leqslant R \leqslant 2/3$) kullanılmaktadır [2].

$$K_b * Z_c \geq K \quad (\mathbf{2.10})$$

H parite check matrisinin oluşturulması için standartta iki tane temel parite matrisi belirtilmiştir. Kontrol bloğundan gelen bilgilere göre BGN1 ve BGN2'den birinin seçilimi yapılmaktadır. 3GPP TS 38.212 [2] standardına göre kod bloğu veri uzunluğu 3824'den küçük veya eşitse ve kod oranı 0.67 ise BGN1 seçilmesi gerekiyor, bu sağlanmıyor ise kod bloğu veri uzunluğu 292'den küçük veya eşitse BGN2 seçilmesi gerekiyor. Temel matris belirlenmesi yapıldıktan sonra K_b değerinin hesaplanması işlemi yapılmaktadır. BGN1 için K_b değeri 22 alınıyor. BGN2 için K_b değeri K değerine bağlı olarak 6,8,9 yada 10 olabiliyor. K_b değeri hesaplandıktan sonra K ve K_b kullanılarak Z_c değeri Denklem **2.10**'de gösterilidği şekilde hesaplanmaktadır. Z_c değerleri Şekil 2.4'de gösterilmektedir. Standartta belirtilen dizin hesaplanmasını

Set index (i_{LS})	Set of lifting sizes (Z)
0	{2, 4, 8, 16, 32, 64, 128, 256}
1	{3, 6, 12, 24, 48, 96, 192, 384}
2	{5, 10, 20, 40, 80, 160, 320}
3	{7, 14, 28, 56, 112, 224}
4	{9, 18, 36, 72, 144, 288}
5	{11, 22, 44, 88, 176, 352}
6	{13, 26, 52, 104, 208}
7	{15, 30, 60, 120, 240}

Şekil 2.5 : 3GPP yükseltgenme faktörü hesap tablosu [2].

bulunan Z_c değerine göre yapılmaktadır. Şekil 2.5'de 3GPP standarında yükseltgenme boyutlarının tablosu mevcuttur. Bu tabloda hesaplanan Z_c değeri hangi satıra düştüğünün bulunması gerekmektedir. Bulunan I_{LS} değerine göre standartta belirtilen temel matris seçilmesi yapılmaktadır. Şekil 2.5'de standarttan alınan bir parça gösterilmektedir. Bu parçada da görüldüğü gibi her temel matris için 8 tane sütün mevcuttur. Bu sütünlardan hangisinin seçileceği bilgisi I_{LS} ile elde edilmektedir. Temel matris seçim işlemi tamamlandıktan sonra Denklem **2.11** kullanılarak şekil 2.3'deki temel matris elde edilmektedir. Matris olarak hesaplanan veriler vektör formatına çevirilerek çıkış dataları elde edilmektedir.

$$P_{ij} = mod(V_{ij}, Z_c) \tag{2.11}$$

Verinin başında bulunan $2*Z_c$ boyutundaki sistematik bitler atılmaktadır [8]. Bant genişliğinden daha fazla yararlanmak ve oran uyumlandırmada istenilen veri boyutlarına uygun hale getirmek için $2*Z_c$'lik kısımın atılmasına imkan sağlanılmaktadır. Bu şekilde sistematik bilgi bitleri kanal gürültüsünden etkilenmesi azaltılmış bir şekilde kanala verilebiliyor.

2.5 Gerçeklemesi Yapılan QC-LDPC Kodlayıcı Yapısı

Richardson - Urbanke tasarımından sonra LDPC kodlarının yarı döngüsel olarak tasarlanmasına ağırlık verilmiştir. Bu şekilde LDPC tasarımları daha donanım dostu olmuşlardır. Ayrıca kaynak tüketimi azalmıştır. Kaynak tüketimini azaltmak

için yapılan çalışmalardan bir tanesi de Nguyen, Tan ve Lee tarafından yapılmıştır [19]. Yapılan çalışmadaki ana amaç temel matris üzerindeki her elemanın tek tek yapılmasından dolayı kaybedilen kaynakaları azaltmaktır. Kaynak tüketimi azaltmak için satır olarak paralel işlemlerin yapılması amaç edinilmiştir. Önerilen hesaplamada C matirsi iki parçaya bölünmüştür. İlk parçanın (C_1) sütun boyutu A'nın sütün boyutu ile aynı olarak k_b boyutundadır. İlkinci parça sütun boyutuda B'nin sütun boyutu ile aynı olarak 4'dür. Denklem **2.2**'i Nguyen'nın geliştirmiş olduğu algoritmaya göre açtığımızda Denklem **2.12**'i matris formatında gösterilmektedir.

$$\begin{bmatrix} A & B & 0 \\ C_1 & C_2 & I \end{bmatrix} \begin{bmatrix} s \\ p_a \\ p_c \end{bmatrix}^T = 0^T \tag{2.12}$$

Matris çarpımlarını denklem formatında açtığımızda iki ana denklem elde edilmektedir. İki denklemi Denklem **2.13** ve Denklem **2.14**'de gösterilmektedir.

$$\left[As^T + Bp_a^T + 0p_d^T\right] = 0^T \tag{2.13}$$

$$\left[C_1 s^T + C_2 p_a^T I p_d^T\right] = 0^T \tag{2.14}$$

Denklem **2.13** detaylı bir şekilde açılıp incelendiğinde p_a değeri Richardson-Urbanke metodunda olduğu gibi p_{a_1}, p_{a_2}, p_{a_3} ve p_{a_4} vektörlerinden oluşmaktadır [19]. Denklem **2.15**'de p_a vektörlerinin hesaplanmasında kullanılan λ gösterilmektedir. Şekil 4.14'de gösterilen D matrisi çift asal eksenli olmasından dolayı λ değeri dört satır içinde hesaplanıp toplandığında p_{a_1} vektörü hesaplanmaktadır. Denklem **2.16**'de p_a vektörünün hesaplanması gösterilmektedir. Çift asal eksen özelliğinden dolayı iki kere aynı değerin toplanması sıfır olduğundan tüm p_a'lar birbirlerini sadeleştirmektedirler. H matrisinin çift asal eksenli olması kullanılarak p_{a_1} hesaplandıktan sonra bir önceki değerden mevcut değerlin hesaplanabilmesi mümkündür.

$$\lambda_i = \sum_{j=0}^{k_b} a_{i,j} s_j, \text{ for i = 1,2,3,4} \tag{2.15}$$

Bu denklemlerde çarpım ve toplama işlemleri mevcuttur. Bu iki işlem ile p_a matris hesabının yapılabillmesi mümkündür. Nguyen'ın belirtmiş olduğu algoritmada p_{a_1}'in hesaplanması dört saat çevriminde hesaplanmaktadır. Geri kalan üç vektörün hesaplanması toplam iki çevrim süresinde hesaplanmaktadır. Vektör p_{a_3} ile p_{a_4} arasında ilişki olduğundan önce dördün hesaplanması gerekmektedir. Bu bağımlılıktan dolayı hesaplamalar iki çevrim süresinde yapılmaktadır.

$$p_{a_1} = \sum_{j=1}^{4} \lambda_i, \tag{2.16}$$

$$p_{a_2} = \lambda_1 + p_{a_1}^{(1)} \tag{2.17}$$

$$p_{a_3} = \lambda_3 + p_{a_4} \tag{2.18}$$

$$p_{a_4} = \lambda_4 + p_{a_1}^{(1)} \tag{2.19}$$

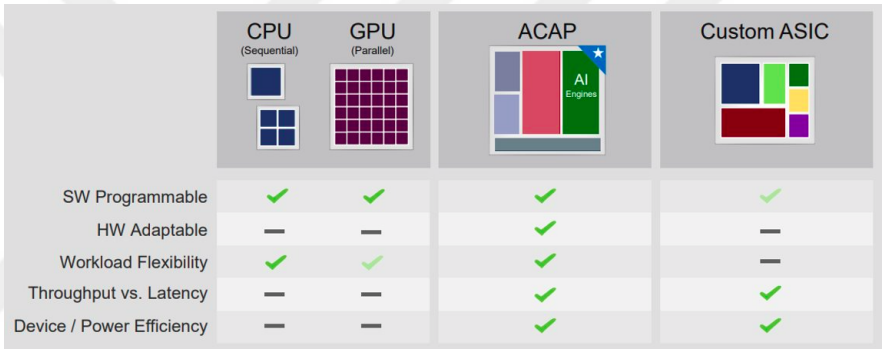
Burada p_a değerinin hesaplanmasından sonra p_c vektörlerinin değerlerinin hesaplanması işlemlerine geçilmektedir. Denklem **2.20**'da p_c vektörünün parametrik olarak hesabı gösterilmektedir. Bu parametreler içerisinden g değeri B matrisinin boyutunu, m_b kullanılan temel matrisin satır sayısını, k_b kod blok sayısını aynı zamanda A matrisinin sütun sayısını belirtmektedir. Sistem içerisinde p_c vektör elemanlarıda Denkelem **2.20**'da gösterildiği gibi C_1 ve C_2 matrisleri, S ve p_a vektörleri kullanılarak hesaplanmaktadır. Nguyen p_c vektörlerinin her birinin hesaplanmasının bir çevrim süresinde hesaplanabileceğini belirtmiştir [19].

$$p_{c_{m_b-g}} = \sum_{j=1}^{k_b} a_{c_{m_b-g},j} s_j + \sum_{j=1}^{g} a_{c_{m_b-g},k_b+j} p_{a_j} \tag{2.20}$$

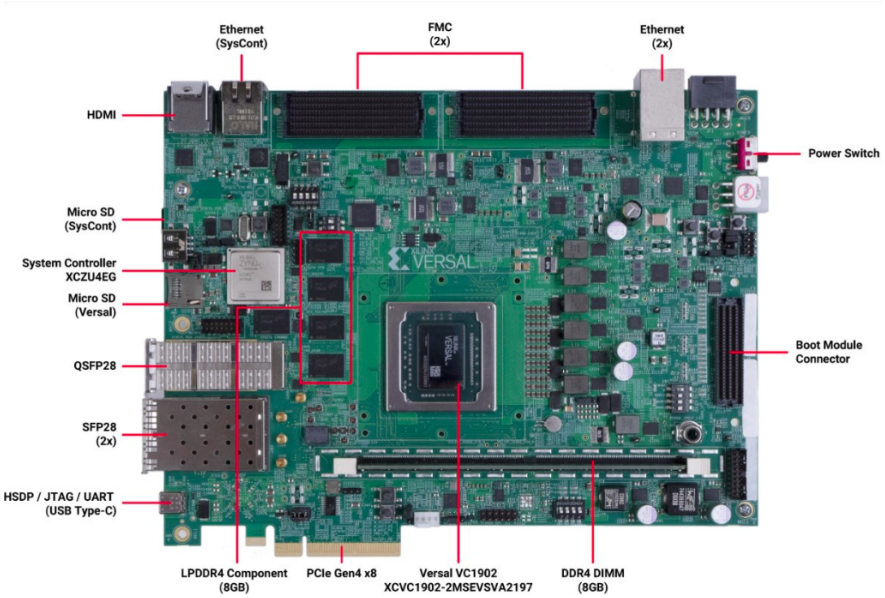
Sistemde p_a ve p_c vektörlerinin hesaplanması tamamlandıktan sonra sisteme giriş olarak verilen sistematik bitlerin ucuna parite bitleri eklenerek sistem çıkışına verilmektedir. Denkelem **2.12**'da belirtildiği gibi s, p_a ve p_c vektörleri sistem çıkışına verilmektedir.

3. XİLİNX AI ENGİNE VE ÖRNEK ÇALIŞMA

Bu bölümde Xilinx AI Engine bloklarının çalışması anlatılmıştır. Bu gerçeklemede Xilinx VCK190 geliştirme kartı kullanılmıştır [3]. Xilinx VCK190 geliştime kartı Şekil 3.2'de gösterilmektedir. VCK190 kartı FPGA matığının yanında Xilinx Artificial Intelligence (AI) Engine bloklarını da içermektedir. Tüm uygulamalar için Xilinx Vivado Design Suit ve Model Composer programları kullanılmıştır. Yapılan çalışmalar Vivado Design Suit ile derlenip VCK190 kartına yüklenmiştir. AI Engin kullanılmasının motivasyonu Figue 3.1'da gösterilmektedir.



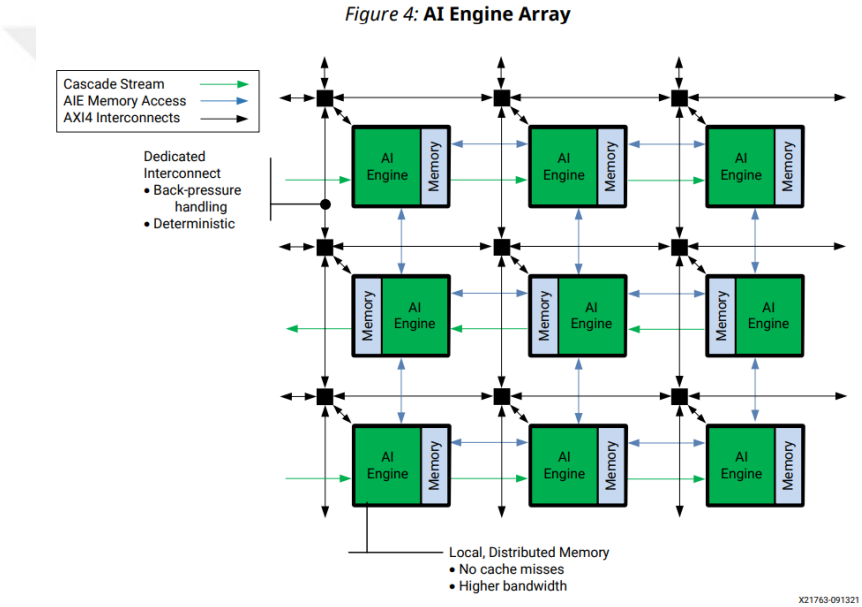
Şekil 3.1 : AI Engin kullanılması motivasyonu.



Şekil 3.2 : VCK190 FPGA kartı [3].

3.1 AI Engine Bloğu

AI Engine blokları programlanabilir mantığın (PL) dışında, silikona entegre edilmiş ayrı bir yapıdır. AI Engine blokları SIMD (Single Instruction Multiple Data) mimarisine sahip VLIW (Very Long Instruction Word) komutları ile çalışmaları sayesinde çok yüksek hızlara çıkabilmektedirler. AI Engine bloklarının içerisinde giriş çıkış için 32KB'lik tampon (buffer) bulunmaktadır. AI Engin blokları PL bölgesine doğrudan erişim yetkisine sahip oldukları için FPGA tasarımlarında herhangi bir ek bağlantı arayüzüne ihtiyaç duyulmamaktadır.



Şekil 3.3 : AI Engine bloğunun bağlantı şeması [3].

AI Engine bloklarının geleneksel PS(Processing System) ayıran kısım işlemlerin parallel bir şekilde yapılmasına imkan sağlaması ve herhangi ara bir protokol olmadan erişim sağlanabilmesidir. VLIW yapısı komut yapısında paralleliğe imkan sağlarken SIMD mimarisi veri yapısında paralleliğe imkan sağlıyor. İki yapının birleşiminden oluşan bir mimari hem komut hemde veri bakımından paralellik sağlamaktadır.

3.2 Model Tabanlı AI Engine ile Matris Çarpıcı Blok Tasarımı

AI Engin bloğu ile model tabanlı matris çarpıcı bloğu tasarımı yapmak için AI Kernel'inin programının yazılması gerekiyor. AI Kernel'i içerisinde bloğun nasıl çalışacağı C/C++ dili ile tanımlanması gerekmektedir.

```
#include <adf.h>
#include "system_settings.h"

// 16x8  x  8x8
// Can be extended to (2L)x(8M)  (8M)x(8N)
void matmult_float(input_window_float* __restrict matA, input_window_float* __restrict matB, output_window_float* __restrict matC)
  {
    #define FPMUL(Start1,Start2)\
    buf_matB = window_readincr_v8(matB); \
    acc1 = fpmul(buf_matA,Start1,0x00000000,buf_matB,0,0x76543210);\
    acc2 = fpmul(buf_matA,Start2,0x00000000,buf_matB,0,0x76543210)

    #define FPMAC(Start1,Start2)\
    buf_matB = window_readincr_v8(matB);\
    acc1 = fpmac(acc1,buf_matA,Start1,0x00000000,buf_matB,0,0x76543210);\
    acc2 = fpmac(acc2,buf_matA,Start2,0x00000000,buf_matB,0,0x76543210)

    v16float buf_matA = window_readincr_v16(matA);      // holds the first 16 elements of matA
    v8float buf_matB = undef_v8float();
    v8float acc1 = undef_v8float();   // 8 accumulation values on even rows
    v8float acc2 = undef_v8float();   // 8 accumulation values on odd rows

    for (unsigned int i=0; i<F_Ra/2; i++)          // This loop computes one row of C eachtime
    chess_prepare_for_pipelining
    chess_loop_range(8,)
      {
      FPMUL(0,8);          // fills empty acc with results of first element of matA's current row multiplication by all elements on row of matB
      FPMAC(1,9);          // accumulate matA's current row's next element multiplication by all elements of current row of matB
      FPMAC(2,10);
      FPMAC(3,11);
      FPMAC(4,12);
      FPMAC(5,13);
      FPMAC(6,14);
      FPMAC(7,15);

      buf_matA = window_readincr_v16(matA);      // reads the next 2 rows of A
      window_decr_v8(matB,8);                    // reset matB window back to first row
      window_writeincr(matC,acc1);               // Writes 1 row of C
      window_writeincr(matC,acc2);               // Writes the next row of C
      }
  }
```

Şekil 3.4 : matmult Kernel tanımı.

Örnek tasarımda 16x8 ile 8x8 boyutunda 2 tane matrisin çarpımı yapılacak bir tasarım oluşturuldu. Şekil 3.4'da çarpıcı bloğunun tanımı görülmektedir. Bu tanımlamada AI Engine bloklarının paralelliğinden yararlanılmak için 2 tane makro yazılmıştır. Bunlar;

- FPMUL
- FPMAC

FPMUL ve FPMAC makroları, iki tane arguman almaktadır. Bu iki arguman A matrisinin hangi elemanından itibaren çarpmaya başlanacağı bilgisi ile B matrisinin hangi elemanından itibaren çarpmaya başlanacağı bilgisidir. *window readincr v8* verilere erişmek için kullanılan C API'sidir. Sonunda belirtilen "vx" kaç eleman erişmek istenildiğini belirtmek için kullanılıyor. 8 elemana erişim sağlanamak isteniyorsa "v8", 16 elemana erişim sağlanmak isteniyorsa "v16" kullanılmaktadır.

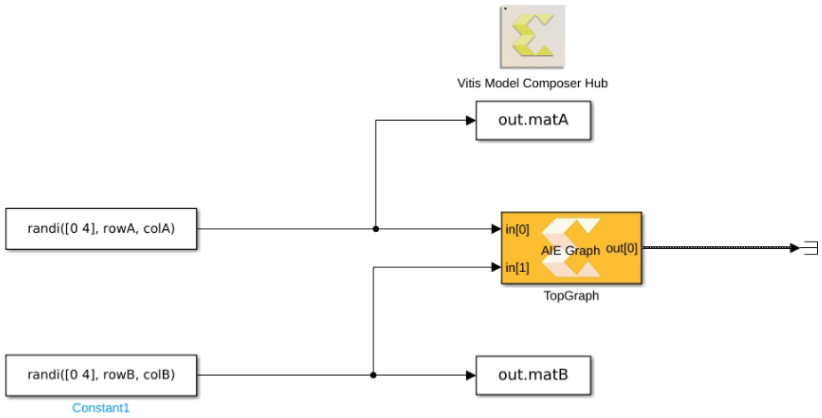
matmul kernel bloğu ilk olarak A matrisinin ilk 16 elemanını okumaktadır. Daha sonra boş bir B matrisi oluşturuyor ve 2 tane veri saklamak için yer açıyor. For döngüsü içerisinde paralleliğin olması için chess komutları kullanılıyor. For döngüsü içerisinde ilk olarak FPMUL makrosu çağırılarak A matrisinin ilk 16 elemanı ile B matrisinin ilk 8 elemanı ters sıra ile çarpılıyor. Bu işlemlerin paralel olması için 2 defaf aynı işlem farklı değerler ile çağırılıyor. fpmul makrosun son argumanı B matrisindeki 8 elemanın hangi sıra ile çarpılacağını göstermektedir.

FPMUL makrosu ile ilk çarpım oluştuktan sonra FPMAC makrosu ile oluşan çarpım sonucu kullanılarak kümülatif bir çarpım yapılmaktadır. FPMAC işleri bittikten sonra A matrisinden yeni 16 tane eleman ve B matrisinden 8 yeni eleman alıyor. Oluşan 16 elemanda dışarı atılarak yeni hesaplamarak devam edilmektedir. Bu işlemler tüm satırlara tekrarlanmaktadır.

```
#include <adf.h>
#include "kernels.h"
#include "system_settings.h"

using namespace adf;

template<int COL,int ROW>
class MatMultFloatGraph : public graph
{
private:
  // kernel declaration
  kernel k;

public:
  // port declaration for graph
  port<input> ina,inb;
  port<output> outc;

  MatMultFloatGraph()
  {
    k = kernel::create(matmult_float);  // kernel constructor from adf library

    // connections of constructed kernel.
    connect< window<NSAMPLES_WINDOW_F_A*NBYTES_FLOAT> > float_ina(ina, k.in[0]);
    connect< window<NSAMPLES_WINDOW_F_B*NBYTES_FLOAT> > float_inb(inb, k.in[1]);
    connect< window<NSAMPLES_WINDOW_F_C*NBYTES_FLOAT> > float_outc(k.out[0], outc);


    source(k) = "matmult_float.cpp";        // source file(cpp) for kernel created
    location<kernel>(k) = tile(COL,ROW);    // location configuration -- this is used to explicitly force a location
    runtime<ratio>(k) = 0.6;                // runtime configuration -- this is used for schedule multiple kernels in one tile
  }
};
```

Şekil 3.5 : Graph tanımı.

AI Engine kernel'in tanımı yaptıktan sonra kernelin çalıştırılacağı Graph tanımına geçilmesi gerekmektedir. Graph aslında Nesneye Yönelik Programlama'da (Object Oriented Programming - OOP) kullanılan Nesneye karşılık gelmektedir. Graph oluşturulurken AI kernel çağrılıyor. Oluşturulan Graph'ın giriş çıkış portları kernel ile bağlantısı yapılıyor. Daha sonra kernelin kaynağı belirtiliyor. Location ile fiziksel olarak hangi kernelin kullanılacağı belirtiliyor. runtime ratio ile seçilen kernelin yüzde kaçının bu graph için kullanılacağı seçiliyor. Eğer ki tek bir graph var ise sistemde seçilen kernelin yüzde yüzü bu graph için kullanılır.



Şekil 3.6 : Model tabanlı AI Engine ile matris çarpıcı blok tasarımı.

Model Composer üzerinde model tabanlı tasarımı Şekil 3.6'da gösterilmektedir. Tasarım basit olarak 2 tane rastgele matris üreten bloktan ve bir tane matris çarpıcı bloktan oluşmaktadır. Similasyon sonuçları Şekil 3.8'de gösterilmektedir.

