İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

FPGA ÜZERİNDE 5G UYUMLU DÜŞÜK YOĞUNLUKLU EŞLİK DENETİM KOD ÇÖZÜCÜ GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Barış BİLGİLİ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

EYLÜL 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

FPGA ÜZERİNDE 5G UYUMLU DÜŞÜK YOĞUNLUKLU EŞLİK DENETİM KOD ÇÖZÜCÜ GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Barış BİLGİLİ
(504191203)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Sıddıka Berna Örs Yalçın
Eş Danışman: Prof. Dr. Ali Emre Pusane

EYLÜL 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

IMPLEMENTATION OF 5G COMPATIBLE LOW DENSITY PARITY CHECK DECODER ON FPGA

M.Sc. THESIS

Barış BİLGİLİ
(504191203)

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Sıddıka Berna Örs Yalçın
Co Advisor: Prof. Dr. Ali Emre Pusane

SEPTEMBER 2022

İTÜ, Lisansüstü Eğitim Enstitüsü'nün 504191203 numaralı Yüksek Lisans Öğrencisi Barış BİLGİLİ, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "FPGA ÜZERİNDE 5G UYUMLU DÜŞÜK YOĞUNLUKLU EŞLİK DENETİM KOD ÇÖZÜCÜ GERÇEKLENMESİ" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Sıddıka Berna Örs Yalçın** İstanbul Teknik Üniversitesi	.....
Eş Danışman :	**Prof. Dr. Ali Emre Pusane** Boğaziçi Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Ece Olcay Güneş** İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Tolga Mete Duman Bilkent Üniversitesi	.....
	Dr. Öğr. Üyesi Semiha Tedik Başaran İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **9 Eylül 2022**
Savunma Tarihi : **12 Eylül 2022**

Anneme,

ÖNSÖZ

Tez çalışmam boyunca bilgi ve tecrübeleriyle bana her zaman yardımcı olan değerli hocalarım Prof. Dr. Sıddıka Berna Örs YALÇIN ve Prof. Dr. Ali Emre PUSANE'ye, Yonga Teknoloji Mikroelektronik bünyesinde deneyim kazanmamda büyük pay sahibi olan Dr. Hayrettin AYAR ve Rifat DEMİRCİOĞLU'na, hep yanımda olan ve beni destekleyen aileme teşekkürlerimi sunarım.

Eylül 2022

Barış BİLGİLİ
(Elektronik ve Haberleşme Mühendisi)

İÇİNDEKİLER

KISALTMALAR

AWGN : Additive White Gaussian Noise
5G NR : Fifth Generation New Radio
CNU : Check Node Unit
FIL : FPGA in the Loop
FPGA : Field Programmable Gate Array
HDL : Hardware Description Language
LDPC : Low Density Parity Check
PUDCH : Physical Downlink Shared Channel
PUSCH : Physical Uplink Shared Channel
SPA : Sum Product Algorithm
VHDL : Very High speed Integrated Circuit Hardware Description Language

TABLO LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

FPGA ÜZERİNDE 5G UYUMLU DÜŞÜK YOĞUNLUKLU EŞLİK DENETİM KOD ÇÖZÜCÜ GERÇEKLENMESİ

ÖZET

Günümüzde giderek artan sayısal veri üretimi ve veri ihtiyacı, bu verilerin iletilebilmesi için yüksek hızlı kablosuz haberleşme sistemlerini giderek daha önemli hale getirmektedir. Taşınan veri miktarının artması yeni gereksinimleri de beraberinde getirmektedir. Bunlardan ilki haberleşmenin daha hızlı yapılabilmesidir. İkincisi ise bu verilerin kanaldaki bozulmalardan etkilenmeden alıcı tarafa iletilebilmesidir. Haberleşme insanlar veya makineler arasında gerçekleşse de, hücresel ağlar veya uydu üzerinden sağlansa da yeni gereksinimler eklenebilmesine rağmen bu iki gereksinim değişmemektedir. Bu noktada üretilen standartlar belirtilen gereksinimleri karşılamaya çalışmaktadır. Hücresel haberleşme için güncel bir standart olan 5G'de ileri hata kodlama olarak Düşük Yoğunluklu Eşlik Denetim (Low Density Parity Check - LDPC) kodları veri kanallarındaki bu gereksinimleri karşılamak için önerilmiştir. Uydu haberleşmesinde ise İkinci Nesil Sayısal Video Yayını (Digital Video Broadcasting - DVB S2) gibi standartlarda LDPC kodları kullanılmaktadır.

LDPC kodları yapıları itibariyle esnek tasarım ve uygulamalara uygun kodlardır. Farklı blok boylarında ve paralel çalışmaya elverişli oldukları için Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array - FPGA) ile gerçeklenmeleri avantajlı bir hale gelmektedir. LDPC kodları farklı kod çözme algoritmalarıyla çalışabildikleri için FPGA gerçeklemeleri yapılmadan önce bu algoritmalar performans ve gerçeklemeye uygunluk açısından incelenmelidir. Kod çözücünün düşük alan kullanımına ve yüksek veri hacmine sahip olması gerektiği için buna uygun bir algoritma seçilmelidir.

LDPC kodları genellikle bir eşlik denetim matrisi ile tanımlanırlar. Kod çözücü tasarımında bu matris, veri depolama birimlerinin boyutlarını ve bağlantıları belirler. Kod çözücüde algoritmanın çalıştığı asıl birim ise Denetim Düğümü Birimi (Check Node Unit - CNU) olarak tanımlanır. Bu çalışmada 5G Yeni Radyo (5G New