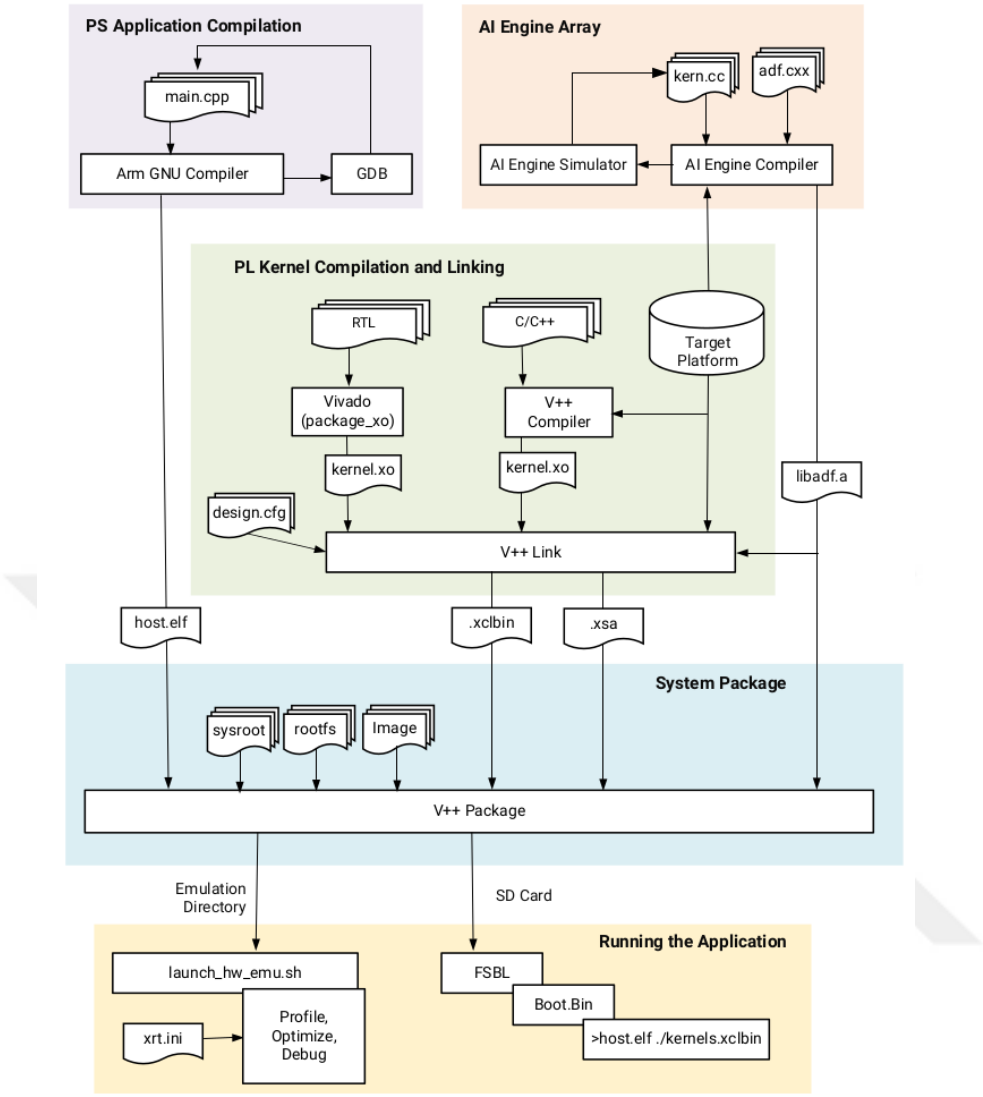
Similasyondan sonra kart üzerinde testler için Vivado ortamında AI Engine için proje açılması gerekiyor. AI Engine için geleneksel yöntemler dışında Vitis Tools Flow'un kullanılması gerektiği dökümanlarda belirtilmiştir [4]. Şekil 3.7'de Vitis Tools Flow gösterilmektedir. Bu akışa göre sırasıyla;

- AI Engine yükü oluşturulmalı
- PL yükü oluşturulmalı
- V++ ile AI Engine yükü ile PL yükü link edilmeli
- V++ Package ile rootfs,sysroot,image ve xsa birleştirilerek karta atılabilir yük çıkarılması

işlemlerinin takip edilmesi gerekmektedir. AI Engine yükü Model Composer üzerinde test edilen Graph ve AI Kernel tasarımları. Vivado tarafında tasarımda proje oluşturulduktan sonra Şekil 3.9'de gösterildiği gibi extensible projectin açılması gerekmektedir.

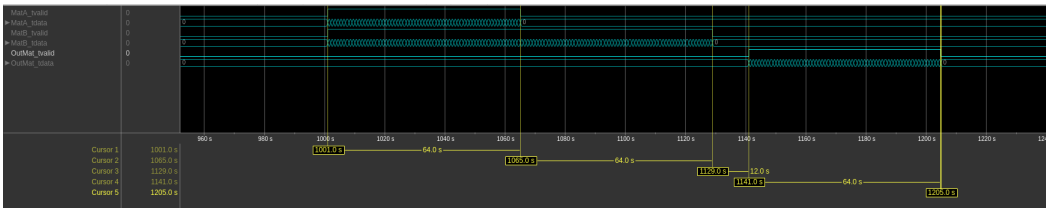
Vivado tasarımında blok tasarımına Xilinx Versal Control,Interface & Processing System (CIPS) bloğunu yerleştiriyoruz. Bu blok PS ve PL arasında haberleşmeyi sağlamaktadır. Network on Chıp (NoC) RAM ve AI Engine bloklarına PS üzerinden

Figure 52: **Vitis Tools Flow**

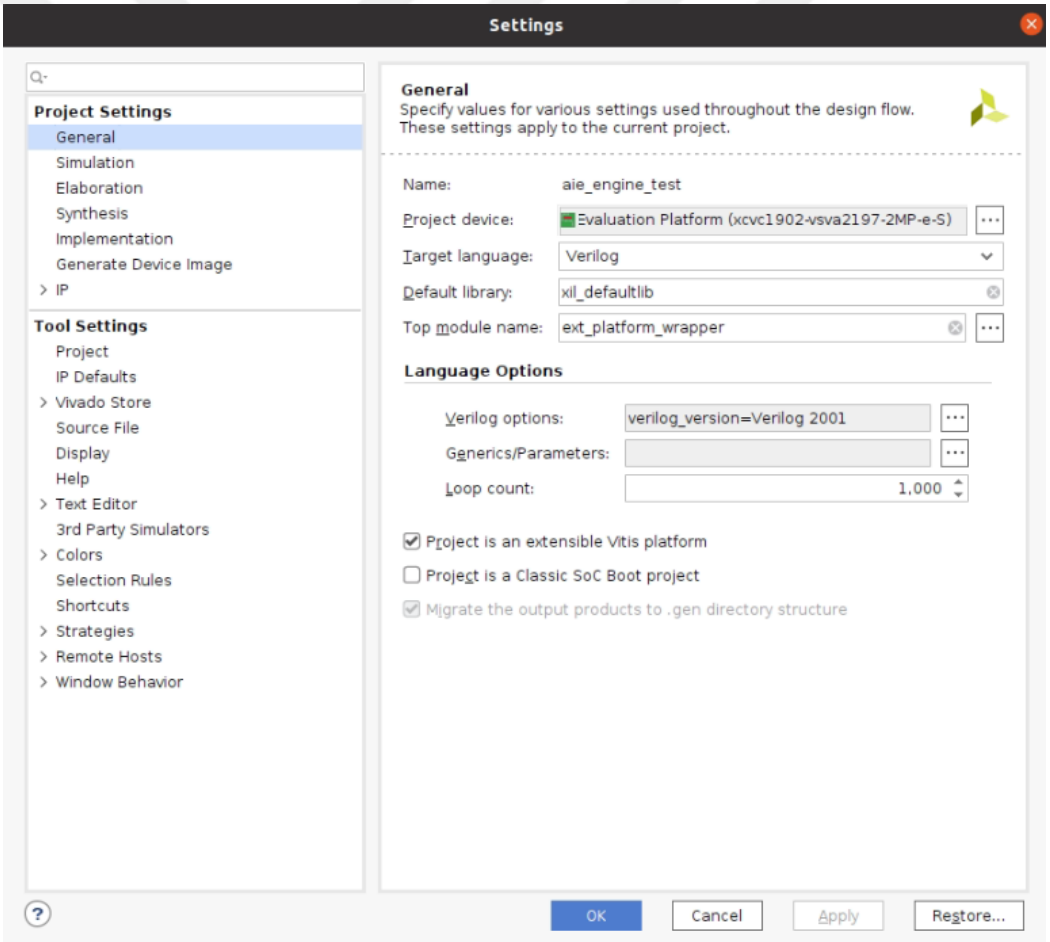


Şekil 3.7 : Vitis Tools Flow [4].

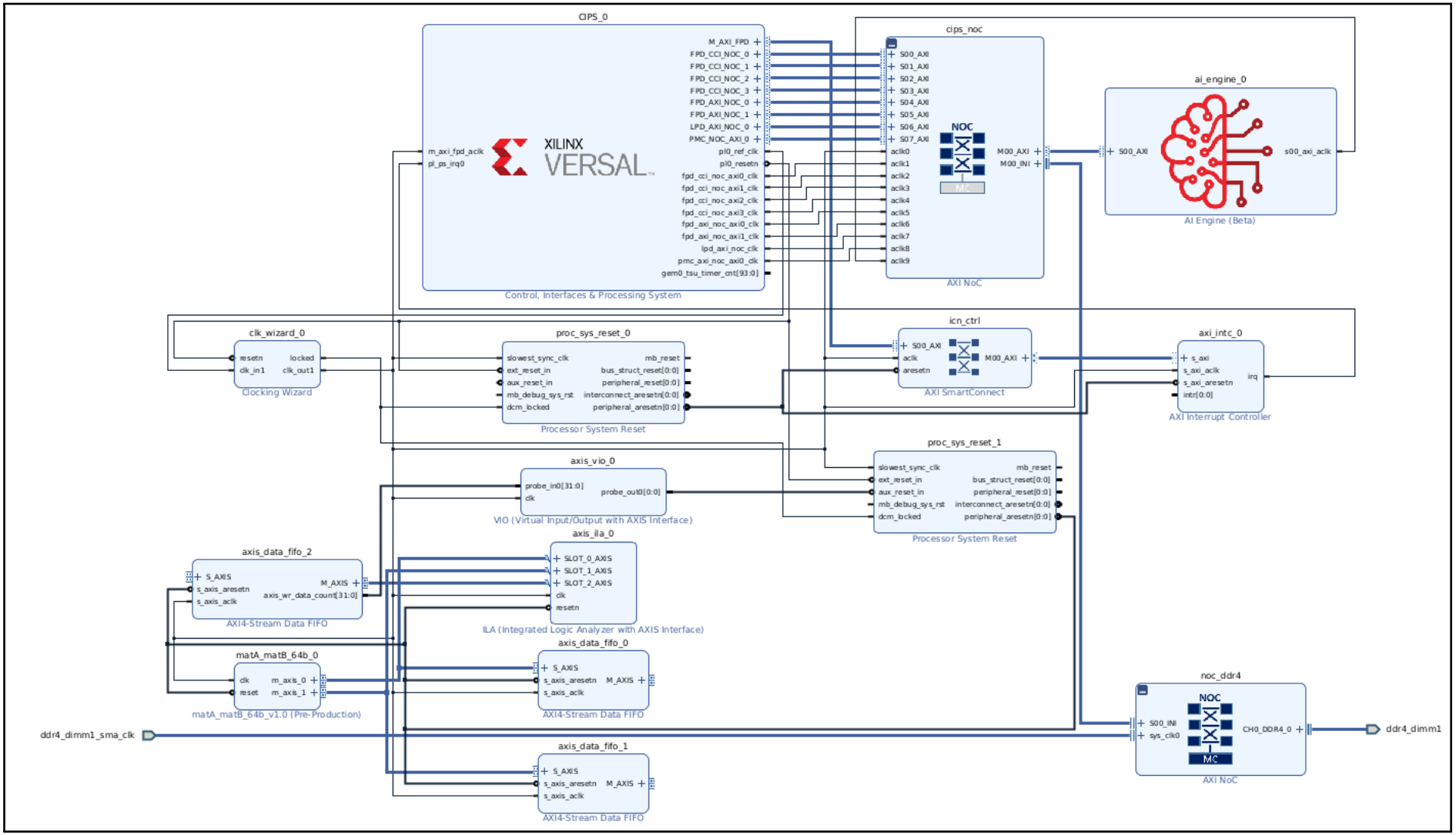
erişim için kullanıldığından tasarıma eklenmesi gerekmektedir. Matris çarpıcı bloğu için AI Engine bloğu eklenmesi gerekmektedir. Ancak AI Engine bloğunu kapalı kutu olarak tasarıma eklenmesi gerekmektedir. Yani giriş çıkış portu olmadan sadece AXI portu bağlı olacak şekilde bağlantısı yapılıyor. AI Engine bloğuna girmesi gereken portları açıkta bırakılması gerekmektedir. AI Engine bloğundan data alacak portunda açıkta bırakılması gerekmektedir. Şekil 3.10'da gösterildiği gibi bağlantıların yapılması gerekmektedir.



Şekil 3.8 : Model tabanlı AI Engine similasyon çıktısı.



Şekil 3.9 : Extensible Proje gösterimi.



Şekil 3.10 : Base Platform dizayn.

Platform Setup sayfasında AXI Stream Port sekmesinde boşta kalan portlara SP Tag verilmesi gerekmektedir. Burada verilen Tag Vitis Tools Flow içerisinde connectivity liste içerinde kullanılarak port bağlantılarının yapılması sağlanıyor.

```
1 [connectivity]
2
3 sc=matA:ai_engine_0.inA
4 sc=matB:ai_engine_0.inB
5 sc=ai_engine_0.outC:matC
```

Şekil 3.11 : Bağlantı listesi.

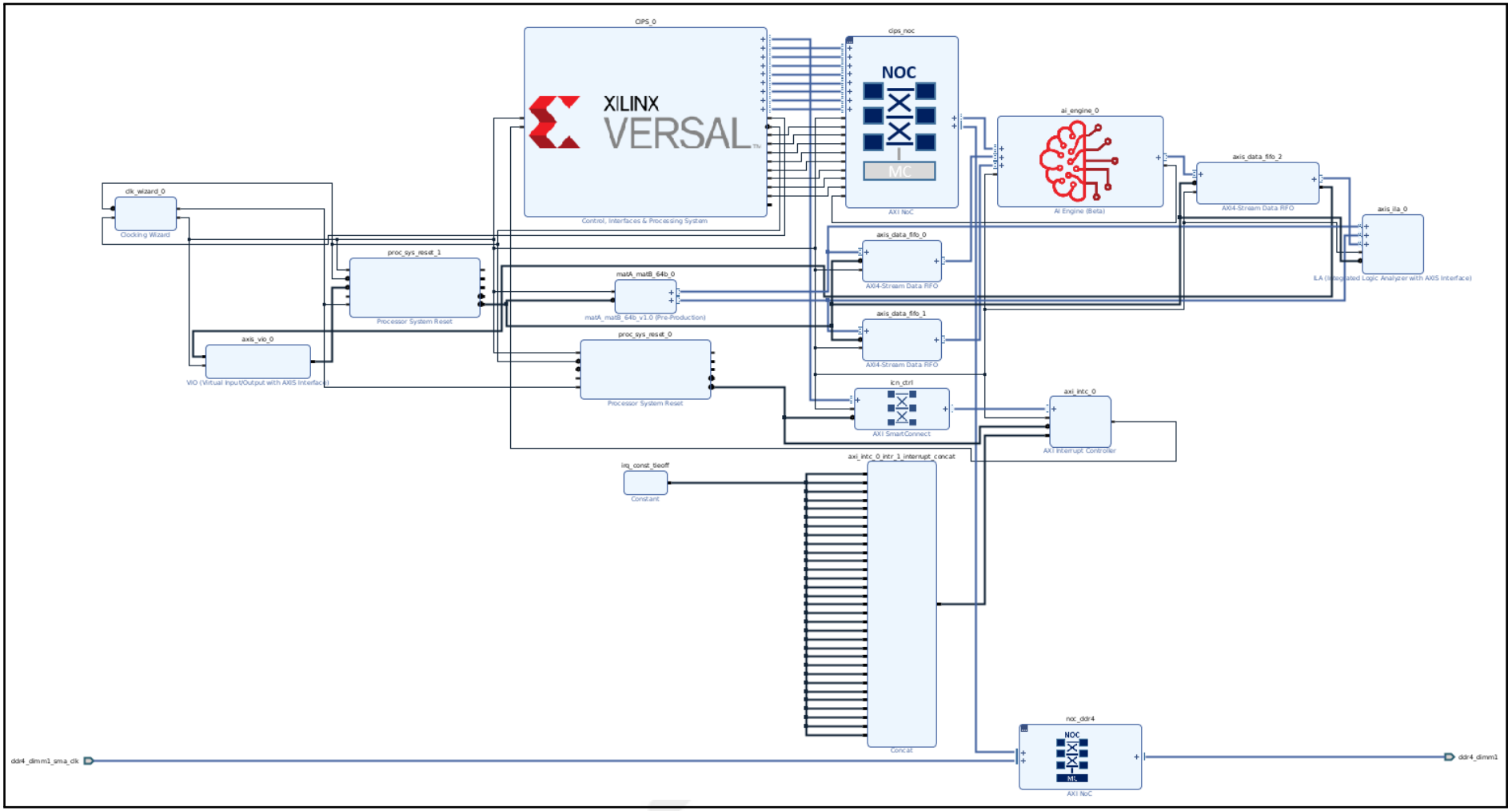
Connectivity list içerisinde boşta kalan portların AI Engine portları ile eşlenmesi yapılır. Şekil 3.11'de gösterildiği gibi AI Engine bloğunun giriş ve çıkış portları verilen SP Tagler ile eşleniyor. Şekil 3.7'da gösterilen adımların yapılabilmesi için makefile dosyası oluşturuldu. Bu makefile dosyası ile sırası ile

- Petalinux derlemesi yapılıyor.
- Platform projesi derleniyor.
- Petalinux derlemesi sonucu oluşan
  - u-boot.elf
  - system.dtb
  - boot.scr
  - boot-custom.bif

  dosyalarını platform projesinin içine taşınıyor.
- AI Engine Compilerının oluşturduğu libad.a ile conectivity list kullanılarak SD kart yükü oluşturulması

yapılmaktadır.

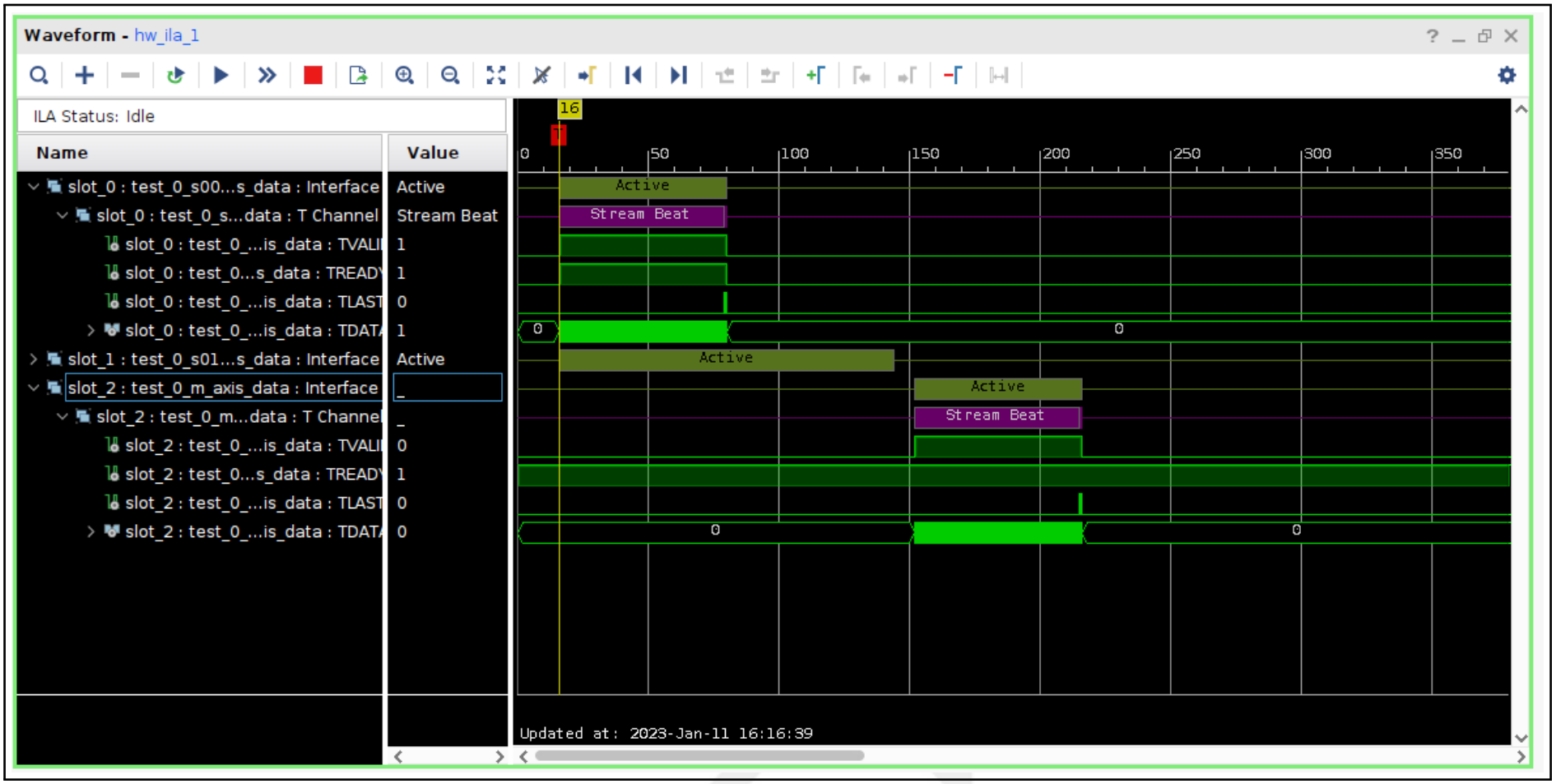
Petalinux derlemesi ve sysroot ile birleştirilmesi yapıldığında yeni bir .XPR dosyası oluşmaktadır. Bu .XPR dosyası nihai tasarımı içermektedir. Açık bırakılan tüm bağlantıların yapılmış ve eksik bağlantının kalmadığı tasarım olarak derlenmektedir.



Şekil 3.12 : AI Engine eklenmiş Vivado tasarımı.

Şekil 3.12'de nihai tasarım gösterilmektedir. Burada AI Engine bloğunun tüm bağlantıları yapılmış ve açıkta port kalmamıştır.

FPGA kartı için yük oluşturulduktan sonra testler yapıldı. Sistem testlerinde model tabanlı testler ile aynı veriler kullanıldı. FPGA testlerindeki ILA çıktısı Şekil 3.13'da gösterilmektedir. Bu test esnesinde giriş verisi olarak iki tane matris kullanılmaktadır. Matris elemanları AXI Stream protokülü çerçevesinde örneklemeler halinde sisteme alınmaktadır [26]. Her matris için gönderilecek verinin bittiğini göstermek için tlast sinyali kullanıldı. İki matrisin siteme alınmasından sonra matris çarpım işlemleri yapılmaktadır. Matris çarpma işlemi yapıldıktan sonra blok çıkışına veriler AXI Stream protokolü ile verilmektedir. Çıkışında da son veri gönderildiğinde paketin bittiğini göstermek için tlast sinyali kullanılmaktadır.



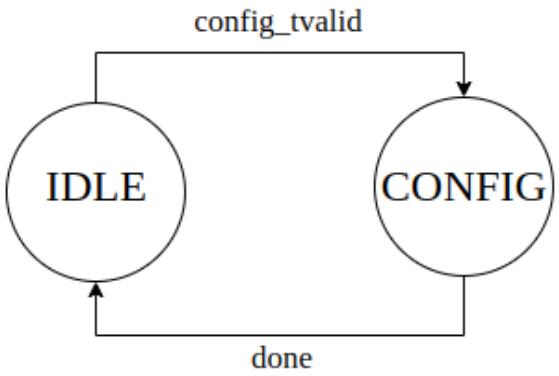
Şekil 3.13 : AI Engine FPGA testi ILA gösterimi.

4. MODEL TABANLI QC-LDPC KODLAYICI BLOĞU TASARIMI

Model Composer [17] Xilinx firmasının model tabanlı tasarım ortamıdır. MATLAB [18] firması ile birlikte geliştirilen ortamda FPGA dilleri kullanılmadan AND, OR, Counter vb. gibi temel bloklar ile tasarım yapılmasına imkan sağlanmaktadır. Yapılan tasarım FPGA atılmadan önce similasyon ortamında testleri yapılabilmektedir. Bu sayede tasarım FPGA içerisinde çalışıyormuş gibi test edilebilmektedir. Yapılan test sonuçları ile kart üzerinde alınan sonuçlar bir birleri ile aynı olmaktadır. FPGA programlama dilleri olan Verilog ve VHDL ile uzun tasarımlar yerine model tabanlı olarak tasarım ve test imkanı Model Composer tasarım ortamı ile sağlanabilmektedir.

Model tabanlı tasarım öncesinde QC-LDPC bloğunun ihtiyacı olduğu bilgileri oluşturabilecek ufak bir tasarım ile tasarım ortamların karşılaştırılması yapıldı. Tasarımda 3 tasarım ortamı karşılatırıldı;

- Xilinx Model Composer
- MATLAB HDL Coder
- Donanım Tanımlama Dili (Verilog)



Şekil 4.1 : Config bloğu durum makinesi.

Üç tasarım ortamında da gelen sinyal üzerinden aynı çıkarımlar yapıldı. Bu çıkarımlar QC-LDPC kodlayıcı bloğunda kullanılacak olan Temel Matris (Base Graph 1 - $BGN1$ / Base Graph 2 - $BGN2$), Yükseltgenme Faktörü (Lifting Size - Z_c), Kod Bloğu Uzunluğu (Code Block - K), Kod Oranı'dır (Code Rate - R). Şekil 4.1'de görülen durum makinesini uyan tasarımlar yapıldı. Yapılan tasarımlar karşılaştırıldığında 4.1'de olduğu gibi bir kaynak karşılaştırılması elde edildi. Çizelge 4.1'de görüldüğü gibi HDL Coder ile Model Composer kaynak tüketimleri benzer olduğu ancak Model Composer ile yapılan tasarımın kaynak tüketimi açısından daha iyi olduğu gözlemleenmektedir. Verilog ile yapılan tasarımda daha az kaynak tüketmesine rağmen tasarım süresi bakımından Model Composer ile yapılan tasarım bir birim süre almasına karşılık Verilog ile yapılan tasarım yaklaşık olarak altı birim süre aldığı gözlemlendi. Tasarım süresinin uzamasının nedenlerinden biri yapılan tasarımın matematiksel doğrulma için MATLAB üzerinde yapıldıktan sonra Vivado Design Suit üzerinde gerçekleme yapılmasıdır. Model Composer veya HDL Coder tasarım ortamlarında tasarımın daha hızlı gerçeklenmesinin MATLAB ortamı içerinde olmasıdır.

Çizelge 4.1 : Platform kaynak tüketim karşılaştırması.

Tasarım	Kaynak Tüketimleri		
	DSP Bloğu	Flip Flop (FF)	Look Up Table (LUT)
Model Composer	4	759	1179
HDL Coder	4	872	1347
Verilog	3	508	903

Kaynak tüketimi ve tasarım süreleri karşılaştırıldığı zaman tasarıma Model Composer üzerinde devam edilmesine karar verildi.

4.1 Model Composer Tasarım Ortamının Kurulması

Tasarım süreci boyunca Vivado 2021.1 sürümü Linux ortamında kullanıldı. Model Composer tasarım ortamında tasarım yapılabilmesi için MATLAB yazılımana ihtiyacı duyulmaktadır. Tasarım süresi 2021.1 sürümü ile yapıldığından dolayı MATLAB'in 2020a sürümü kullanılması gerekmektedir. Linux ortamında Vivado kurulumundan

önce temel kütüphanelerin kurulması gerekmektedir. Şekil 4.2'de görüldüğü gibi kütüphanelerin yüklenmesi gerekmektedir. Kütüphaneler yüklenmediği durumda Vivado 2021.1 yükleme işlemi son adımda takılmaktadır.

Şekil 4.2 : Gerekli kütüphanelerin yüklenmesi.

sh dosyası içerisine yazdıktan sonra

./initial-setup-libs.sh

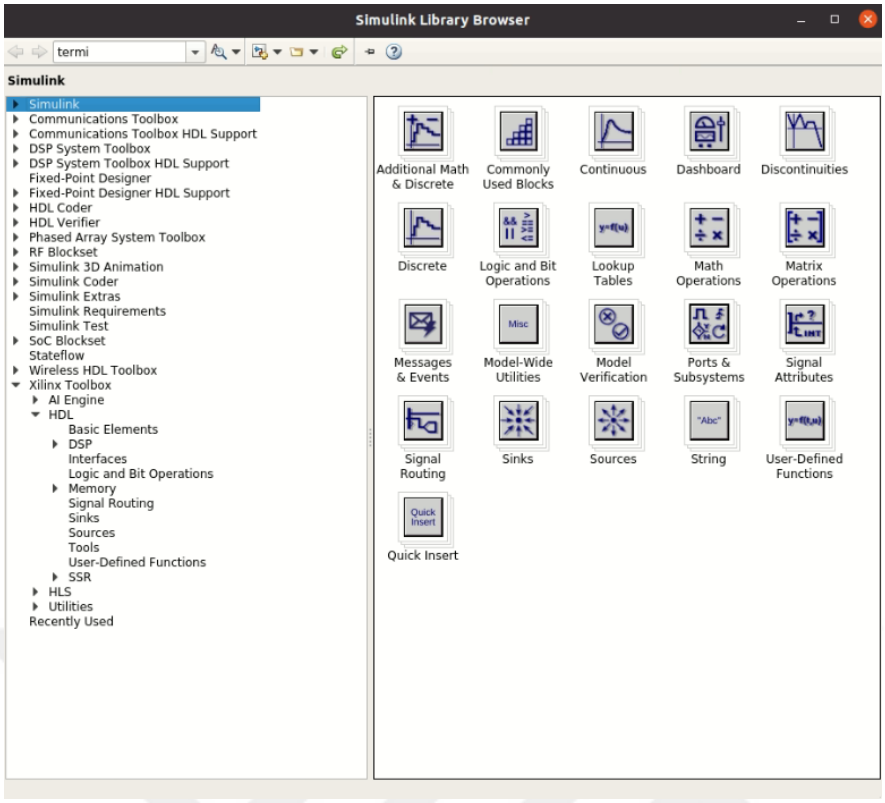
komutunun koşulması gerekmektedir. Kütüphaneler kurulumundan sonra Vivado Vitis kurulumu yapılıyor. Vitis kurulumu ile birlikte Vitis, Vivado ve Model Composer aynı anda yüklenmektedir. Yüklemeler yapıldıktan sonra Model Composer'ın açılabilmesi için şu şekilde önce Model Composer'ın source edilmesi gerekmektedir, daha sonrasında da terminal üzerinden açılması gerekmektedir:

source /tools/Xilinx/Vivado/2021.1/settings64.sh

model_composer

Model Composer uygulması açıldığında, MATLAB uygulaması açılıyor. Açılan MATLAB uygulaması üzerinden Simulink açılması gerekiyor. Simulink açıldığında Library Browser sekmesi açıldığında kullanılabilecek bloklar gösterilmektedir. Şekil 4.3'da görülebildiğii gibi birden çok alt başlık mevcuttur.

Model Composer ile yapılan tasarımlarda Xilinx Toolbox altında bulunan blokların kullanılması gerekmektedir. Bunların içerisinde basit toplama, çıkarma, sayıcı, üst alma, sayısal işaret işleyici blokları, hafıza blokları gibi çeşitli alt bloklar mevcut. HDL Coder veya Simulink sekmesinde bulunan bloklarda kullanılabilir tasarımda ancak



Şekil 4.3 : Simulink araçları.

tasarımın derlenmesi istenildiğinde bu bloklarda hata alınmaktadır. Gerçeklemeye yönelik tasarım yapılamamaktadır. Sadece similasyon olarak yapılabilmektedir.

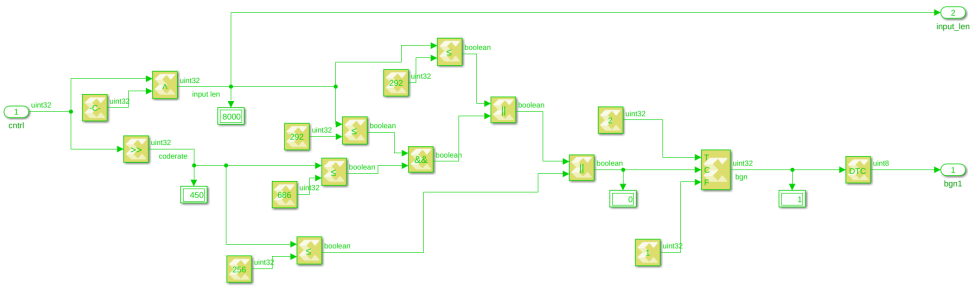
4.2 Model Composer ile Model Tabanlı QC-LDPC Tasarımı

4.2.1 QC-LDPC konfigürasyon blok tasarımı

Bu bölümde, Bölüm 2'de matematiksel olarak ifade edilen QC-LDPC kodlayıcı bloğunun tasarımını Xilinx Model Composer üzerinde tasarımı anlatılacaktır. QC-LDPC kodlayıcı bloğu birden fazla alt bloktan oluşacak şekilde tasarım oluşturuldu.

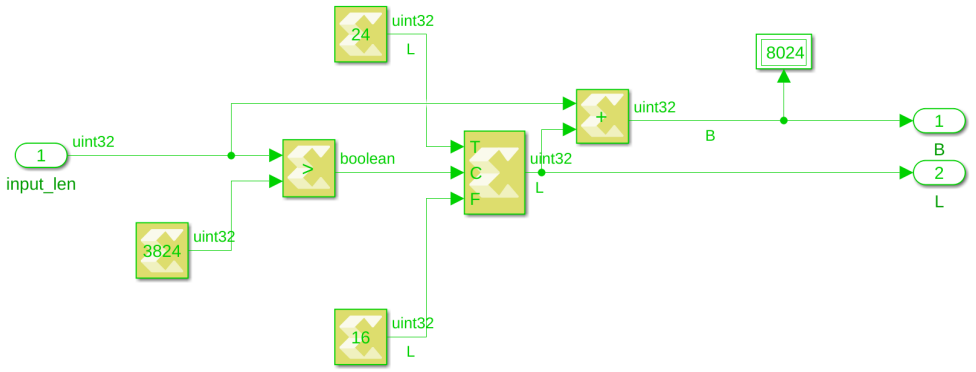
İlk olarak tasarımda blok dışından verilen konfigürasyon bilgilerinin neler olacağı kararlaştırıldı. Buna göre tasarım içerisinde kullanılması gereken parametrelerin elde edilmesi sağlanmaktadır. Konfigürasyon bilgisi olarak kod oranı ve giren data boyutu kullanılmasına karar verildi. Bu iki bilgi ile gerekli diğer bilgilerin çıkarımının yapabileceği 3GPP 38.212 [2]'de belirtilmektedir. 32 bit boyutunda konfigürasyon

datasının oluşturulması planlandı. İlk 16 bitine giriş veri boyutu, ikinci 16 bitlik kısmına ise kod oranı yazılması kararlaştırıldı.



Şekil 4.4 : Parametre hesaplama blok gösterimi.

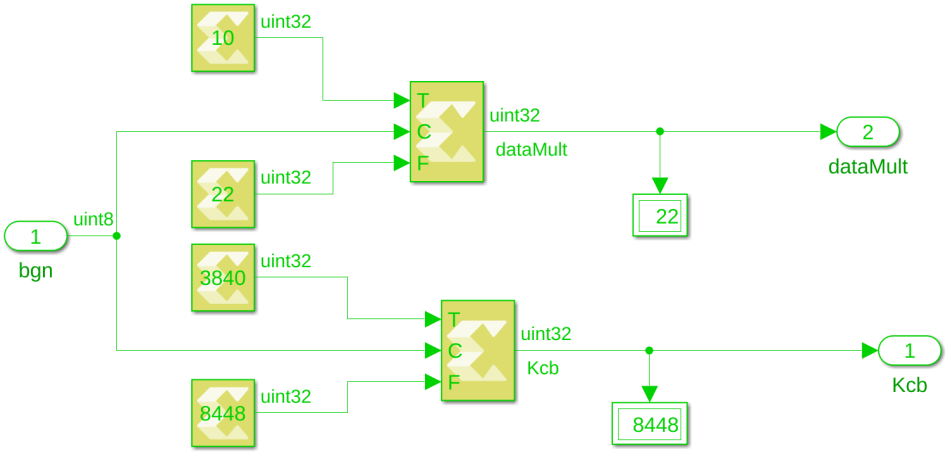
Şekil 4.4'de gösterildiği gibi kod oranı ve giriş veri boyutundan hangi Base Graph'ın kullanılacağı bilgisi çıkartılabilmektedir. Örnek kod oranı gösterimi (450/1024) olmasına rağmen parametre olarak 450 girilmektedir. Bu alt blokta gelen konfigürasyon datası 2 parçaya ayrılmaktadır. İlk 16 bitlik kısımda giriş veri boyutu tutulmaktadır. Giren boyutu 292'den küçük ise direk olarak Base Graph 2 seçilmektedir. Ayrıca kod oranı 256'dan küçükse de Base Graph 2 seçilmektedir. Ancak giriş veri boyutu 3824'den küçük ve kod oranı 686'dan küçük ise yine Base Graph 2 seçilmesi gerekmektedir. Diğer tüm durumlarda Base Graph 1 seçilmesi 3GGP 38.212 [2]'ye göre gerekmektedir.



Şekil 4.5 : CRC eklendikten sonra oluşacak taşıma bloğu boyut hesaplama gösterimi.

Giriş veri boyutu konfigürasyon mesajı içerisinden çıkarıldıktan sonra kaç bitlik CRC eklmesi yapılması gerektiğinin bilgisinin çıkarılması gerekmektedir. Şekil 4.5'de kaç bitlik CRC ekleneceği ve eklenen bitler ile taşıma bloğunun boyutunun ne olacağının hesabı gösterilmektedir.

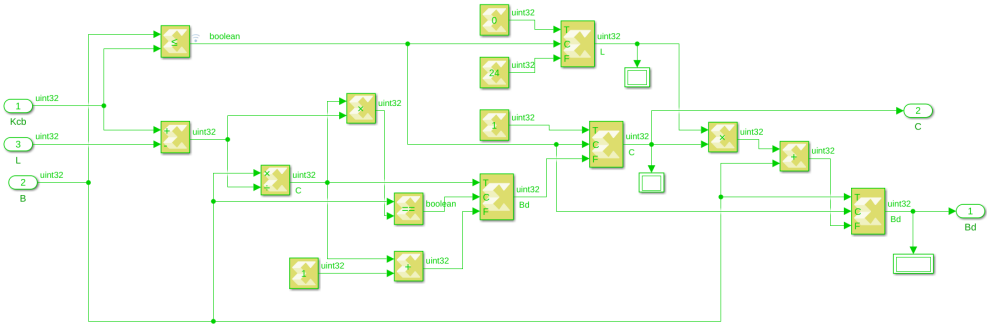
Standart çerçevesinde kod blokların boyutları belirlenmiştir. 3GPP 38.212 [2]'de Base Graph 1 için kod blok boyutu 8448, Base Graph 2 için kod blok boyutu 3840 olarak belirtilmiş. Şekil 4.6'de gösterildiği gibi hangi base graphın kullanılacağı bilgisi ile kod blok boyutu hesaplanması yapılmaktadır.



Şekil 4.6 : Kod blok boyut hesaplanması.

Kod blok boyutu belirlendikten sonra giriş verisinin boyutuna göre kaç tane kod bloğunun olacağıın hesaplanması gerekmketedir. Bir öncceki adımda hesaplanan kod blok boyutu ile giriş verisinin boyutu karşılaştırılması yapılması gerekmektedir. Giriş datasının boyutu kod blok boyutuna eşit ise veya daha az ise bir tane kod blok kullanılması gerekmektedir. Ancak giriş verisinin uzunluğu kod blok uzunluğundan fazla ise giriş verisi kod blok uzunluğunda parçalara bölünmesi gerekmektedir. Giriş veri boyutu kod blok boyutuna bölündüğünde ve yukarı yuvarlandığında kullanılması gerekli olan kod blok sayısı ortaya çıkmaktadır. Toplam veri boyutunda her kod blok sonuna CRC eklenmesi gerektiği için giriş veri boyutuna kod blok sayısı kadar CRC boyutu eklenmesi gerekmektedir. Şekil 4.7'da kod blok sayısının ve toplam boyutun CRC ile hesaplanması gösterilmektedir. Bu çalışmada temel matris 1'e uygun LDPC kodlama bloğunun kendisi gerçeklenmiştir. Segmentlere ayrılmış ve CRC eklenmiş olarak sisteme veriler alınmaktadır.

Her bir kod bloğunda kaç bitlik veri olacağı bilgisi hesaplanması için 3GPP 38.212 [2] standartının 5.2.2 bölümünde $K' = B'/C$ işlemi sonrasında elde edilen değer ile Base Graph ve giriş veri boyutundan elde edilen K_b değerlerinin kullanılması ile 3GPP 38.212'de bulunan Table 5.3.2-1'in kullanılması ile Z_c değeri hesaplanması



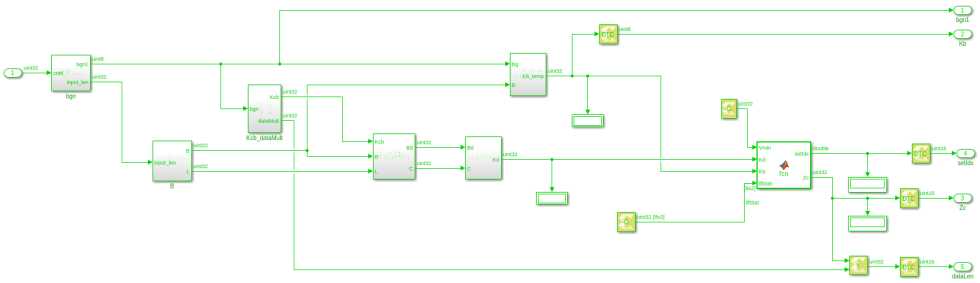
Şekil 4.7 : Kod blok sayısının hesaplanması.

yapılmaktadır. Her bir kod blokta bulunan bit sayısının hesabı Base Graph 1 için Denklem **4.1**'de, Base Graph 2 için Denklem **4.2**'de gösterilmektedir. Şekil 2.4'de Denklem **4.1** ve Denklem **4.2**'de kullanılan Z_c değerinin hesaplanmasının formülü gösterilmektedir.

$$K = 22 \times Z_c \tag{4.1}$$

$$K = 10 \times Z_c \tag{4.2}$$

Şekil 4.8'de genel olarak bilgilerin nasıl çıkarılacağına dair tasarım gösterilmektedir. Bu tasarım ile 3 çevrim süresinde kullanılacak base graph bilgisi, kullanılacak kod blok sayısı, kod blokların boyutu ve base graph oluşturulurken kullanılacak sütün bilgileri hesaplanabilmektedir.

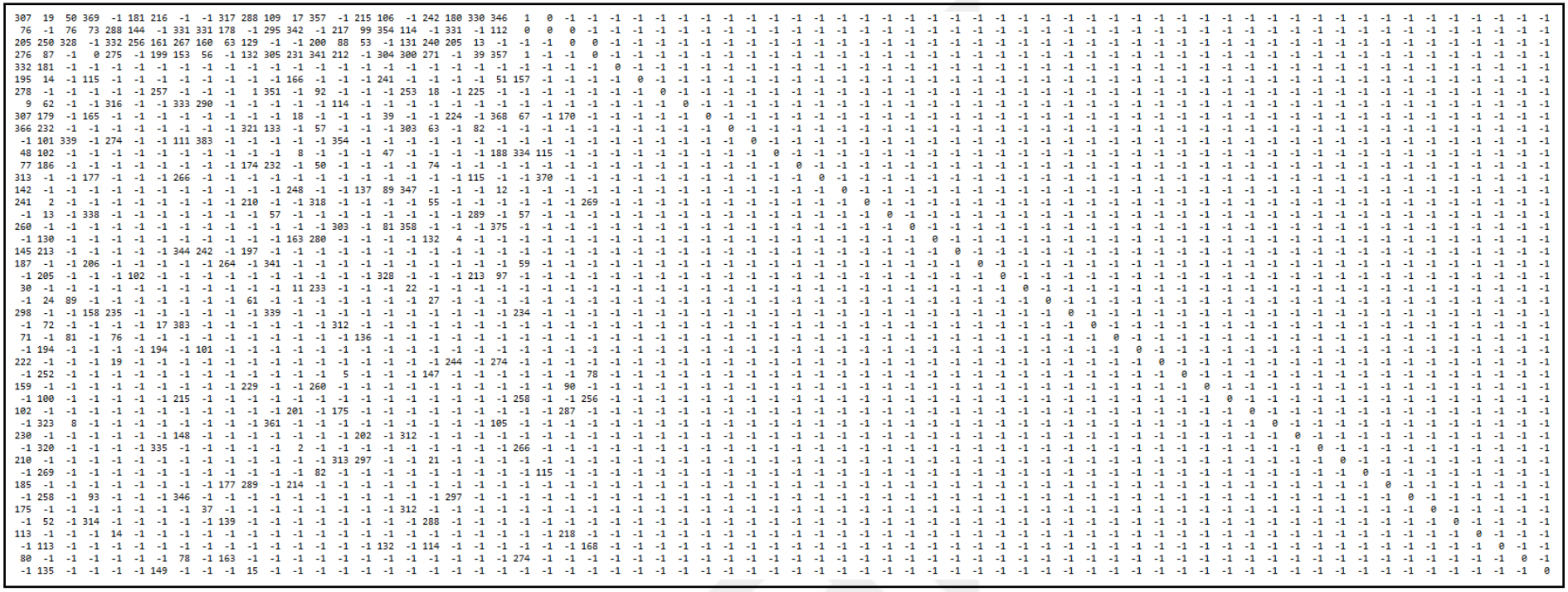


Şekil 4.8 : Bilgilerin çıkarılması.

4.2.2 QC-LDPC blok tasarımı

Konfigürasyon bilgilerinin oluşturulması işlemleri tamamlandıktan sonra kodlama işlemi için temel matrisin oluşturulması işleminin yapılması gerekmektedir. Temel matrisi oluşturulması için 3GPP 38.212 [2]'de belirtilen Base Graph 1 kullanılmaktadır. Temel matris bilgileri sadece gerekli oldukları zamanlarda kullanılacağı, içerisinde herhangi bir değişiklik yapılmayacağı ve dışarıdan alınmayacağı için Sadece Okunabilir Bellek (Read Only Memory (ROM)) içine yazılmasına karar verildi. Şekil 4.10'de 3GPP standartında belirtilen temel matris ve parite kontrol matrisleri gösterilmektedir. Şekil 4.9'de birinci temel matris'in birinci indeksinden üretilen, yükseltgenme faktörü 384 olan parite matris gösterilmektedir. Şekil 4.13'de ikinci temel matris'in birinci indeksinden, yükseltgenme faktörü 384'den üretilen parite matrisi gösterilmektedir. Şekil 4.10'de i indisleri temel matris 1 için 0'dan 45'e kadar, temel matris 2 için 0'dan 42'ye kadar artarken j indislerinde atlamalar olduğu gözlemlenmektedir. Belirtilmeyen indisler tasarım içerisinde -1 ile gösterilerek mevcut olmadıkları belirtilmiştir. Richardson-Urbanke ve Ngyuan tasarımında ilk $k_b + 4$ sütunu kullanılmaktadır [16] [19]. Matrisin hepsinin ROM'a kaydedilmesine gerekliliği yoktur. İlk $m_b \times k_b + 4$ matrisi ROM'a kaydedilmektedir. Bu yaklaşımının yapılmasının nedeni ilk k_b sutünundan sonra matris içerisinde bilgi içerek kısım bulunmamaktadır. Sadece Şekil 4.14'de gösterilen I matrisininin asal ekseninde 1 bulunmaktadır. Bu da ilgili satırın parite vektörünü göstermektedir.

Bölüm 4.2.1'da hesaplanan bgn (Base Graph Number - Temel Matris Numarası) ve set index i_{LS} değerlerine göre temel matris oluşturulması Şekil 4.11'da gösterilmektedir.. Oluşturulan matris içerisinde kullanılmaması gereken indeksdeki konumlar -1 yazılarak belirtilmektedir. Eğer konfigürasyon bilgilerine göre i_{LS} değeri 1 ve bgn değeri 1 hesaplanırsa, $V_{i,j}$ matrisi Şekil 4.12'da gösterildiği gibi olmaktadır. İlk satırda (i = 0) 5. sütunda (j=4) -1 kullanılması matris içerisinde o noktanın olmadığını göstermektedir.



Şekil 4.9 : Birinci temel matrisin bir numaralı indexinden üretilen yükseltgenme faktörü 384 olan temel matris.

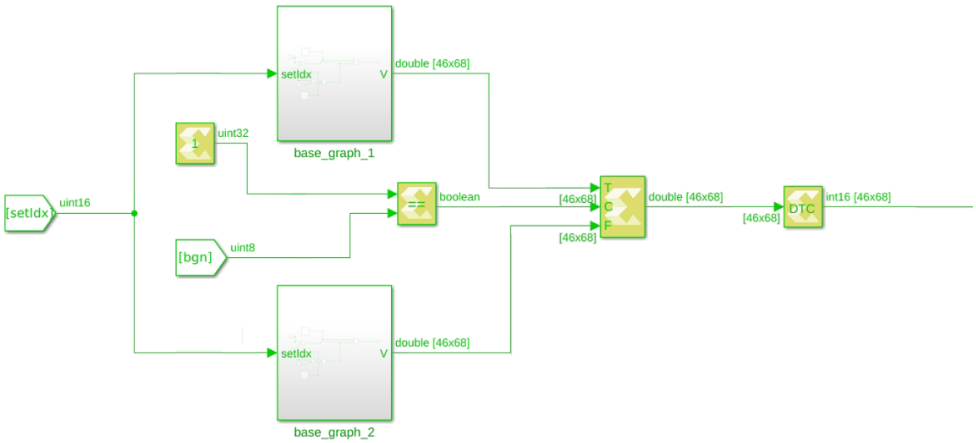
H_{BG}		$V_{i,j}$							
Row index i	Column index j	Set index i_{LS}							
		0	1	2	3	4	5	6	7
0	0	250	307	73	223	211	294	0	135
	1	69	19	15	16	198	118	0	227
	2	226	50	103	94	188	167	0	126
	3	159	369	49	91	186	330	0	134
	5	100	181	240	74	219	207	0	84
	6	10	216	39	10	4	165	0	83
	9	59	317	15	0	29	243	0	53
	10	229	288	162	205	144	250	0	225
	11	110	109	215	216	116	1	0	205
	12	191	17	164	21	216	339	0	128
	13	9	357	133	215	115	201	0	75
	15	195	215	298	14	233	53	0	135
	16	23	106	110	70	144	347	0	217
	18	190	242	113	141	95	304	0	220
	19	35	180	16	198	216	167	0	90
	20	239	330	189	104	73	47	0	105
	21	31	346	32	81	261	188	0	137
	22	1	1	1	1	1	1	0	1
	23	0	0	0	0	0	0	0	0
1	0	2	76	303	141	179	77	22	96
	2	239	76	294	45	162	225	11	236
	3	117	73	27	151	223	96	124	136
	4	124	288	261	46	256	338	0	221
	5	71	144	161	119	160	268	10	128
	7	222	331	133	157	76	112	0	92
	8	104	331	4	133	202	302	0	172
	9	173	178	80	87	117	50	2	56
	11	220	295	129	206	109	167	16	11
	12	102	342	300	93	15	253	60	189
	14	109	217	76	79	72	334	0	95
	15	132	99	266	9	152	242	6	85
	16	142	354	72	118	158	257	30	153
	17	155	114	83	194	147	133	0	87
	19	255	331	260	31	156	9	168	163
	21	28	112	301	187	119	302	31	216
	22	0	0	0	0	0	0	105	0
	23	0	0	0	0	0	0	0	0
	24	0	0	0	0	0	0	0	0

H_{BG}		$V_{i,j}$							
Row index i	Column index j	Set index i_{LS}							
		0	1	2	3	4	5	6	7
15	1	96	2	290	120	0	348	6	138
	10	65	210	60	131	183	15	81	220
	13	63	318	130	209	108	81	182	173
	18	75	55	184	209	68	176	53	142
	25	179	269	51	81	64	113	46	49
	37	0	0	0	0	0	0	0	0
16	1	64	13	69	154	270	190	88	78
	3	49	338	140	164	13	293	198	152
	11	49	57	45	43	99	332	160	84
	20	51	289	115	189	54	331	122	5
	22	154	57	300	101	0	114	182	205
	38	0	0	0	0	0	0	0	0
17	0	7	260	257	56	153	110	91	183
	14	164	303	147	110	137	228	184	112
	16	59	81	128	200	0	247	30	106
	17	1	358	51	63	0	116	3	219
	21	144	375	228	4	162	190	155	129
	39	0	0	0	0	0	0	0	0
18	1	42	130	260	199	161	47	1	183
	12	233	163	294	110	151	286	41	215
	13	8	280	291	200	0	246	167	180
	18	155	132	141	143	241	181	68	143
	19	147	4	295	186	144	73	148	14
	40	0	0	0	0	0	0	0	0
19	0	60	145	64	8	0	87	12	179
	1	73	213	181	6	0	110	6	108
	7	72	344	101	103	118	147	166	159
	8	127	242	270	198	144	258	184	138
	10	224	197	41	8	0	204	191	196
	41	0	0	0	0	0	0	0	0
20	0	151	187	301	105	265	89	6	77
	3	186	206	162	210	81	65	12	187
	9	217	264	40	121	90	155	15	203
	11	47	341	130	214	144	244	5	167
	22	160	59	10	183	228	30	30	130
	42	0	0	0	0	0	0	0	0
	1	249	205	79	192	64	162	6	197
	5	121	102	175	131	46	264	86	122

Şekil 4.10 : 3GPP birinci temel matris ve parite kontrol matrisleri $V_{i,j}$ [2].

Şekil 4.12'da gösterilen temel matris oluşturulduktan sonra genişleme faktörü (Z_c) kullanılarak nihai temel matris elde ediliyor. Denklem **2.11**'de kaydırma katsayılarının hesaplarının nasıl olacağını göstermektedir. Bu denkleme göre H_{BG} matrisi elemanlarından -1 olmayan ve Z_c'den büyük olanların Z_c'ye göre modu alınmaktadır. Kalan değer sağ kaydırma miktarını göstermektedir. Mod işleminin yapılmasının nedeni giriş datalarının boyutu $k_b \times Z_c$ boyutunda olması ve her bir elemanın maksimum Z_c kadar kaydırılabilmesindendir. Z_c boyutundan fazla bir değer ile kaydırma işlemine tabi tutulur ise tekrara düşülmektedir. Bu yüzden işlem yükünü azaltmak için en baştan sadeleştirilmiş kaydırma matrisi oluşturulmaktadır. H_{BG} matrisi elemanlarından -1 olanlara ise herhangi bir işlem yapılmamaktadır. Bu işlemi yapan blok tasarımı Şekil A.1'de gösterilmektedir.

Tasarım çerçevesinde giriş verileri CRC eklenmiş ve kod bloklara ayrışmış olarak sisteme alınmaktadır. Tasarımda sadece LDPC kodlaması yapılmaktadır. Bu yüzden sisteme alınan her bir kod blok, temel matris bilgisine göre $K = 22 \times Z_c$ boyutlarında olması gerekmektedir. Sisteme alınan bit sekansı matris formatına çevrilmesi işlemi yapılmaktadır. K boyutunda sistematik bit sisteme girdiğinde $Z_c \times (n-m)$ boyutunda



Şekil 4.11 : Temel matris üretimi.

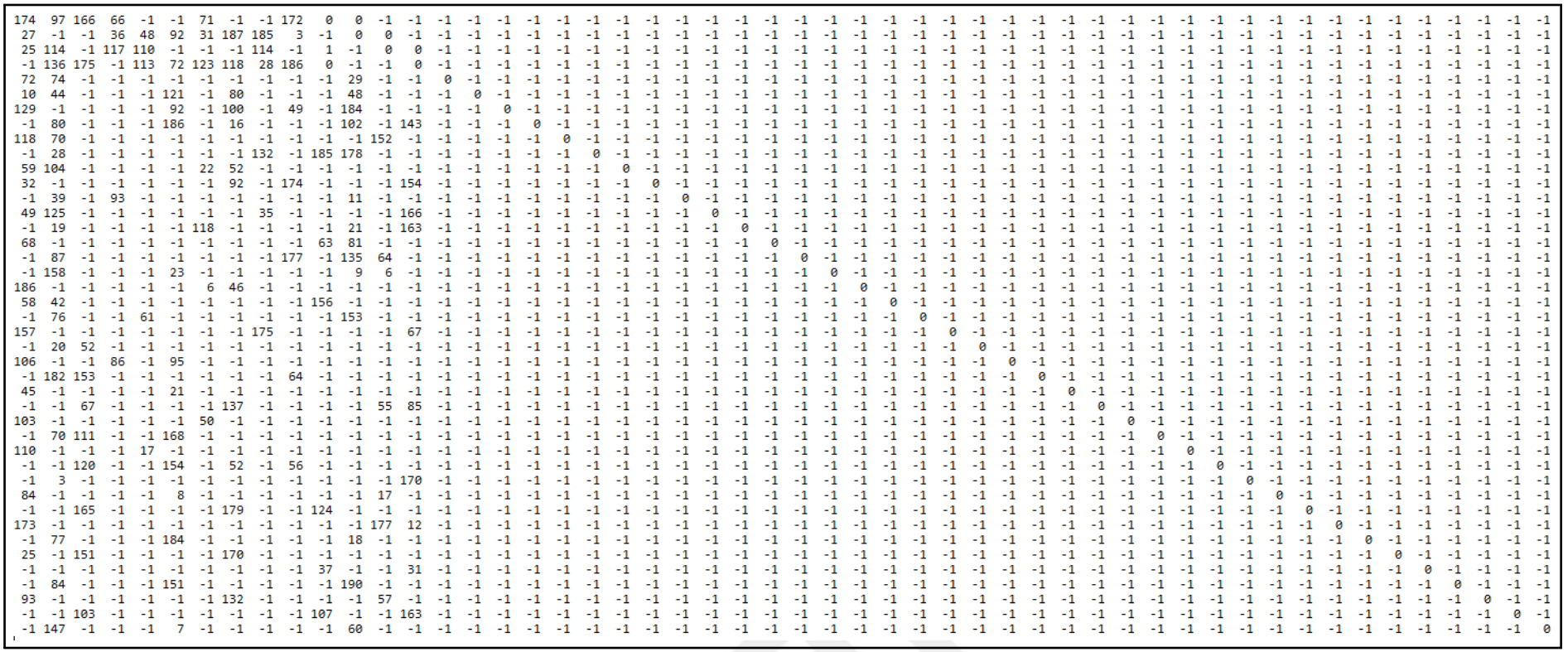
307	19	50	369	-1	181	216	-1	-1	31
76	-1	76	73	288	144	-1	331	331	17
205	250	328	-1	332	256	161	267	160	6
276	87	-1	0	275	-1	199	153	56	-
332	181	-1	-1	-1	-1	-1	-1	-1	-
195	14	-1	115	-1	-1	-1	-1	-1	-
278	-1	-1	-1	-1	-1	257	-1	-1	-
9	62	-1	-1	316	-1	-1	333	290	-
307	179	-1	165	-1	-1	-1	-1	-1	-
366	232	-1	-1	-1	-1	-1	-1	-1	-
-1	101	339	-1	274	-1	-1	111	383	-
48	102	-1	-1	-1	-1	-1	-1	-1	-
77	186	-1	-1	-1	-1	-1	-1	-1	-
313	-1	-1	177	-1	-1	-1	266	-1	-
142	-1	-1	-1	-1	-1	-1	-1	-1	-
241	2	-1	-1	-1	-1	-1	-1	-1	-
-1	13	-1	338	-1	-1	-1	-1	-1	-
260	-1	-1	-1	-1	-1	-1	-1	-1	-
-1	130	-1	-1	-1	-1	-1	-1	-1	-

Şekil 4.12 : Temel matris RAM gösterimi.

matris elde ediliyor. Burada n_b temel matrisin sütun boyutunu, m_b temel matrisin satır boyutunu belirtmektedir. Temel matris üzerinde anlamlı bilgilerin olduğu alan ilk $(n_b - m_b + 4)$'lık alan olduğundan bu alan boyutunda matris oluşturulmaktadır.

LDPC kodlama bloğunun çalışması için H matrisi hesabı, giriş bitlerinin oluşturulması işlemleri tamamlandıktan sonra LDPC kodlayıcı bloğunun ana hesaplama işlemleri başlamaktadır.

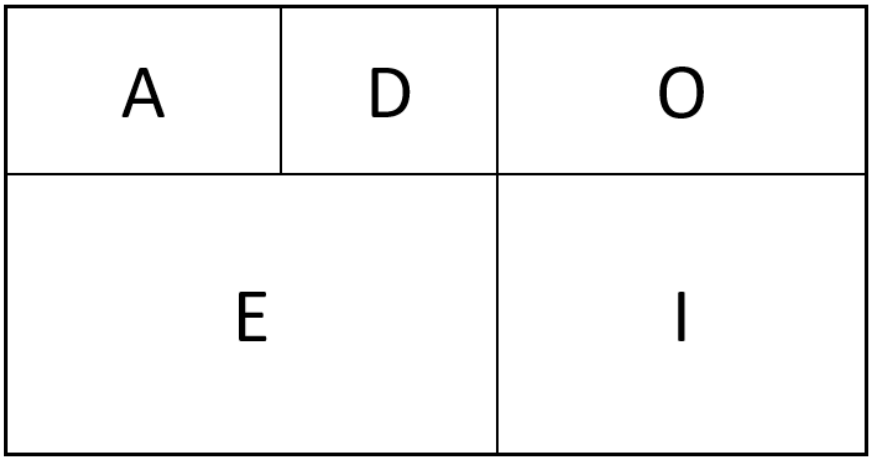
Şekil 4.14'de QC-LDPC kodlayıcı bloğunda kullanılan temel matris bölümleri gösterilmektedir. İki farklı temel matris için iki farklı m ve n değerleri mevcuttur. Temel matris 1 için $n_b = 68$, $m_b = 46$, temel matris 2 için $n_b = 52$, $m_B = 42$'dir.



Şekil 4.13 : İkinci temel matrisin bir numaralı indexinden üretilen yükseltgenme faktörü 384 olan temel matris.

- A : Mesaj Bitleri $(4 \times (n_b - m_b))$
- D : Çekirdek Parite Kontrol Bitleri (Çift Köşegen) (4×4)
- E : Genişleme Matrisi $((m-4) \times (n_b - m_b + 4))$
- I : Birim Matris $((n_b - m_b - 4) \times (n_b - m_b + 4))$
- 0 : Sıfır Matrisi $(4 \times (m_b - 4))$

D matrisinin çift köşegen olması önemlidir. Çünkü çift köşenlik özelliği kullanılarak p_1, p_2, p_3 ve p_4 parite vektörleri Denlem **2.16**'da belirtilen denklemler kullanılarak yerine koyma methodu ile hesaplanmaktadır.

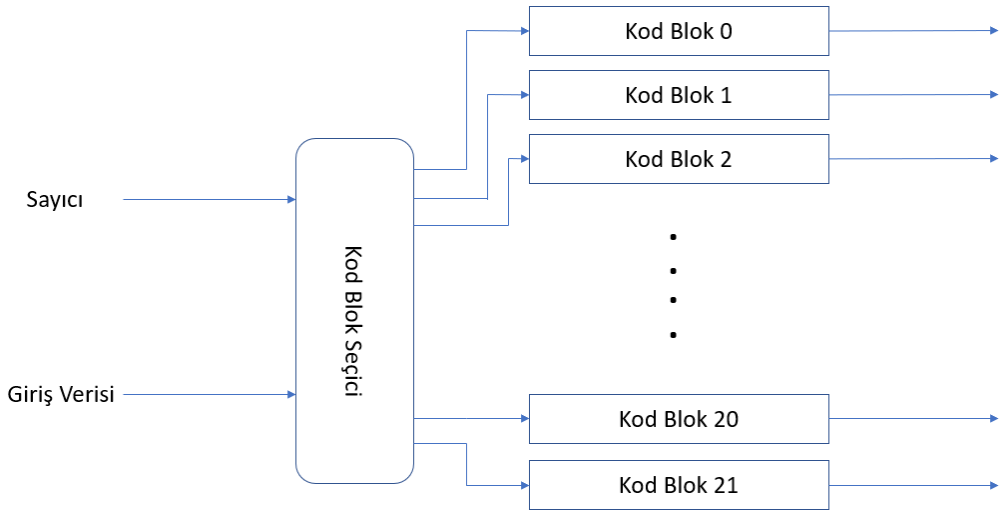


Şekil 4.14 : QC-LDPC 5G formatı.

H matrisinde bulunan elemanların her biri $Z_c \times Z_c$ boyutunda birim matrisin kaç birim sağ kaydırılacağı bilgisini taşımaktadır. Her bir kod blokta giriş verisi olarak $(n_b - m_b) \times Z_c$ bit veri gelirken çıkışında $n_b \times Z_c$ bit boyutunda data çıkması gerekmektedir. Giriş verisi olduğu gibi çıkışa verilmektedir. Ancak geriye kalan $m_b \times Z_c$ bit boyutundaki veri LDPC ile hesaplanarak çıkışa verilmektedir.

Sistem içerisine alınan datalar 32 bit halinde alınarak RAM'e kaydedilmektedir. RAM kullanılarak her bir işlem için giriş verilerinin tekrar tekrar kullanılması sağlanılmaktadır. RAM yapısı oluştururken okuma işleminin parallel bir şekilde yapılabilmesi için CAM (Content Addressable Memory - İçerik Adreslenebilir Bellek) tablosu formatında tasarlanmasına karar verildi. Şekil 4.15'de RAM yapısı

gösterilmektedir. RAM alanına bilgilerin yazılması için giriş verisinin indisi, giriş verisi ve Z_c yükseltgenme faktörü RAM bloğuna verilmesi gerekmektedir. RAM bloğu yükseltgenme faktörü ile her bir kod blok registerına kaç bit yazılacağı kontrol edilmektedir. Sayıcı değeri sistem içerisinde kod blok registerını seçmek için kullanılmaktadır. Giriş verisi kod blok registerlarına yazılacak değeleri sisteme almak için kullanılır. RAM'e veri yazılması işlemi yapılamadığı her an RAM'den okuma işlemi yapılabilir olarak kullanılmaktadır.

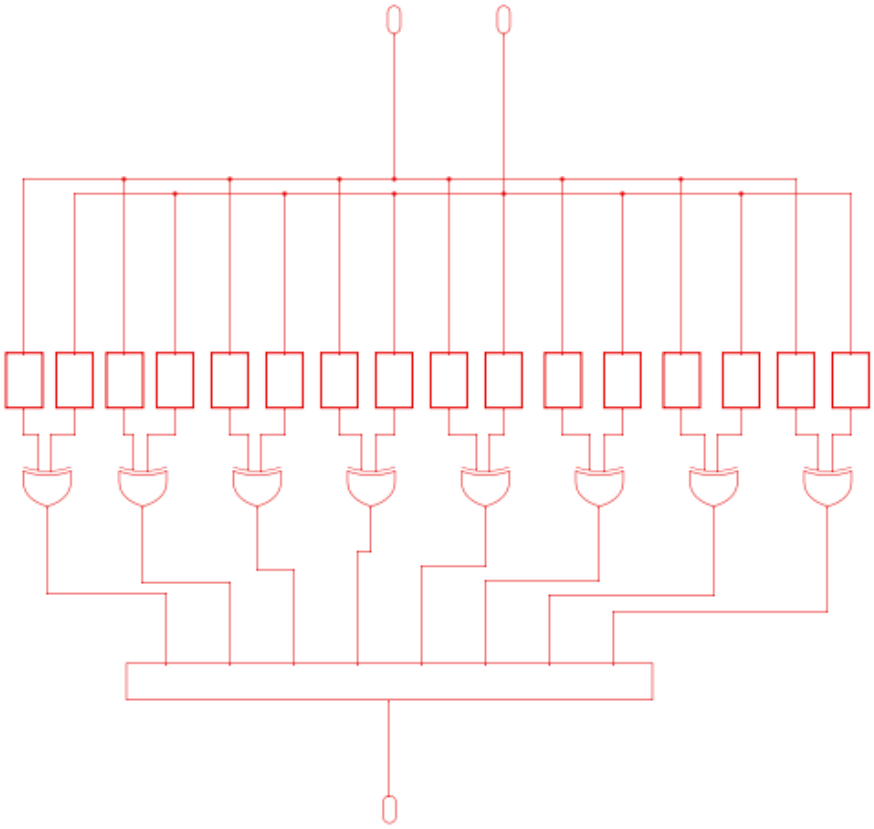


Şekil 4.15 : RAM okuma yazma bloğu.

LDPC kodlama bloğu için ilk olarak Nguyen'nin belirtmiş olduğu gibi p_a vektörlerinin hesaplanması gerekmektedir [19]. Giriş verileri k_b parçaya ayrılmıştır. Her bir parçanın boyutu Z_c bit boyutundadır. Giriş verileri, parçalara ayrılma işlemi sonrasında A matrisi üzerinde işleme alınmaktadır. Bu hesaplamalar için iki tane alt blok tasarımı yapılmıştır.

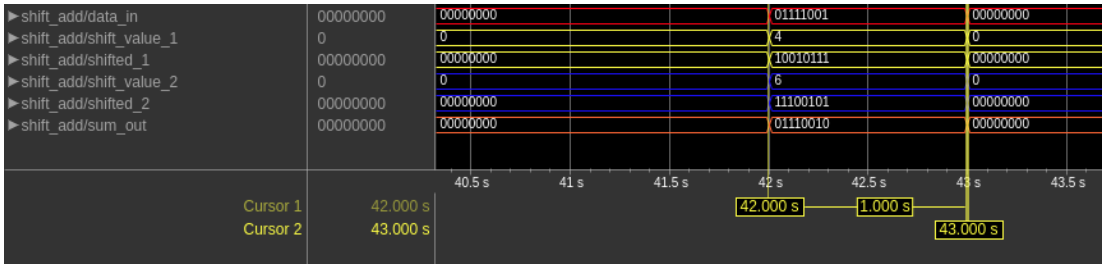
İlk olarak tasarlanan blok kaydırma bloğudur. Bloğun amacı girişte verilen Z_c bit uzunluğundaki veriyi verilen kaydırma bilgisi kadar dairesel olarak sağ kaydırmasıdır. Şekil A.2'de örnek bir kaydırma bloğu yapısı gösterilmektedir. Burada dört bitlik bir kaydırma işlemi yapılmaktadır. Kaydırma bloğu tasarımında çoğullayıcı (multiplexer) yapıları kullanılmıştır. Örnek kaydırma bloğu dört bitlik bir yapıda olduğu için dört

tane dört bitlik çoğullayı ihtiyacı oluşmuştur. Örnek kaydırma bloğunun test sonuçları Şekil B.1'de paylaşılmıştır.



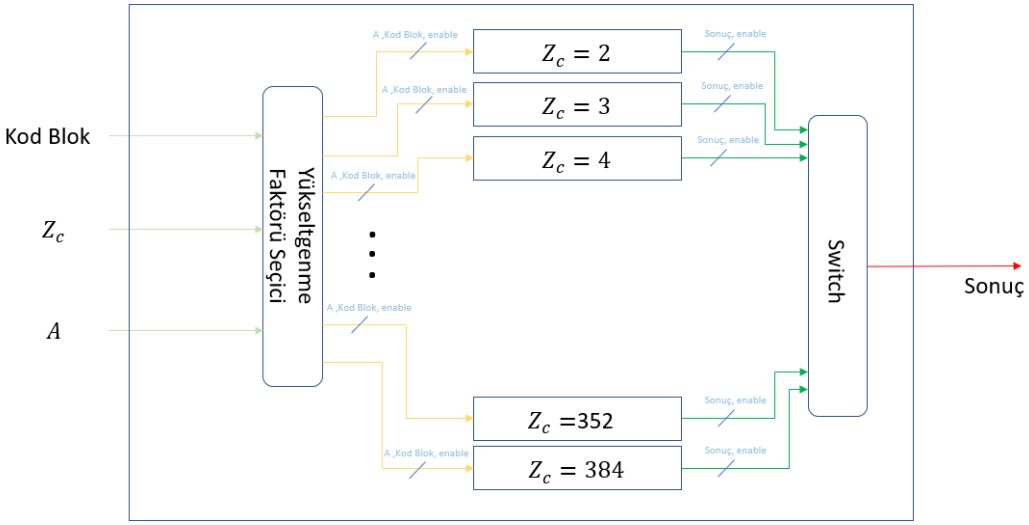
Şekil 4.16 : Sekiz bit Mod2 toplama alt bloğu.

İkinci olarak tasarlanan blok paralel olarak toplama bloğudur. Blok tasarımda bit seviyesinde işlemler yapılacağı için toplama yapıp daha sonra mod2 işlemi yapılmasının yerine xor kullanılmıştır. Şekil 4.16'da sekiz bitlik iki tane sayının toplanması için tasarlanan alt blok gösterilmektedir. Gösterim karmaşıklığının daha az olması için küçük boyutlu varil kaydırma ve mod2 toplama tasarımları gösterilmiştir. Tasarlanan bu iki blok peş peşe eklenmesi ile p_a vertörlerinin hesaplamaları yapılmaktadır. Şekil 4.17'da sekiz bitlik bir sayının farklı miktarlarda parallel bir şekilde ve mod2 olarak toplanmasının bir çevrim süresinde hesaplandığı gösterilmektedir. Bu işlemlerin tek çevrim süresinde yapılması, LDPC kodlama bloğunda giriş verilerinin A matrisi elemanları kadar kaydırılmasının parallel bir şekilde yapılabileceğini ve çıktılarının tek çevrim süresinde toplanabileceğini göstermektedir.



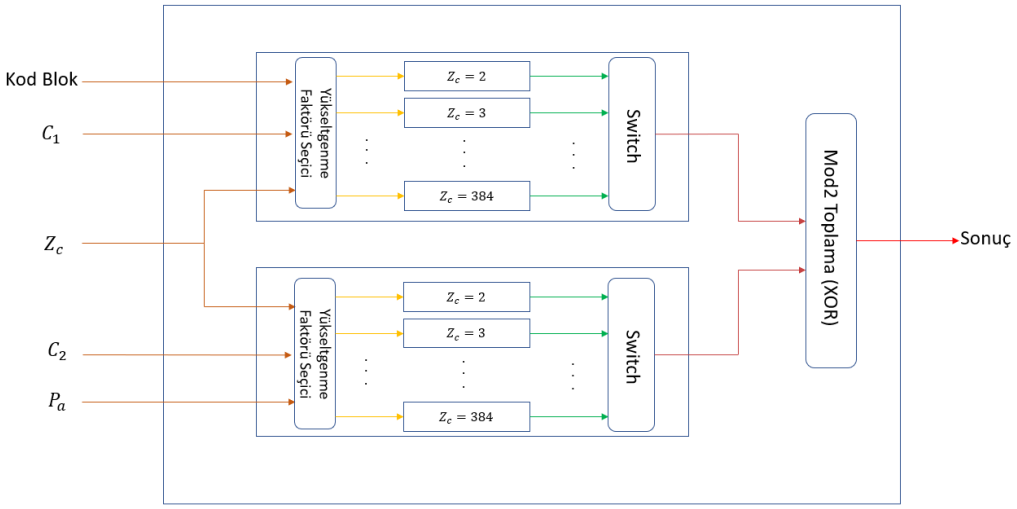
Şekil 4.17 : Sekiz bitlik kaydırma ve toplama Model Composer simülasyon çıktısı.

p_a vektörleri temel matrisi içerisinde bulunan en anlamlı kısımdan üretilmektedir. Temel matris içerisindeki en anlamlı kısım Şekil 4.14'de gösterilmekte olan A matrisidir. p_a parite vektörleri çekirdek parite kontrol vektörleridir. Şekil 4.9 ve 4.13'de gösterildiği gibi D matrisinin çift köşegen özelliği kullanılarak ilk satırda p_{a_1}, ikinci satırda p_{a_2}, üçüncü satırda p_{a_3} ve dördüncü satırda p_{a_4} parite vektörlerinin hesapları yapılmaktadır. Her bir hesaplanan p_a vektörü, Z_c boyutundaki RAM'e kaydedilmektedir. RAM alanı en büyük Z_c boyutu olan 384'e göre açılmıştır. Farklı boyutlarda da kullanılabilmesi için maksimum boyutta açılarak tasarıma esneklik sağlanmıştır. Şekil 4.18'da p_a vektörlerinin hesaplanmasında kaç bitlik kaydırma ve toplamı işlemi yapılacağının karar mekanizması gösterilmektedir. Dört çevirim süresi sonunda p_{a_1} hesaplaması tamamlanmaktadır. Daha sonraki her bir çevrim süresinde bir önce hesaplanan p_a vektörü kullanılarak hesaplamalar yapılmıştır. Toplamda 7 çevirim süresinde dört p_a vektörlerününde hesaplanması tamamlanmaktadır.



Şekil 4.18 : p_a vektör hesabında kullanılan kaydırma ve toplama blok seçilimi.

LDPC kodlama bloğunun ikinci aşaması olan p_c vektörlerinin hesaplanması için RAM'e kaydedilen p_a vektörleri, E_1 ve E_2 matrisleri kullanılmaktadır. Bu bölümde bir önceki bölümde tasarlanan ve kullanılan kaydırma ve toplama blokları kullanılmaktadır. Her bir çevirim süresi sonunda hesaplanan p_c değerler p_a vektörleri gibi rekürsif olarak kullanılma ihtiyacı olmadığından dolayı sistemden dışarı atılması için FIFO'ya yazılmaktadır. Bu kısımda $(m_b - 4)$ tane parite vektörü hesaplanmaktadır. Burada herhangi bir RAM'e yazma işlemi yapmaya gereksinim bulunmamaktadır. Şekil 4.19'da p_c vektörlerinin hesaplanma grafiği gösterilmektedir. Burada p_a vektörlerinde kullanılan λ hesaplamalarından farklı olarak ikinci bir hesaplama zinciri eklenmektedir. Burada hesaplanan p_a değerlerine karşılık E matris elemanları ile yapılan hesaplamalar sonucu oluşan değerin eklenmesi yapılmaktadır.



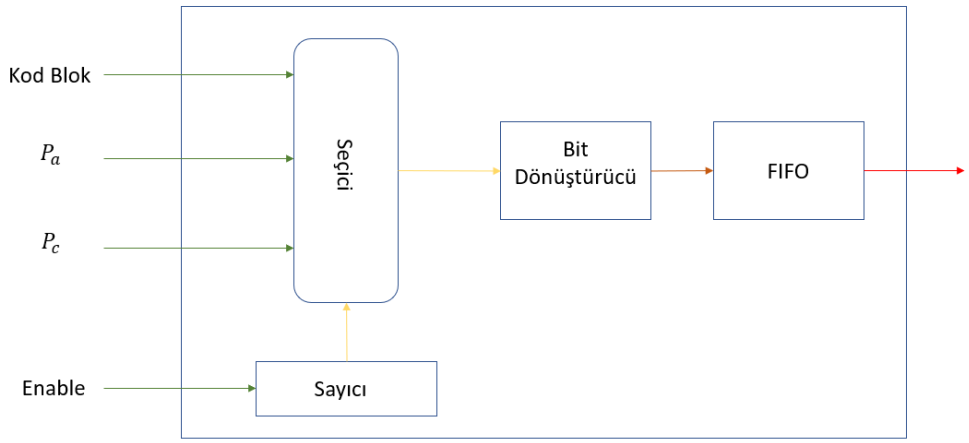
Şekil 4.19 : p_c vektör hesabında kullanılan kaydırma ve toplama blok seçilimi.

$$t_{giri} = \frac{(k_b \times Z_c)}{32} \tag{4.3}$$

LDPC kodlama bloğunun çıkışına veriler girişte alındığı gibi 32 bit olarak gönderilmektedir. Kodlama bloğuna giren verinin alış süresi Denklem **4.3**'da gösterilmektedir. İlk olarak sisteme giren veriler dışarı gönderilmektedir. Daha sonrasında temel matrisin her bir satırına karşılık gelen parite vektörü sistem dışına verilmektedir. İlk olarak p_a vektörleri daha sonrasında p_c vektörleri kodlama bloğunun çıkışına verilmektedir. Şekil 4.20'da çıkış alt blok tasarımı gösterilmektedir. Burada sayıcı bloğu ile çıkışa hangi girişten veri verileceği belirlenmektedir. Bit dönüştürücü

ile Z_c boyutundaki veri 32 bite getirilerek FIFO'ya yazılmaktadır. FIFO'dan verilerin alınması için AXI Stream protokolünde belirtildiği gibi tready verilmesi gerekmektedir. Toplam verilerin kodlayıcı bloğundan çıkış süreleri Denklem **4.4**'da gösterilmektedir.

$$t_{ßkß} = \frac{(n_b \times Z_c)}{32} \tag{4.4}$$

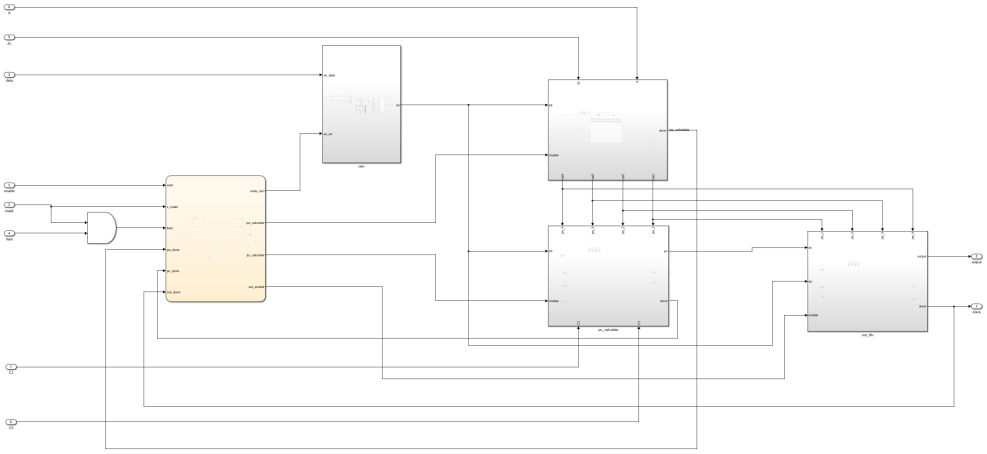


Şekil 4.20 : LDPC kodlayıcı bloğu çıkış alt bloğu.

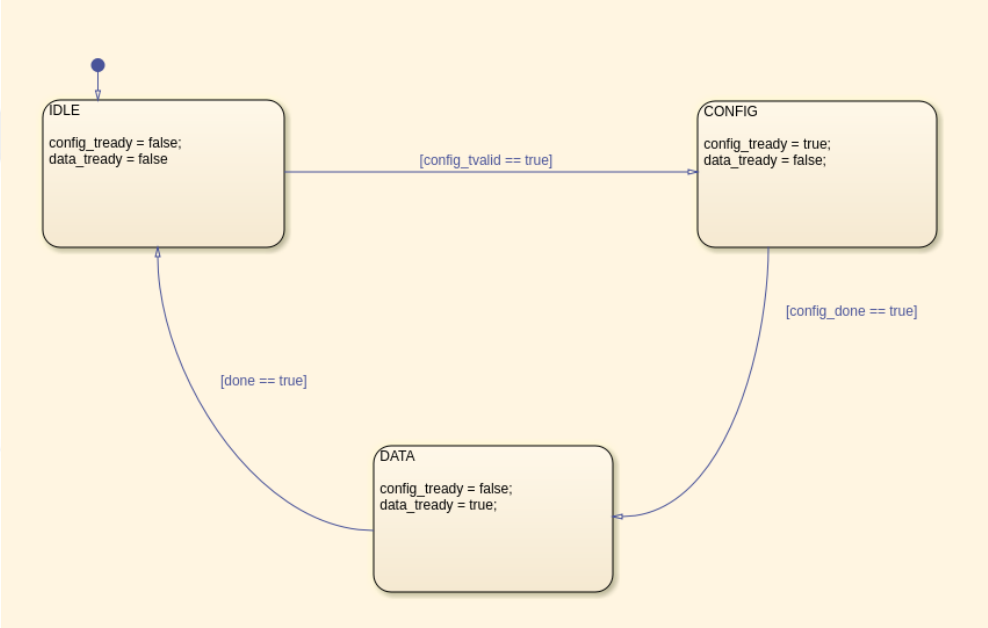
Şekil 4.21'da QC-LDPC Kodlayıcı bloğunun üst görünümü gösterilmektedir. Kod oranı (Code rate) $1/3$ için kodlayıcı çıkışında s (sistematik bitleri) ve p (parite bitleri) peş peşe eklenerek sistemden $N = 68 \times Z_c$ bit boyutunda codeword çıkmaktadır. Standart gereği ilk $2 \times Z_c$ boyutundaki Puncture dataları silinerek kanala verilmektedir. Puncture işleminden sonra sistem çıkışına $N = 66 \times Z_c$ bit boyutunda veri verilmektedir.

4.2.3 QC-LDPC ana modül tasarımı

LDPC kodlayıcı bloğu için gerekli bilgilerin hesaplanması için alt bloğun tasarımdan sonra LDPC bloğu ile konfig bloğunun senkron bir şekilde çalışması için 3 durumlu sonlu durum makinesi tasarımı yapıldı. Şekil 4.22'de sistemde kullanılan sonlu durum makinesi gösterilmektedir. Sonlu durum makinesine göre sistem IDLE konumunda yeni bir konfigürasyon gelene kadar beklemektedir. Konfigürasyon bilgisi geldiği andan itibaren CONFIG durumuna geçerek, blokta kullanılacak parametrelerin hesaplanması yapılacak. Parametrelerin hesaplanması bittikten sonra



Şekil 4.21 : QC-LDPC kodlayıcı tasarımı ana tasarımı.

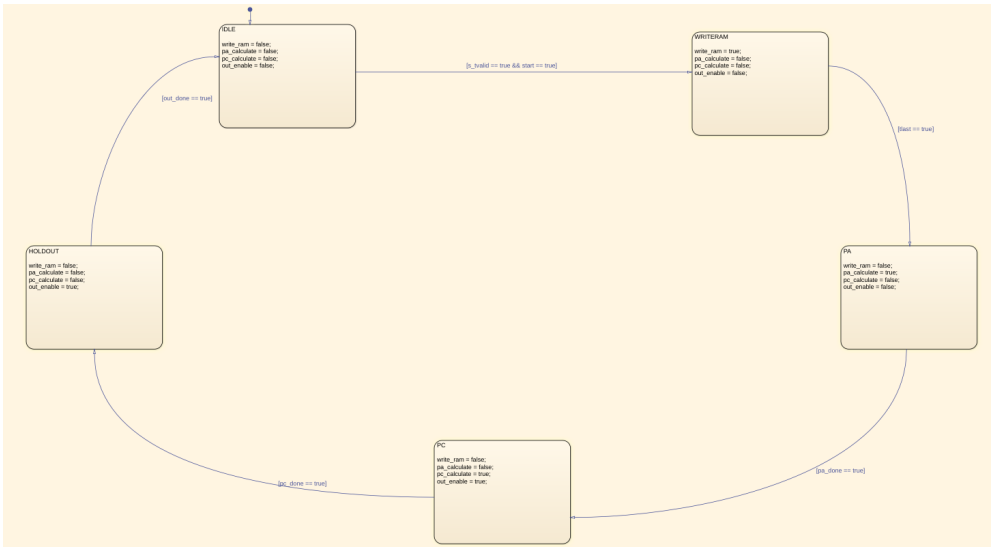


Şekil 4.22 : LDPC sonlu durum makinesi.

DATA durumuna geçerek kodlama algoritmasının çalışması başlayacaktır. DATA durumuna gelindiğinde Şekil 4.23'de gösterilen sonlu durum makinesi çalışmaktadır. Burada yapılan adımlar tek tek çalıştırılmaktadır. Sistemde, durum makineleri çalışmalarını bitirdiğinde done sinyalleri çıkmaktadır. Done sinyali çıktığında sistem IDLE konumuna gelerek yeni veri girişini beklemektedir.

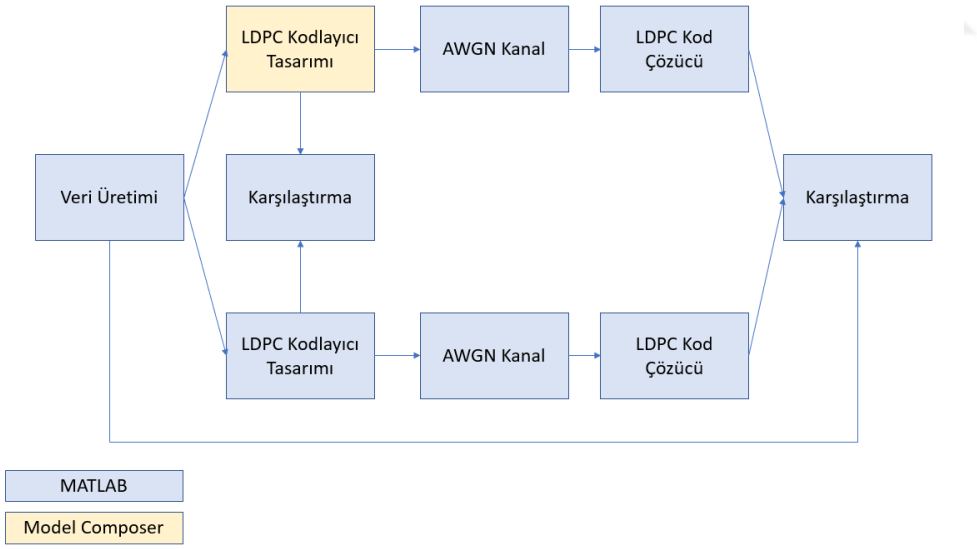
4.2.4 QC-LDPC kodlayıcı blok testi

Sistem tasarım çalışmaları bittikten sonra çalışma ortamı değiştirme ihtiyacı olmadan testlere başlanabilmektedir. Model tabanlı tasarımın en büyük avantajlarından biride



Şekil 4.23 : DATA durumuna ait durum makinesi.

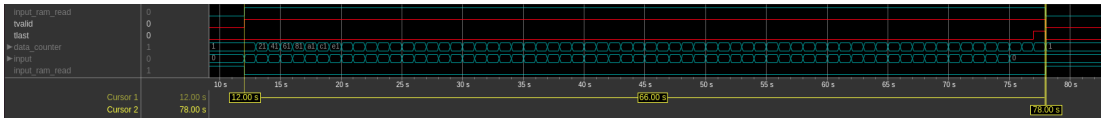
sistem testlerinin tasarım ortamında MATLAB betiklerinin kullanılarak yapılmasına olanak sağlamasıdır. Sistem testleri iki tane test senaryosu oluşturuldu. İlk olarak FPGA ihtiyacı olmadan model tabanlı olarak test, ikincisi ise FPGA içerisinde test edilmesidir. Şekil 4.24'de model tabanlı tasarım avantajı kullanılarak oluşturulan test senaryosunun blok diagram gösterilmektedir.



Şekil 4.24 : LDPC Kodlayıcı Blok Tasarım Diagramı.

Bu sistemde Model Composer üzerinde tasarlanan LDPC Kodlayıcı bloğunun çıktıları ile MATLAB 5G Toolbox içerisinde bulunan nrLDPCEncode betiği [23] çıktıları karşılatırılmaktadır.

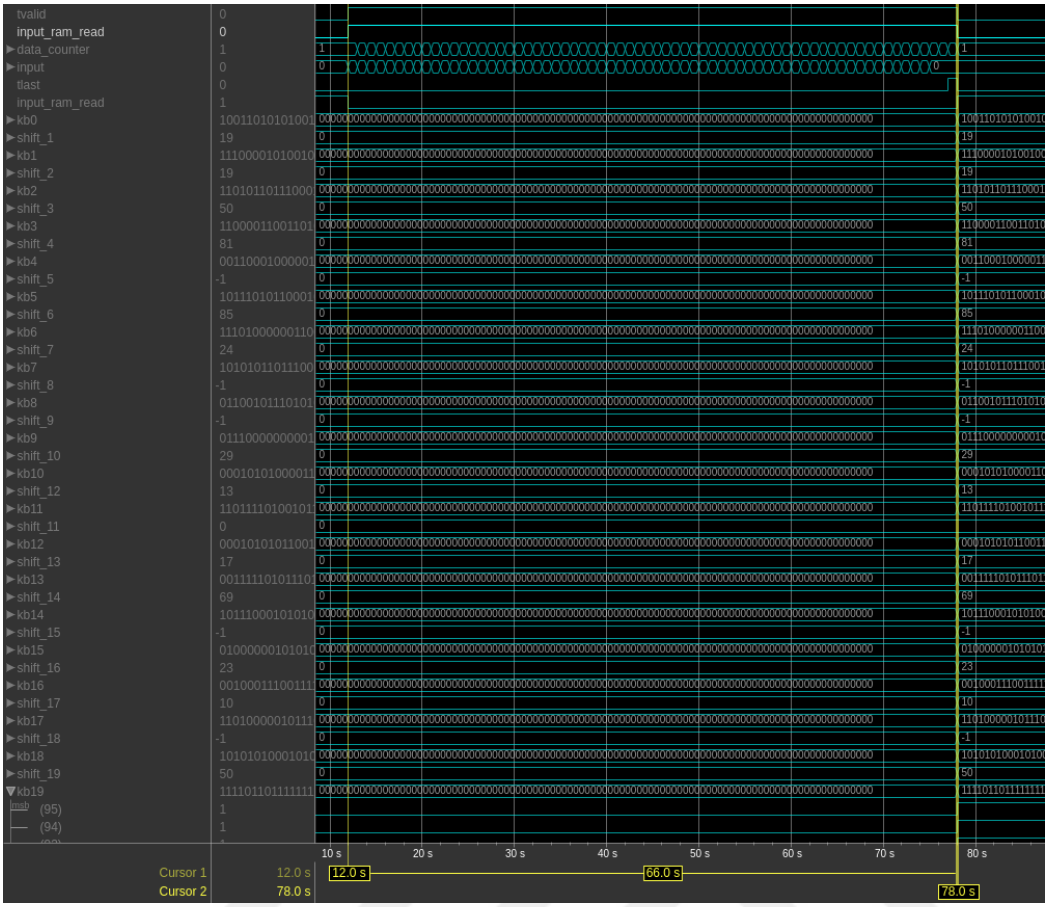
Model tabanlı tasarım ortamında yapılan testlerde çıktılar logic analyzer ile gösterilmektedir. Tasarım axi stream transfer protokolünü sağlaması gerektiğinden gelen veriler parçalı olarak alınmaktadır. Veri transferinde AXI Stream protokolü kullanıldığı için tvalid, tlast ve tdata sinyalleri kullanılmaktadır [26]. Gelen veriler, AXI Stream protokölünden dolayı her bir transfer süresinde bir veri RAM alınana yazılacak şekilde yazma işlemleri gerçekleştirilmektedir.



Şekil 4.25 : Giriş verilerin RAM'e yazılma işlemi.

Örnek olarak Z_c değeri 96 olan Temel Matris 1'in kullanıldığı bir sistem gösterilecektir. Şekil 4.25'da giriş verilerin tvalid olduğu süre boyunca RAM'e yazıldığı gösterilmektedir. Gelen verilerin RAM alanına yazılma işlemi sonunda paralel bir şekilde okuma yapılabilemketedir. Şekil 4.26'de gösterildiği gibi RAM'e sıra ile yazılan veriler parallel bir şekilde okunabildiği ve her bir okunan elemanın boyutunun Z_c kadar olduğu gösterilmektedir.

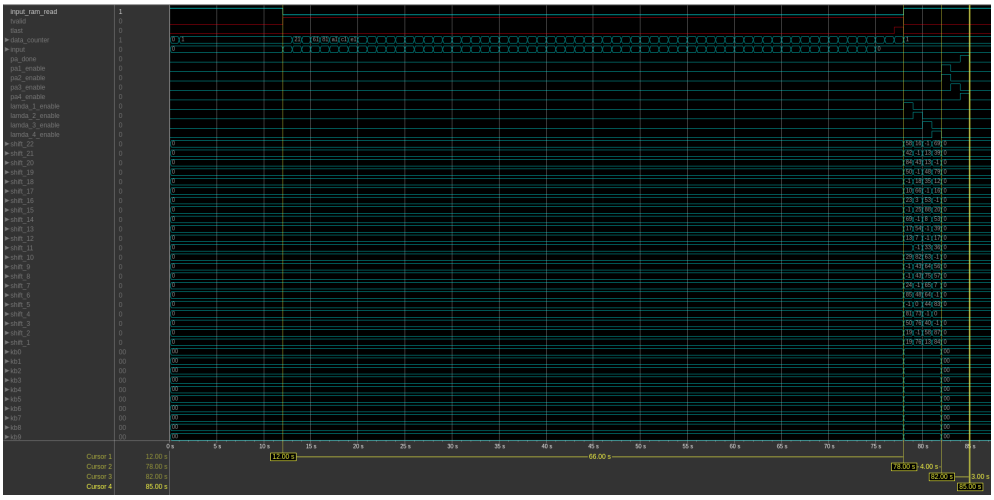
Parallel okuma yapıldıktan sonra A matrisi elemanları ile parallel bir şekilde varil kaydırma bloğuna işleme sokulmaktadır. Kaydırma bloklarının çıktıları Mod2 toplama (XOR) bloğu ile işleme sokulmaktadır. Daha sonrasında kaydırılmış ve toplama işlemi yapılmış Z_c bit dizisi geçici yazmaçlara yazılmaktadır. Şekil 4.27'da varil kaydırma ve toplama sonucunda oluşan 96 bitlik serisinin yazmaça yazılması gösterilmektedir. Çekirdek parite vektörlerin hesabı için A matrisi boyunca bu işlemin tekrarlanması gerekmektedir. Şekil 4.27'da dört satır boyunca kaydırma ve toplamı işlemi yapılmasının sonucu gösterilmektedir. Bölüm 2.5'da belirtildiği gibi dört satırın toplanması ile p_{a_1} ve p_{a_2} vektörleri hesaplanmaktadır. D matrisinin çift köşegen olma özelliği kullanılarak diğer parite vektörleri p_{a_3} ve p_{a_4} hesaplanması yapılarak yazmaçlara kaydedilmektedir. Şekil 4.27'da p_a vektörlerinin hesaplanarak yazmaçlara yazıldığı gösterilmektedir. Şekil 4.27'de üstte ilk olarak giriş verisinin alınışı gösterilmektedir. Giriş verisinin alınmasından hemen sonra parallel olarak okunan datalar kb0, kb1, ... gibi gösterilmektedir. A matrisi içerisinde olan kaydırma bilgileri



Şekil 4.26 : RAM'e sırayla yazma ve parallel okuma işlemi.

shift olarak gösterilmektedirler. Her bir satır parallel olarak varil kaydırma ve toplama işlemine girdikten sonra lamda yazmaçlarında kayıt altına alınmaktadır. Yazmaça yazılmasının nedeni hızlı bir şekilde okunabilmesidir. Her bir yazmaça yazılabilmesi ilgi döngüde olabilmektedir. Lamda yazmaçlarına yazılan veriler Denklem **2.16**'de gösterildiği gibi parallel olarak toplandığında p_{a_1} parite vektörü hesaplanabilmektedir. Diğer çekirdek parite vektörleride Denklem **2.17**, **2.18** ve **2.19**'de gösterilmektedir. Giriş verilerinin sisteme alınma işleminin tamamlanmasından sonra toplam yedi çevrim süresinde dört parite vektörün hesaplamalarının tamamlandığı gösterilmiştir.

Model Composer ile yapılan testler sonrasında alınan veriler ile MATLAB betiğinin sonuçları karşılaştırıldığında aynı olduğu gözlemlenmiştir. Kodlama yapılmış verileri AWGN kanaldan geçirdikten sonra MATLAB'ın nrLDPCDecode betiği ile test edildiğinde MATLAB kodlayıcı bloğu ile aynı paternde kodları çözebildiği gözlemlenmiştir.



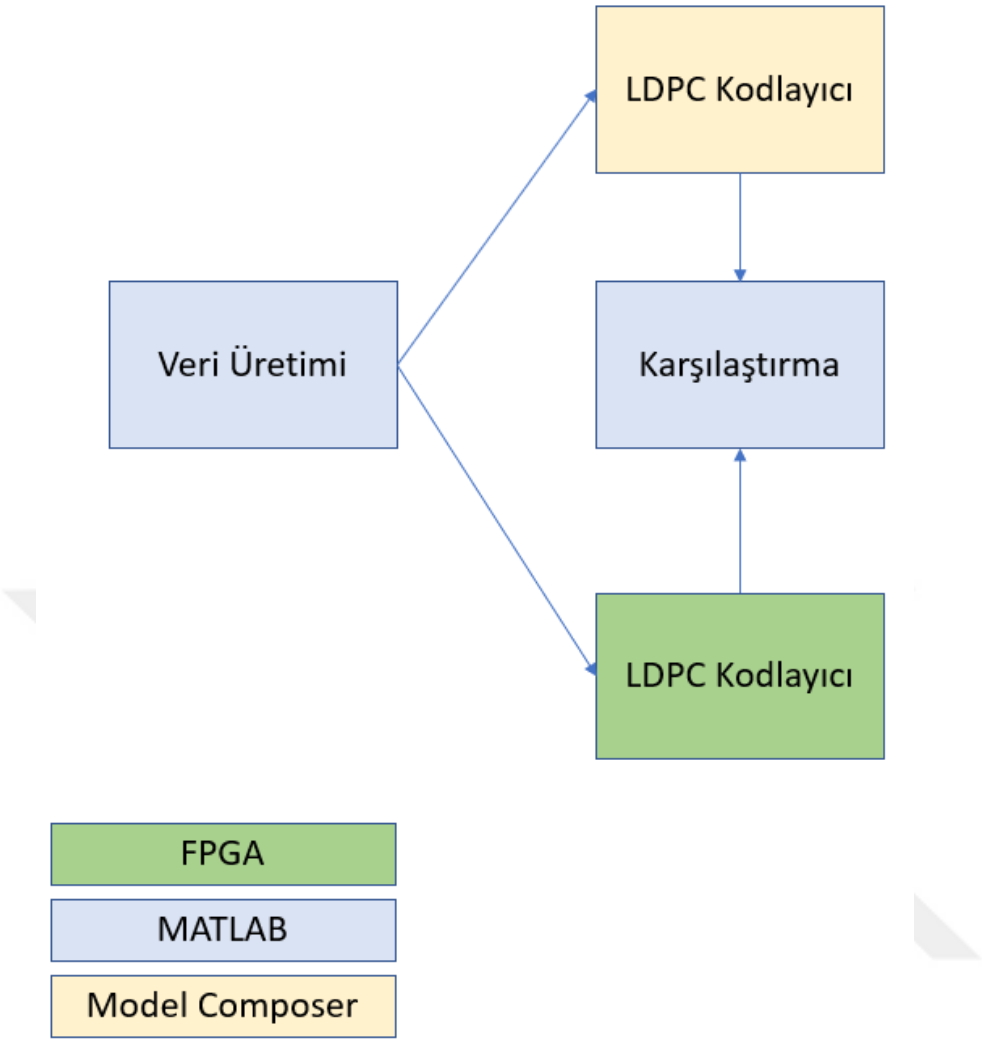
Şekil 4.27 : p_a vektörlerinin hesaplanması.

İkinci test senaryosunda Model Composer ile IP üreterek FPGA sentezi gerçekleştirilmesi gerekmektedir. Şekil 4.28'de FPGA ile test senaryosu gösterilmektedir. Bu senaryoda DMA ile verilerin toplanması gerekmektedir. Sistemden toplanan veriler MATLAB ortamında incelendiğinde MATLAB betiği ile aynı sonuçlar gözlemlendi. Kod çözdürme testleri olmamasının nedeni ilk test senaryosunda sistemde başarılı bir şekilde kodların çözülebildiği gözlemlenmesidir.

4.2.5 QC-LDPC kodlayıcı blok IP üretimi

Tasarlanan LDPC kodlayıcı bloğunun similasyon testleri tamamlandıktan sonra Vivado IDE'si üzerinden sentez yapılıp karta atılması işlemleri yapılacak. Model tabanlı olarak üretilen IP'nin donanım tanımla dillerinden olan Verilog diline dönüştürme işlemini Xilinx'in Model Composer ToolBox [17] uygulması gerçekleştirmektedir. Tasarımın top modlüne bir tane Xilinx'in Vitis Model Composer Hub bloğundan konulması gerekmektedir. Şekil 4.29'de test ortamında en üst modüle Model Composer Hub ekleniyor.

Model Composer Hub içerisinde ilk olarak Target sekmesinden hangi FPGA için üretileceğinin seçilmesi gerekmektedir. Hedef FPGA kartı yada çipi seçildikten sonra hangi tip tasarım yapıldıysa ona uygun bir metot seçilmesi gerekmektedir. Bizim tasarımımız HLS kullanıldığı için HLS sekmesinden hangi bloğun kodunun



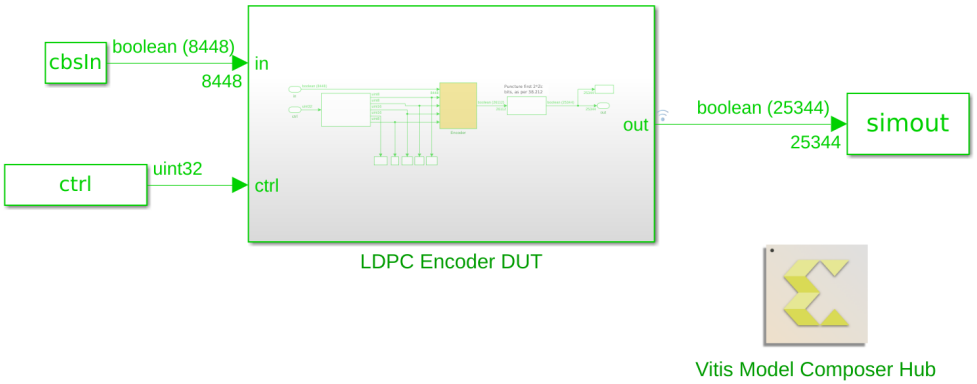
Şekil 4.28 : LDPC kodlayıcı blok ikinci test senaryosu.

üreteceğimizi seçmemiz gerekiyor. Daha sonrasında hedef saat frekansı ve throuhput faktörü Şekil A.3'da gösterildiği gibi seçildikten sonra IP üretimi yapılabilmektedir.

IP üretimi için ayarlar yapıldıktan sonra Generate butonuna basarak kod üretimine başlatabiliyoruz. Şekil A.4'de görüldüğü gibi kod üretimine başladığı zaman verisiyon kontrolleri, hedef FPGA özelliklerini kontrol ettikten sonra IP üretimine başlıyor. Kod üretimi bittiğinde yeşil bir zemine üzerine DONE yazarak tamamlanmaktadır.

4.2.6 Vivado IDE üzerinde LDPC kodlayıcı tasarım sentezi

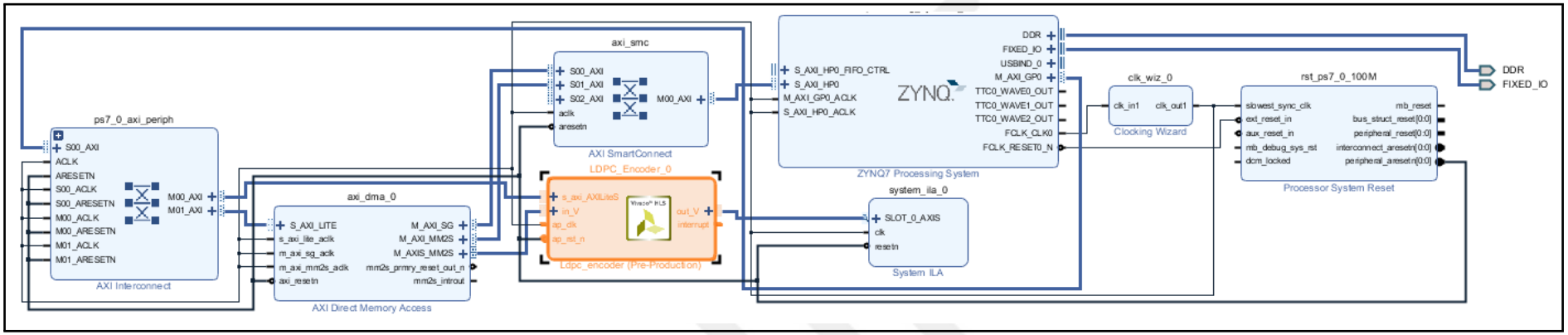
Model Composer üzerinde yapılan tasarım testlerden geçtikten sonra IP üretilmesi işlemine geçiliyor. Üretilen IP Vivado IDE üzerinde sentezlenerek FPGA kartına



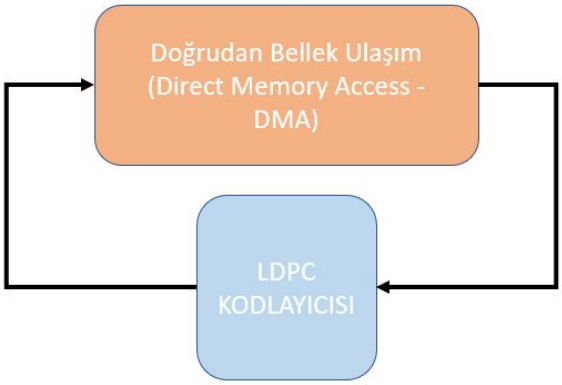
Şekil 4.29 : Vitis Model Compopser Hub eklenmesi.

uygun yük çıkarılıyor. Vivado tasarımı Şekil 4.30'da gösterilmektedir. Kart üzerinde test için DMA (Doğrudan Bellek Erişimi - Direct Memory Access) bloğu kullanıldı. Sisteme verilen DMA IP ise ile iletilmektedir.

Şekil 4.31'de gösterildiği gibi DMA ile testler yapılmaktadır. Sistem çıışını izlemek için Entegre Mantık Analizörü (Integrrated Logic Analyzer - ILA) kullanıldı. Sisteme DMA ile $n - m \times Z_c$ bit boyutunda girdi verildiğinde $n \times Z_c$ bit boyutunda çıktı gözlemlendi. Model Composer ile aynı testler yapıldı. Yükseltgenme 96 olan ve temel matris 1 ile yapılan test Şekil 4.33'de gösterilmektedir. Bu testte giriş ve çıkışlar Model Composer ile aynı olarak çıkmaktadır ve aynı zaman sürmektedir. Yapılan testte Model Composer ile aynı sonuçları aldığımız için Model Composer ile yapılan testlerin donanımsal testlere eş değer olduğu söylenebilmektedir. Yükseltgenme faktörü 384 olan test sonucu alınan ila logları EK A.2'de paylaşılmaktadır. Burada 8448 bitlik dizi 32 bit olarak 264 örnekte sistem içine alınmaktadır. Sistem içerisine alınmasına devam edilirken aynı zamanda ilk giren kısım sistem dışına verilmesine başlanarak giriş çıkış arasındaki geçikme azaltılması amaçlanmıştır. Sistem çıkışına verilmeden önce 7 örnek beklenmektedir. Bu süre p_a vektörlerinin sistem içinde hesaplanma süresine eşit olmaktadır. Sistemden 25344 bitlik dizi 792 örnek ile çıkmaktadır. Figure B.2'de yükseltgenme faktörü 384 olan örnek test sonucu gösterilmektedir.



Şekil 4.30 : Vivado blok dizayn.

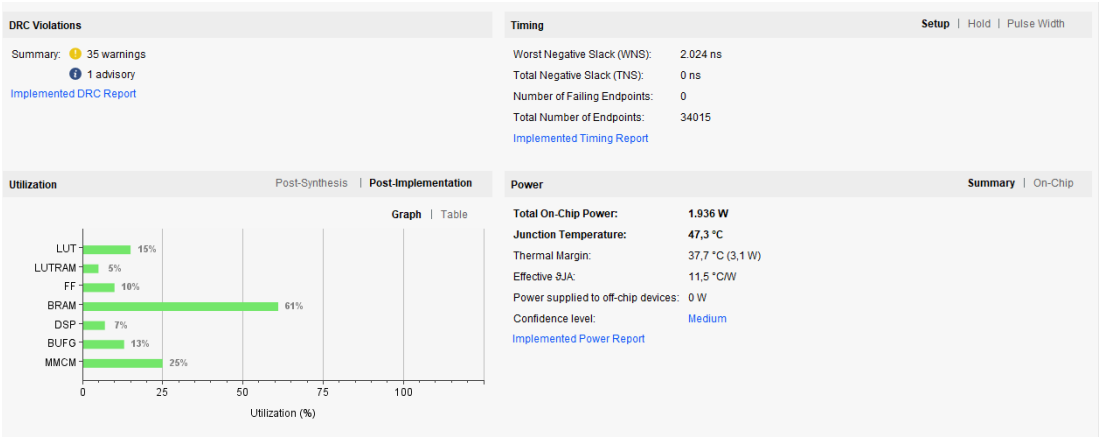


Şekil 4.31 : LDPC ve DMA sisteminin blok diagramı.

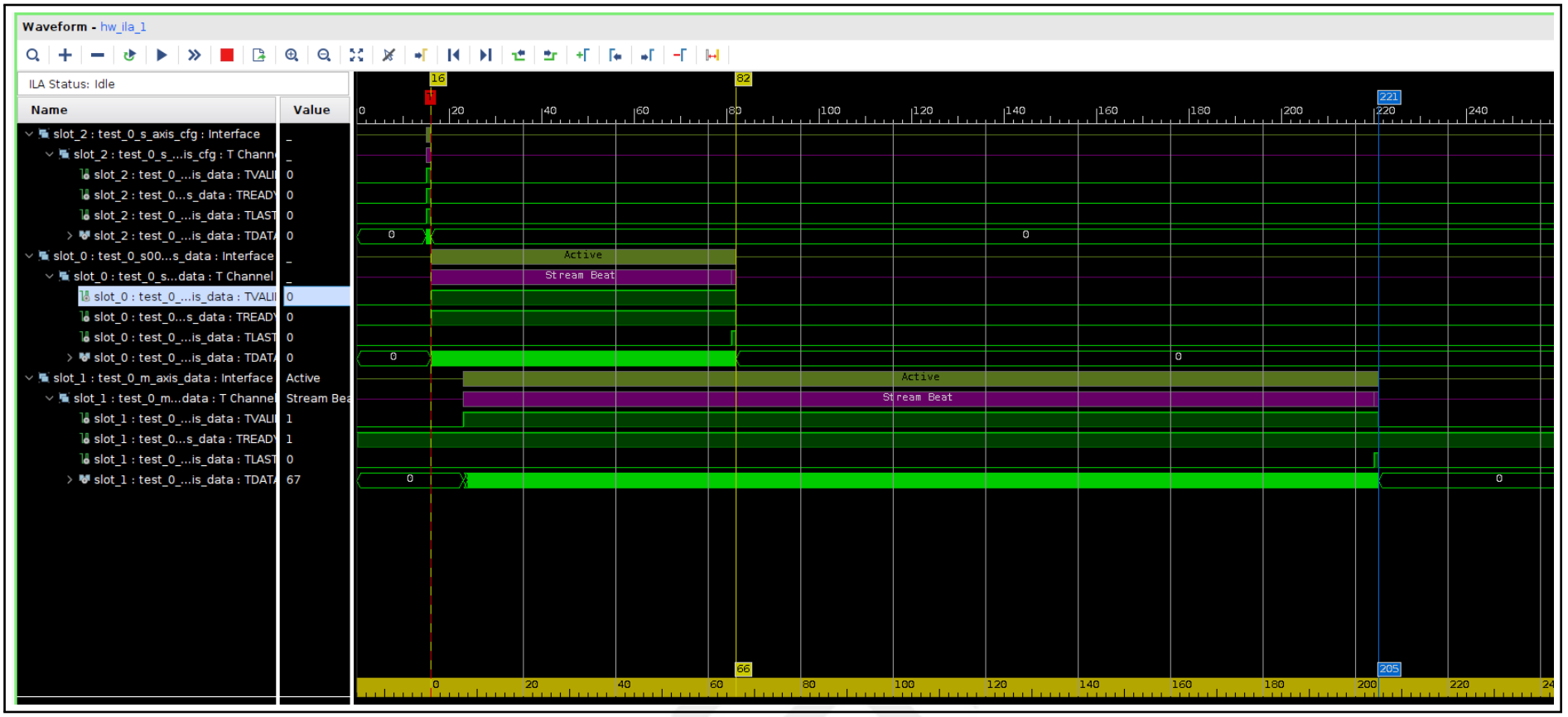
Donanım üzerinde süre bakımından aynı olduğunu gözlemlediğimiz testlerin detaylı bir şekilde incelemek için DMA ile kodlanmış verileri işlemciye çıkararak yedekleme işlemi yapıldı. Yedeklenen donanım çıktıları MATLAB içerisinde incelendiğinde Model Composer ve FPGA çıktılarının aynı olduğu gözlemlenmiştir.

Üç testtede Model Composer ile aynı sonuçları aldığımız için Model Composer ile yapılan testlerin donanımsal testlere eş değer olduğu söylenebilmektedir.

Tasarımın kullandığı kaynak ve güç kullanım değerleri Şekil 4.32'de görülmektedir. Bram kullanımın çok yüksek olmasının nedeni 32K'lık ILA kullanımından kaynaklanmaktadır. Sistemden ILA ve DMA çıkarıldığında sadece LDPC Kodlayıcı bloğunun kaynak tüketimi 4.2'de gösterilmektedir.



Şekil 4.32 : Tasarımın kaynak tüketimi ve güç kullanımı.



Şekil 4.33 : Yükseltgenme faktörü 96 ile yapılan test sonucu ILA gösterimi.

Daha önceki yapılan çalışmalar ile karşılaştırıldığında satır paralel algoritma gerçeklemesi olan [19]'dan saat frekans ve kaynak tüketimi açısından daha etkili bir tasarım gerçeklemesi yapılmıştır. Ancak [24]'in yapmış olduğu çalışma mevcut gerçeklemeden kaynak tüketimi açısından daha etkili bir çalışma olmaktadır. Block RAM tüketiminin az olmasının ve LUT ve FF miktarının fazla olmasının nedeni içerde kullanılan RAM yapılarının BRAM kullanılmadan gerçeklenmesinden kaynaklanmaktadır. Ancak saat frekansları gerçekleme yapılan FPGA boardlarının sınırlarına geldiğinden benzerlik göstermektedir.

Çizelge 4.2 : Model Composer ile yapılan tasarımın kaynak tüketim karşılaştırması

Tasarım	Kaynak Tüketimleri			
	Flip Flop (FF)	Look Up Table (LUT)	36K BRAM	f_{max}
[19]	131185	126986	3.5	470
[24]	9635	11156	5	580
Gerçeklenen Tasarım	18962	24094	2	540

5. XİLİNX AI ENGİNE İLE QC-LDPC KODLAYICI BLOĞU TASARIMI

Bu bölümde bir önceki bölümde Model Composer ile gerçeklemesi yapılan QC-LDPC kodlayıcı bloğunun Xilinx AI Engine ile gerçeklemesi yapılmıştır. Fonksiyolite olarak iki tasarımda aynıdır. Ancak gerçeklendiği ortamdan kaynaklı hız ve kaynak tüketim farkları oluşmaktadır.

5.1 AI Engine kernel yapıları

AI Engine ile yapılan gerçeklemelerde sistem ilk olarak 4 parçaya ayrıştırıldı. AI Engine blokları içerisinde hafıza için çok büyük alan olmamasından dolayı LDPC tabloları sistem içende saklanamamaktadır. LDPC tablolarını yine PL (Programabilir logik) alanında saklanmasının sistemin etkinliği için daha iyi olacağı kararlaştırıldı.

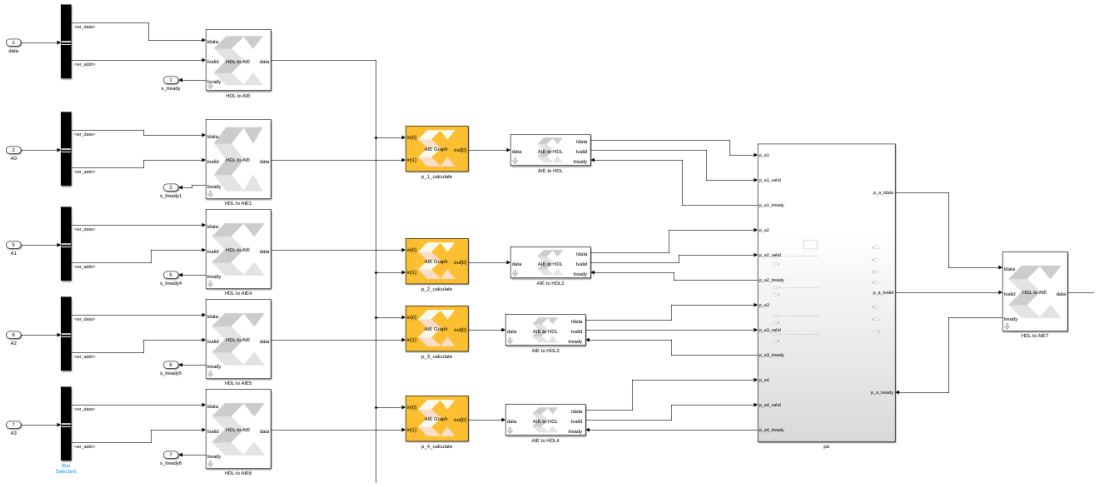
İlk olarak AI Engine içerisine LDPC tablosu ve sistematik bitleri verecek blok tasarımı yapıldı. Bu blok konfigürasyon geldikten sonra tablonun belirlenmesi ve tablo ile beraber sistematik bitleri AI Engine bloğuna vermektedir. Bu sayede AI Engine blokları içerisndeki fifoların etkili bir şekilde kullanılması sağlanmaktadır. Bu blok tasarımı Model Composer ile yapılan tasarımdan alındı. Bu sayede model tabanlı tasarım ortamının sağlamış olduğu pratiklik ile hız sağlanmış oldu.

LDPC tablosu ve sistematik bitler sistem içerisine alındıktan sonra paralel olarak kontrol bitlerinin hesaplanması işlemlerine başlanmaktadır. İlk olarak LDPC tablosundan Z_c (Yükseltme Faktörü) ile mod işlemi yapılmış elemanlar kadar kod blokların ötelenmesi yapılmaktadır. Ötelenme işlemi sonrasında ötlenemiş 22 tane kod bloğun toplanması yapılmaktadır. Toplama işleminde daha önceki bölümde XOR ile yapılan kısım yerine mod2 işlemi yapılmaktadır. AI Engine SIMD ve VLIW mimarisi sayesinde bu işlemler paralel bir şekilde yapılabilmektedir. Mod alma işleminin intrinsic hali olmadığı için çıkarma ve karşılaştırma işlemleri kullanıldı. Bu işlemler için

- FPGE (Vektör olarak karşılaştırma işlemi (büyük ve eşittir))
- FPSUB (Vektör olarak çıkarma işlemi)

komutları kullanıldı. Bu komutlar ile paralel olarak mod alma işlemi yapılabilmektedir.

Bu işlemlerle ilk olarak Ön Bilgiler bölümünde belirtildiği gibi, p_1, p_2, p_3 ve p_4 vektörlerinin hesaplanması gerekmektedir. Bu dört değer Z_c boyutunda vektörlerdir. Ancak hesaplanmaları birbirlerine bağlı oldukları için sırayla hesaplanmaları gerekmektedir. Bu yüzden dört satırın parallel olarak kaydırma ve toplama işlemleri yapılarak $\lambda_1, \lambda_2, \lambda_3$ ve λ_4 değerleri hesaplanmaktadır. Şekil 5.1'de blok diagram olarak gösterilmektedir.

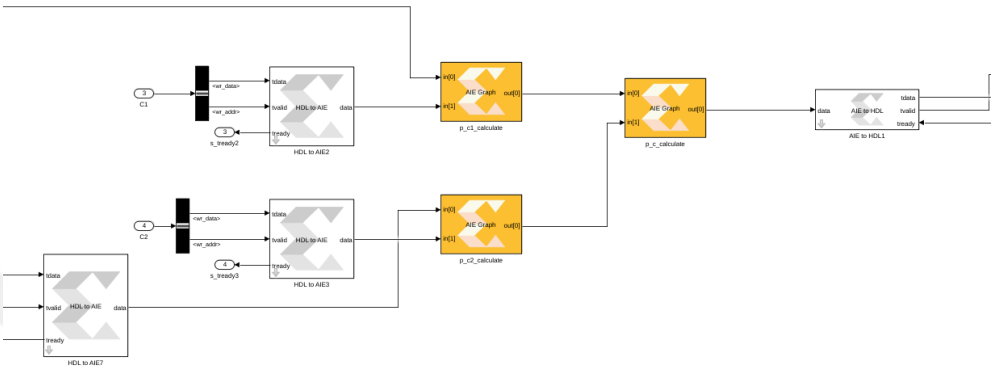


Şekil 5.1 : AI Engine ile LDPC kodlayıcı blok P_a vektör hesaplama tasarımı.

Bölüm 2.5'de bulunan Denklem **2.15**'da belirtilen λ değerleri hesaplandıktan sonra elde edilen değeleri AI Kernel'inden PL bölgesine alınarak p_1, p_2, p_3 ve p_4 vektörleri hesaplanmaktadır. Burada AI Engine kısmından PL kısmına geçiş yapılmasının iki ana nedeni bulunmaktadır. İlk olarak sadece parallel olarak dört vektörün toplanması ve Denklem **2.16**'den Denlem **2.19**'e kadar olan toplamların yapılmasıdır. AI Graph'larında iki port girişi olduğundan dört vektör için birden fazla AI Graph tanımı ihtiyacı olmaktadır. İkinci olarak bu yapının bir önceki yapılan tasarımda mevcut olması ve o tasarımdan bu tasarıma rahatlıkla taşınabilmesidir.

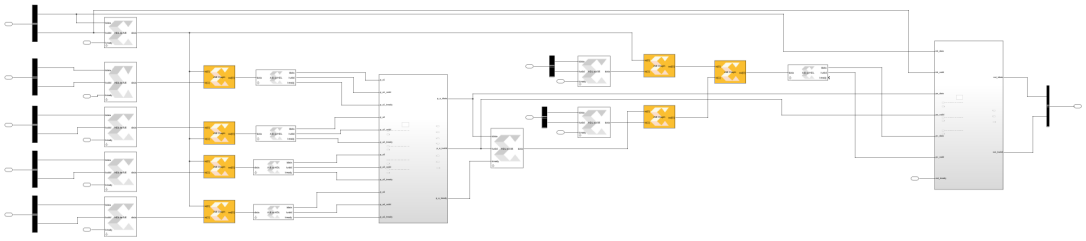
İlk dört parite kontrol bit vektörleri p_1, p_2, p_3 ve p_4 hesaplandıktan sonra diğer parite bit vektörleri paralel olarak hesaplanabilmektedir. AI Engine hesaplamalarında

XOR işlemi ile mod2 olarak toplamak yerine kümülatif olarak toplama işlemi yapılamadığından dolayı PL bölgesinde P_a vektör hesaplamından sonra mod2 işlemide PL bölgeisnde yapılmaktadır. Son basamak olarak mod2 işlemi ile sistemden bit olarak çıkması sağlandı. Şekil 5.1'de LDPC kodlama bloğunda bulunan P_a vektör hesaplanmasının ana blok diagramı gösterilmektedir. LDPC tablosunun üretimi Bölüm 4.2.2'de Model Composer ile yapılan tasarımda gerçeklenen kullanılmıştır.



Şekil 5.2 : AI Engine ile LDPC kodlayıcı blok P_c vektör hesaplama tasarımı.

LDPC Kodlama bloğu içerisinde p_a vektör hesaplamarı tamamlandıktan sonra p_c vektörlerinin hesaplanma işlemlerine başlanmaktadır. Şekil 5.2'da p_c vektörlerinin hesaplanması gösterilmektedir. Burada ilk olarak C_1 alt matrisi ile giriş dataları işleme sokuluyor. Alt matrisin her biri satırı için parallel olarak giriş verileri matris elemanı kadar ötleme yapılarak parallel olarak toplama işlemi yapılıyor. Benzer şekilde C_2 alt matrisi ile p_a parite matrisi işleme sokuluyor. Bu işlemler sonrasında p_{c_1} ile p_{c_2} vektörlerinin parallel olarak toplanması ile nihai p_c vektör hesaplanamsı yapılmaktadır.



Şekil 5.3 : AI Engine ile LDPC kodlayıcı blok tasarımı.

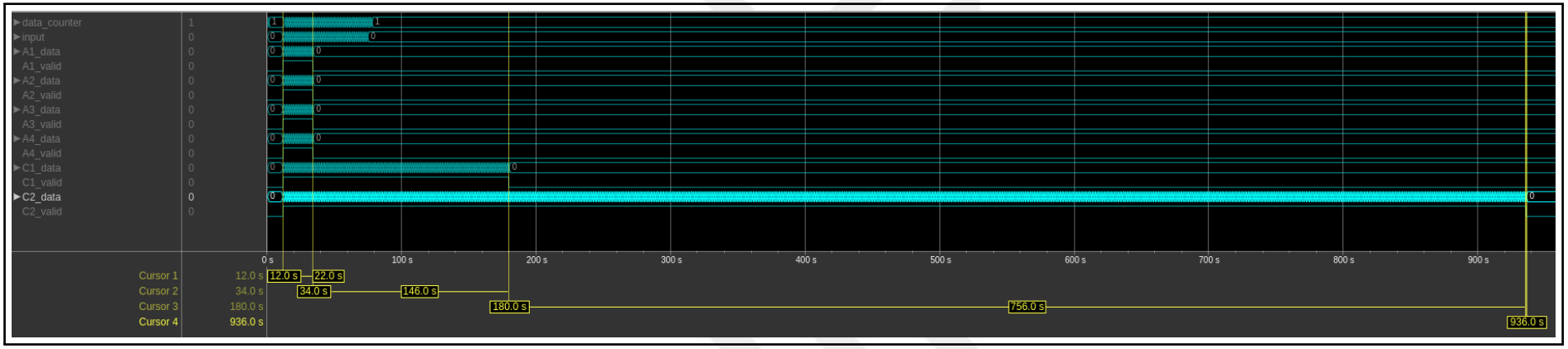
Şekil 5.3'da LDPC Kodlayıcı blok tasarımı gösterilmektedir. Hesaplanan p_a ve p_c parite vektörlerinin giriş datalarının peşine eklenmesiyle çıkış verileri elde

edilmektedir. Çıkış verilerinin hesaplanması için blok tasarımı Bölüm 4.2.3'de kullanılan çıkış blok tasarımı kullanıldı. 3GPP Standardı [2] gereği ilk $2xZ_c$ boyutundaki datayı silerek blok dışına taşınması sağlanmıştır.

5.2 AI Engine QC-LDPC Kodlayıcı Bloğu Test Sonuçları

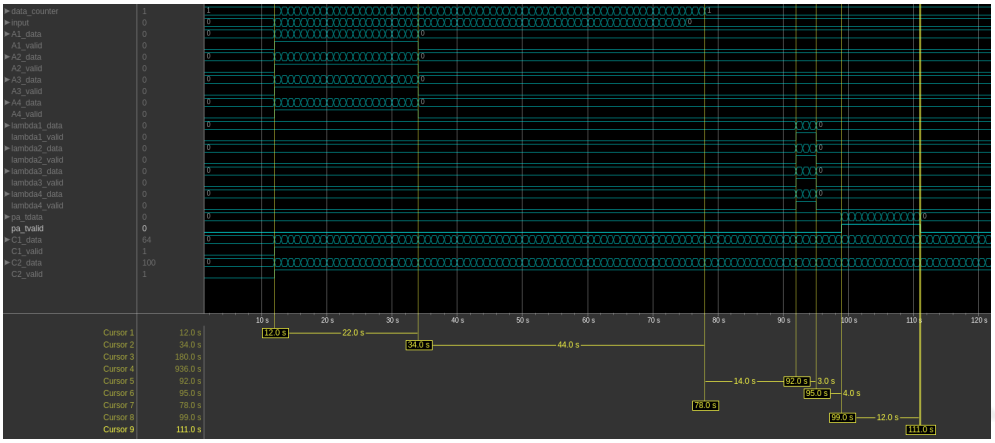
Tasarım testlerinde altn referans olarak MATLAB kullanıldı. MATLAB üzerinden rastgele bit serisi üreterek sisteme verildi ve çıkışındaki datalar matlab çalışma alanına kaydedildi. Paralelde MATLAB üzerinden üretilen rastgele bit serisi MATLAB içerisnde bulunan nrLDPCEncode betiğine verildi. İki çıktı matlab betiği ile karşılaştırıldı ve aynı olduğu gözlemlendi. Tasarımda kod çözücü tasarımı yapılmadığı için MATLAB'in nrLDPCDecode betiği kullanıldı. AI Engine çıktıları nrLDPCDecode betiğine verildiğinde çıktılar ile ilk üretilen datalar karşılatırıldığında aynı olduğu gözlemlendi.

Model tabanlı olarak ilk testler giriş verileri ile başladı. Giriş verilerin sisteme alınması 5.4'da gösterildiği gibi alınmaktadır. Daha önceki bölümdeki gibi $Z_c = 96$, temel matris 1 ile test gösterilmektedir. Bu test ortamında toplam 66 saple sürede kod blok verileri sisteme alınabilmektedir. Parallel olarak λ değerlerinin hesaplanması için A alt matrisinin satırları ayrı ayrı işleme sokulmaktadır. Bu sayede 88 sample sürede hesaplanmaya başlaması yerine 22 sample sürede λ değerlerinin hesaplanması yapılabilmektedir. Ancak C_1 alt matrisinin eleman sayısı 924 olduğundan dolayı nihai hesaplamalar 924 sample süre beklemektedir. AI Engine blokları belirtilen boyutta eleman gelmeden işleme başlamadığı, belirtilen elemandan fazla veri geldiğinde de fazla gelen veriyi almadığından dolayı belirtilen boyut kadar veri sisteme verilmelidir. Sistemi hızlandırmak için satırlara farklı işlem blokları oluşturabilmek mümkündür ancak kullanılacak AI Kernel sayısıda artmaktadır. Bu testte gösterildiği gibi sistemin çalışma süresini uzan işlem C_1 alt matrisinin boyutudur.



Şekil 5.4 : AI Engine LDPC kodlayıcı veri giriş testi.

Parite kontrol vektörlerinin hesaplanmasında ilk olarak p_a vektörlerinin hesaplanması işlemi yapılmaktadır. Şekil 5.5'de p_a hesaplanması logic analyzer çıktısı gösterilmektedir. Burada 66 sample süre boyunca kod blok verileri sisteme alındıktan sonra AI Kernel'inde öteleme ve toplama işlemleri parallel olarak yapılma işlemi 12 sample süre sürdüğü gösterilmektedir. Bu süre sonunda $lambda_1$, $lambda_2$, $lambda_3$ ve $lambda_4$ vektörlerinin hesaplandığı gösterilmektedir. *lambda* vektörleri hesaplanması bittikten sonra AI bölgesinde PL bölgesine geçiş yapılmaktadır. Burada 4 sample süre sonunda pa_1, pa_2, pa_3 ve pa_4 elemanları hesaplanması yapılarak $4 \times Z_c$ bit boyutunda p_a dizini oluşturmaktadır. Bu örnekte Z_c değeri 96 olduğu için 12 sample sürede tekrar AI bölgesine taşınması yapılamaktadır.



Şekil 5.5 : AI Engine LDPC kodlayıcı p_a vektörlerinin hesaplanması.

Kodlayıcı bloğunda p_a parite kontrol bit vektörlerinin hesaplanması tamamlandıktan p_c parite kontrol bit vektörlerinin hesaplanması adımına geçilmektedir. Bu aşamada iki tane alt matris C_1 ve C_2 kullanılmaktadır. Bu alt matrisler ile Denklem **2.20**'de gerçeklenmesi yapılmaktadır.

P_a vektörünün hesaplanması sonrasında P_c vekötrünün hesaplanması Şekil B.3'de logic analyzer logu ile gösterilmektedir. Burada ilk olarak p_a ile C_2 alt matrisi işleme sokulmaktadır. Bu işlem dört sample süre sonunda çıktı vermektedir. Burada çıktı boyutunun 126 sample sürmesinin nedeni 42 tane satır olması ve her satırda Z_c boyutunda vektör olmasından kaynaklanmaktadır. AI kernel'i girişinde belirtilen boyuttan daha az data geldiği süre boyunca çalışmayacağından dolayı C_1 alt matrisinin elemanlarının bitmesi beklenmektedir. C_1 alt matrisinin elemanlarıda sisteme alındıktan sonra p_{c_1} hesaplanması yapılmaktadır. 12 sample süre sonunda çıktı

vermekte olduğu gözükmektedir. Elde edilen p_{c_1} ve p_{c_2} vektörlerinin parallel olarak toplanması sounucu p_c parite kontrol bit vektöleri hesaplanmaktadır. Hesaplanan p_a ve p_c parite kontrol bitleri giriş bitlerinin sonuna eklenerek LDPC kodlaması yapılmaktadır. Sisteme giriş ve çıkış süresi 1315 sample süre sürmektedir.

MATLAB ortamında simülasyon aşamalarından sonra FPGA üzerinde testlere geçildi. Burada Bölüm 3'de gösterildiği gibi template Vivado tasarımı oluşturuldu. Tasarımda sistem içerisine dataların verilmesi için DMA eklemesi yapıldı. DMA, sisteme giriş verilmesi ve sistem çıkışındaki verilerin PS (İşlemci Alt Sistemi - Processing Subsystem) bölgesine taşınması için kullanılacaktır. Bu tasarım içerisinde kullanılacak kernel ve graph dosyaları eklendi, Bağlantı listesi (Connectivity List) verildi. Vitis Deign Flow adımları takip edildikten sonra elde edilen tasarım karta atıldı ve MATLAB testlerinde kullanılan test dataları ile beslendi tasarım. İşlem sonunda elde edilen sonuçlar işlemciye çıkarılarak dosyaya kaydedildi. Dosya MATLAB ortamında açıldığında similasyon sonuçları ile aynı olduğu gözlemlendi.

Xilinx AI Engine ile yapılan tasarımının kaynak tüketimi incelendiğinde model tabanlı tasarımdan farklı olduğu gözlemlendi. Bunun en büyük nedenlerinden biri AI Engine bloklarının PL'den bağımsız olarak farklı bir mimaride tasarlanmış olmasıdır. Model tabanlı yapılan tasarımda PL kaynaklarının kullanılmasına karşın AI Engine ile yapılan tasarımda sadece AI blokları kullanımı olmuştur. Ancak LDPC temel matrislerin hesaplanması ve sistem içine alınması için Model Tabanlı tasarım ile blok tasarımı eklenmiştir. Bu eklenen blok yüzünden PL kaynak kullanımı olmuştur. Ancak LDPC kodlarının üretimi AI engine bloklarında yapılmaktadır.

Çizelge 5.1 : Model tabanlı AI Engine ile yapılan tasarımın kaynak tüketim karşılaştırması

Tasarım	Kaynak Tüketimleri			
	Flip Flop (FF)	Look Up Table (LUT)	36K BRAM	f_{max}
[19]	131185	126986	3.5	470
[24]	9635	11156	5	580
Gerçeklenen Tasarım (Model Composer)	18962	24094	2	540
Gerçeklenen Tasarım (AI Engine)	5817	8164	1	500

Çizelge 5.1'de gösterildiği gibi model tabanlı yapılan tasarımda saat frekansı 540 MHz olmasına karşın AI Engine ile yapılan tasarımda 900 MHz gibi hızlara çıkılabilmiştir. Bunun olmasının nedeni FPGA yapısının içinde olmasına rağmen FPGA limitlerinden bağımsız SIMD ve VLIW mimarisinde çalışabilmesidir. AI Engin blokları maximum 1.2GHz hızlara çıkabilmektedir. Ancak AI Engine tile ile PL arasındaki limitasyondan dolayı 500 MHz'de veri trafiği yapılabilmektedir. Hesaplama işlemlerinin data akış işlemlerinden daha fazla olduğu tasarımlarda kullanılması kaynak tüketimi açısından daha kullanışlı olacağı düşünülmektedir.

6. SONUÇ

5G haberleşme sistemlerinde, veri aktarımının iyileştirilmesi ve hata ayıklama için kullanılan LDPC kodlarının model tabanlı olarak iki farklı ortamda gerçeklenmesi yapıldı. Yapılan test sonucunda sistem çıktılarında elde edilen sonuçlar beklendiği gibi olmuştur. Donanım tanımla dilleri olan Verilog ile yapılan tasarımlara göre daha hızlı tasarım ve test süresi ile tasarımcıya avantaj sağladığı gözlemlenmiştir. Tasarımlara başlanmadan önce Verilog, MATLAB HDL Coder ve Model Composer ile aynı fonksiyoneliteye sahip bir tasarım yapılmıştır. Kaynak tüketimi ve tasarım süresi kıyaslandığında Model Composer ile tasarım yapılmasına karar verilmiştir.

Tez çalışması kapsamında 5G NR standardına uygun olarak çalışabilen LDPC kodlayıcı tasarımı model tabanlı olarak Xilinx Model Composer ve Xilinx AI Engine ile yapılmıştır. Tasarlanan kodlayıcı bloğu Model Composer ve donanım üzerinde doğrulanmıştır. Model tabanlı olarak tasarlamanın avantajları kullanılarak MATLAB üzerinde tasarım incelemeleri ve MATLAB referans betikleri ile karşılaştırmalar yapılabildi.

5G Yeni Radyo için kullanılan LDPC kodlayıcısı hakkında bilgiler edinilmiştir. Farklı kodlayıcı blokları arasında performans analizleri yapılmıştır ve tasarım kompleksitesileri incelenmiştir. Gerçeklemeye satır tabanlı parallel algoritma seçilmiştir. Hem satır olarak parallel olması hemde gerçeklenecek tasarım karmaşıklığı incelendiğinde en uygun algoritma olmuştur. Sütun parallel algoritmasının kaynak tüketimi incelendiğinde FPGA üzerinde çalışmaya satır parallel algoritma kadar uygun olmadığına karar verildi.

Tasarım aşamasında sadece LDPC kodlayıcı bloğunun çekirdek kısmı yapıldığı için üretilen test verilerinde segmentleme işlemlerinin yapılarak sisteme verilmesi gerekmektedir. 5G taşıyıcı blok tasarımı içerisinde kullanılabilecek bir tasarım olmuştur. 5G taşıyıcı blok içerisinde LDPC kodlama öncesinde CRC eklenmesi ve segmentlere ayrılması işlemleri yapılması gerekmektedir. Ayrıca LDPC kodlama bloğu sonrasında

oran uyumlandırıcı ve kod blokları birleştirecek tasarımların eklenmesi gerekmektedir. Ancak LDPC kodlayıcı bloğu tek başına kullanılabilmektedir.

Bu çalışmada sistem 32 bit boyutunda giriş ve çıkış portu oluşturulduğundan sisteme veri girişi ve veri çıkışı uzun sürmektedir. Giriş port boyutu büyütülerek sistemin giriş çıkış süresi azaltılabilmektedir.

Çalışmalar sonucunda tasarlanan LDPC kodlayıcı bloğu geliştirmeye açıktır. Kullanılmayan matris elemanlarının daha etkili bir şekilde kullanılması yada yerlerlerinin değiştirilerek daha küçük boyutta matrisin ROM'a yazılması sağlanarak kaynak tüketimi azaltılabilir, performans arttırılabilir. Bu geliştirmeler için donanım kullanma zorunluluğunu ortadan kaldırmaktadır. Tasarlanan model tabanlı yapının davranışı donanım üzerindeki ile aynı olduğundan yapılacak değişikliklerin testleri model tabanlı olarak devam edilebilmektedir.

Bu çalışmada,

- FPGA donanımına uygun çalışabilecek LDPC kodlama algoritma analileri yapılmıştır.
- 5G NR standartları ile uyumlu LDPC kodlayıcı blok tasarımı model tabanlı yapılmıştır.
- Model olarak tasarlanan yapı hem FPGA üzerinde hemde model tabanlı olarak MATLAB referans modeli ile karşılaştırmalı olarak test edilmiş ve doğrulanmıştır.
- QC-LDPC kodlayıcı blok tasarımı Xilinx AI Engine ile gerçeklenmiştir.
- FPGA ile AI Engine arasındaki kaynak tüketimi ve çalışma hızlı karşılaştırılmıştır.

Gelecek çalışmalarda,

- Giriş ve çıkış bit boyutlarının arttırılması,
- Temel matris 2'nin sisteme entegre edilmesi,
- Temel matrisin etkin bir şekilde kullanılabileceği tasarımın yapılarak kaynak tüketimin azaltılması yapılabilir.

KAYNAKLAR

[1] **David G. M. Mitchell, D.J.C.** (2014). Quasi-Cyclic LDPC Codes based on Pre-Lifted Protographs, *IEEE Transactions on Information Theory*, *60*, 5856 – 5874, https://doi.org/10.1109/TIT.2014.2342735.

[2] **3GPP** (2018). TS 38.212 version 15.2.0 Release 15 - 3GPP Standard 5G; NR; Multiplexing and Channel Coding, **Teknik Rapor**, 3GPP.

[3] **Xilinx** (2022). *Xilinx Versal ACAP VCK190 Evaluation Kit*, https://docs.xilinx.com/r/en-US/ug1366-vck190-eval-bd.

[4] **Xilinx** (2021). *Getting Started with Vitis*, https://docs.xilinx.com/r/2021.1-English/ug1393-vitis-application-acceleration/Getting-Started-with-Vitis.

[5] **Shannon, C.** (1956). The zero error capacity of a noisy channel, *IRE Transactions on Information Theory*, *2*, 10.1109/TIT.1956.1056798.

[6] **Gallager, R.** (1962). Low-density parity-check codes, *IEEE Transactions on Information Theory*, 21–28, https://doi.org/10.1109/tit.1962.1057683.

[7] **Xilinx** (2018). *Zynq UltraScale + Device Technical Reference Manual*.

[8] **Richardson, T. ve Kudekar, S.** (2018). Design of Low-Density Parity Check Codes for 5G New Radio, *IEEE Communications Magazine*, *56*, 28–34, https://doi.org/10.1109/mcom.2018.1700839.

[9] **C. Berrou, A. Glavieux, P.T.** (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes, *Proceedings of ICC '93 - IEEE International Conference on Communications*, IEEE.

[10] **D. Divsalar, S. Dolinar, C.J..K.A.** (2009). Capacity-approaching protograph codes, *IEEE Journal on Selected Areas in Communications*, *27*, 876–888, https://doi.org/10.1109/jsac.2009.090806.

[11] **Tanner, R.** (1981). A recursive approach to low complexity codes, *IEEE Transactions on Information Theorys*, *27*, 533–547, https://doi.org/10.1109/tit.1981.1056404.

[12] **Wang, H.W..H.** (2019). A High Throughput Implementation of QC-LDPC Codes for 5G NR, *IEEE Access*, *7*, 185373–185384, https://doi.org/10.1109/access.2019.2960839.

[13] **R. Goriushkin, e.a.** (2020). FPGA Implementation of LDPC Encoder Architecture for Wireless Communication Standards, *International Conference on Modern Circuits and Systems Technologies (MOCAST)*, Bremen, Germany, 10.1109/mocast49295.2020.9200293.

[14] **G. Tzimpragos, e.a.** (2013). A low-complexity implementation of QC-LDPC encoder in reconfigurable logic, *2013 23rd International Conference on Field programmable Logic and Applications)*, Porto, Portugal, 10.1109/FPL.2013.6645587.

[15] **Xilinx** (2019). *Vivado Design Suite User Guide : High-Level Synthesis*.

[16] **Richardson, T. ve Urbanke, R.** (2001). Efficient encoding of low-density parity-check codes, *Information Theory, IEEE Transactions*, *47*, 638–656, 10.1109/18.910579.

[17] **Xilinx** (2020). *Vitis Model Composer User Guide*.

[18] **Matworks** (2021). *MATLAB HDL Coder User Guide*.

[19] **T. Nguyen, T. Tan, H.L.** (2019). Efficient QC-LDPC Encoder for 5G New Radio, *Electronics*, *8*(6), https://www.mdpi.com/2079-9292/8/6/668.

[20] **J. Yunho, j.K.** (2010). Memory-efficient and high-speed LDPC encoder, *Electronics Letters*, *46*(14).

[21] **C. Huang, J. Huang, C.Y.** (2018). Construction of Quasi-Cyclic LDPC Codes from Quadratic Congruences, *IEEE Communications Letters*, *12*(4).

[22] **P. Zhang, C. Liu, L.J.** (2013). Efficient encoding of QC-LDPC codes based on rotate-left-accumulator circuits, *Electronics Letters*, *49*(13).

[23] **MATLAB** (2022). *MATLAB nrLDPCEncode*, "https://www.mathworks.com/help/5g/ref/nrldpcencode.html.

[24] **V. Petrović, D. El Mezeni, A.R.** (2021). Flexible 5G New Radio LDPC Encoder Optimized for High Hardware Usage Efficiency, *Electronics*, *10*(1106).

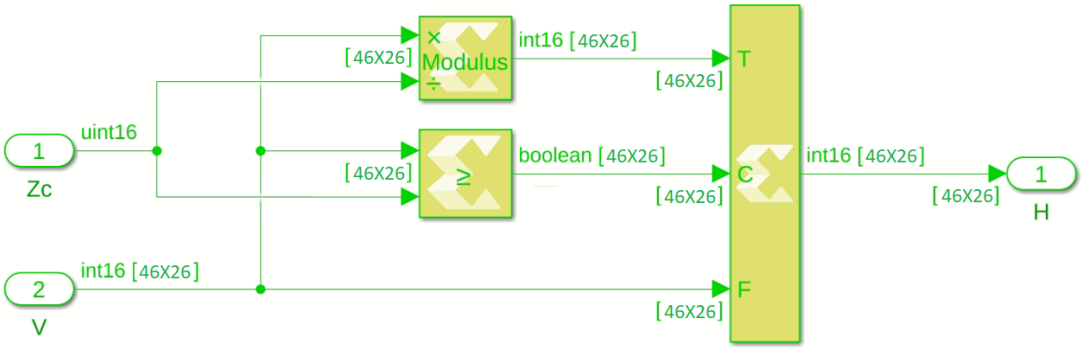
[25] (2009). Proc. 2012 International SoC Design Conference (ISOCC),

[26] **Xilinx** (2017). *AXI Reference Guide, v4.0*, https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.

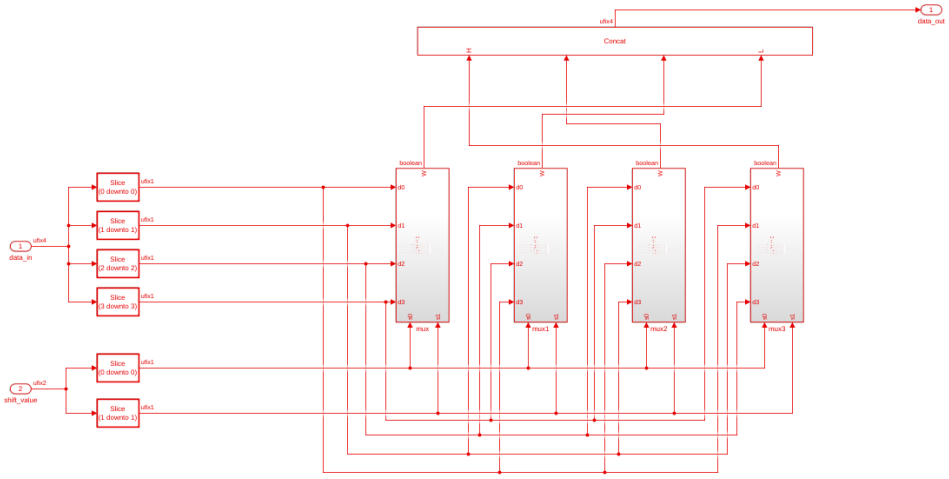
EKLER

EK A : Model tabanlı tasarım
EK B : Model tabanlı test

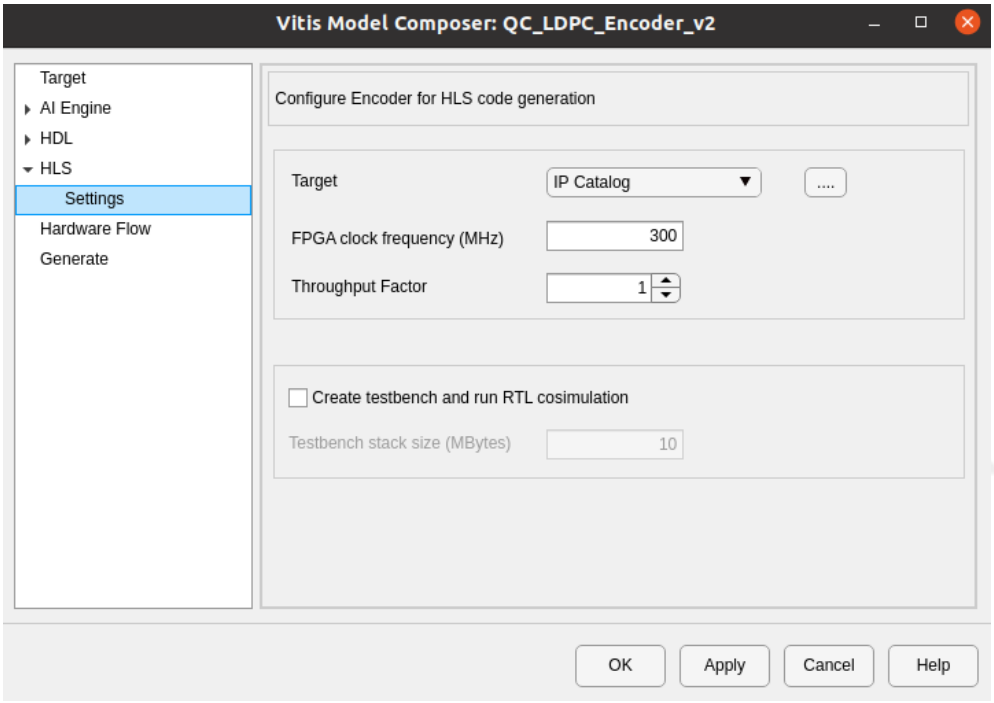
EK A : Model tabanlı tasarım



Şekil A.1 : Denklem **2.11**'nin gerçeklemesi.



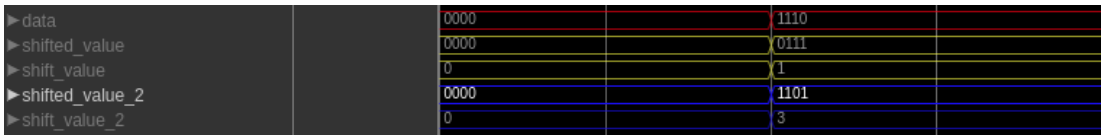
Şekil A.2 : Küçültülmüş bir kaydırma bloğu örneği.



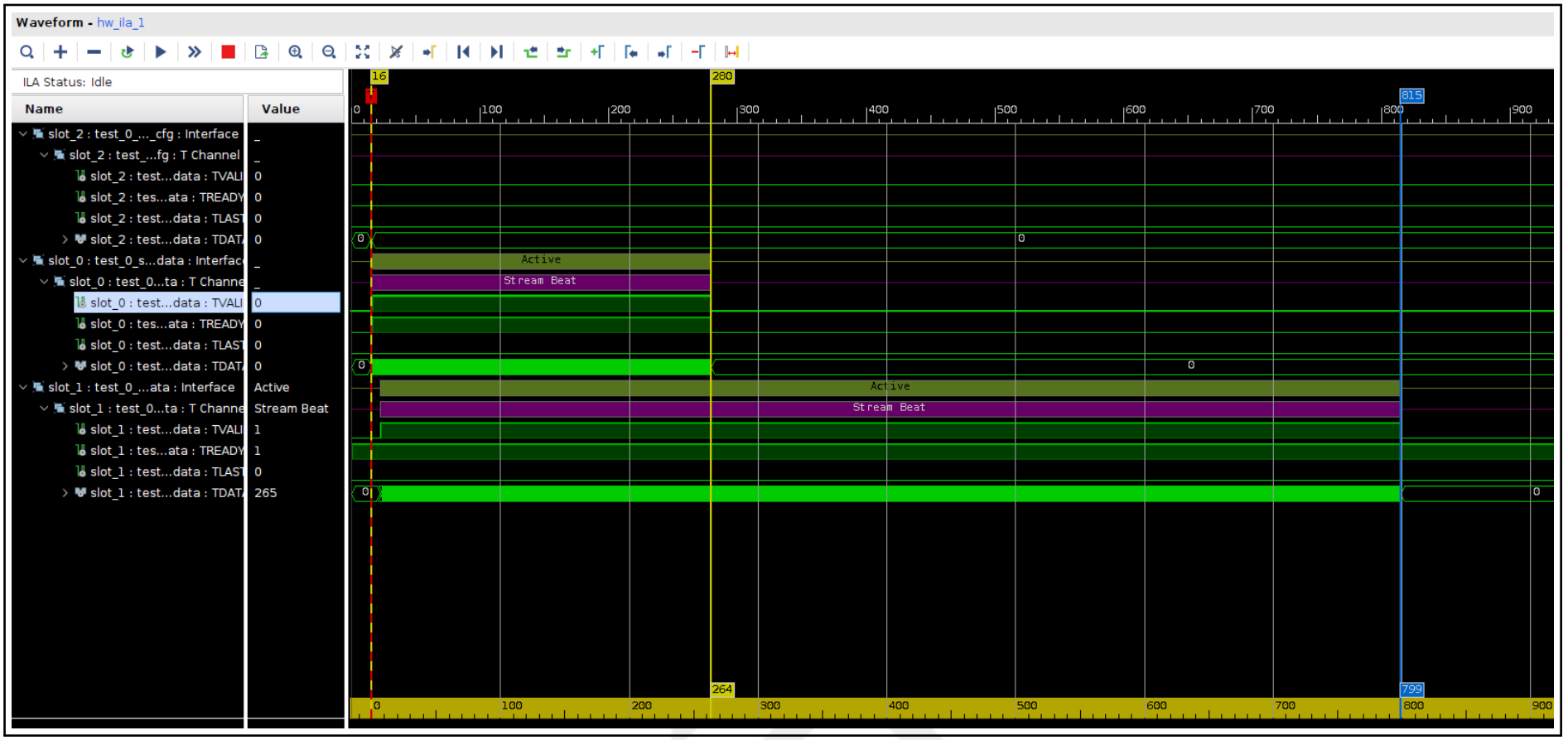
Şekil A.3 : Vitis Model Compopser Hub HLS ayarlar sekmesi.

Şekil A.4 : IP Üretimi ilk ekran.

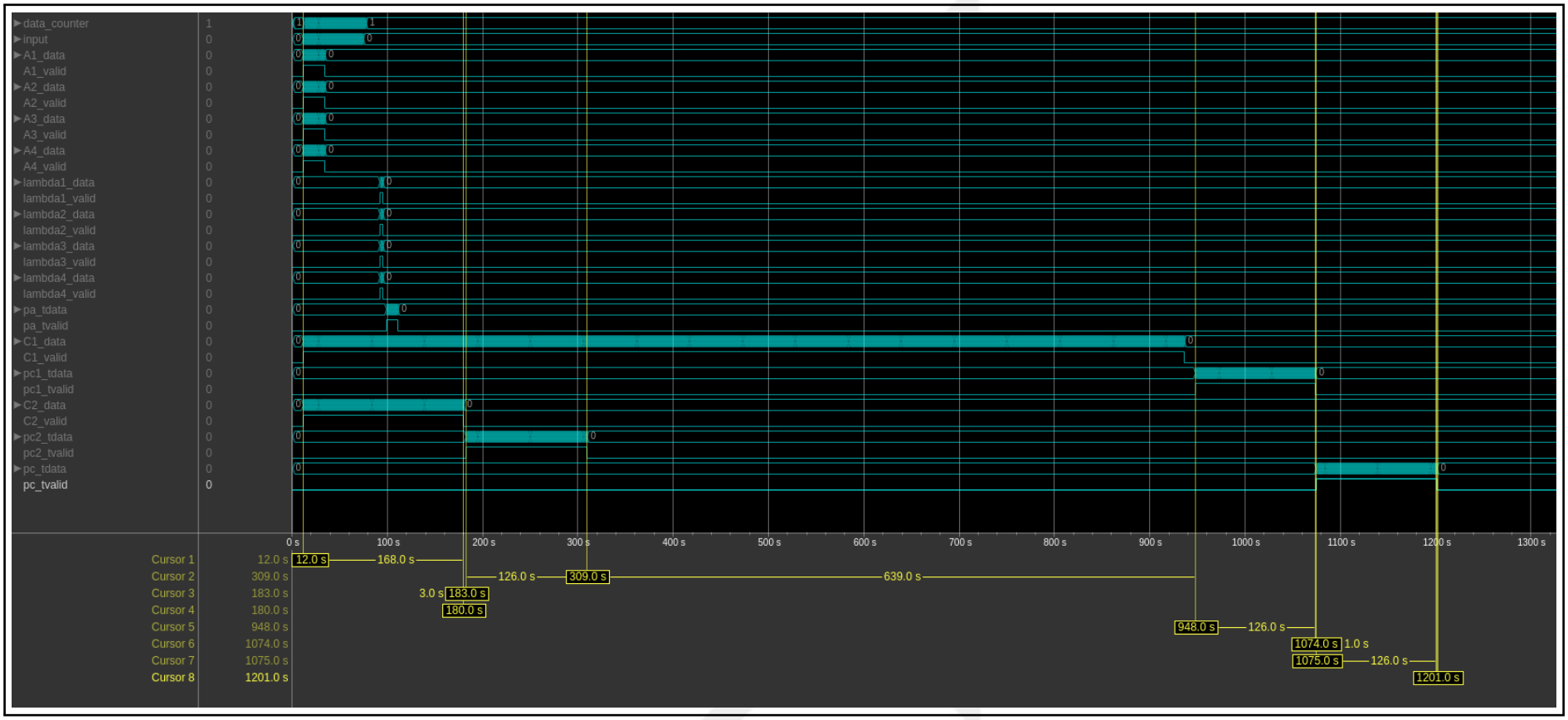
EK B : Model tabanlı test



Şekil B.1 : Dört bitlik kaydırma bloğunun test sonucu.



Şekil B.2 : Yükseltgenme faktörü 384 ile yapılan test ILA sonucu.



Şekil B.3 : AI Engine LDPC kodlayıcı p_c vektörlerinin hesaplanması.

ÖZGEÇMİŞ

ADI SOYADI: Hakan TAŞ

ÖĞRENİM DURUMU:

- Lisans: 2019, İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, Elektronik ve Haberleşme Mühendisligi Bölümü

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2020 yılından beri Ulak Haberleşme şirketinde sayısal tasarım mühendisi olarak olarak çalışıyor.

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- **H. Tas** ve B. Ors, "QC-LDPC Kodlayıcının Model Tabanlı Tasarım Yöntemi ile Etkinlik Analizi", Akıllı Sistemlerde Yenilikler ve Uygulamaları (ASYU) Konferansı 7-9 Eylül, 2022, Antalya, Türkiye.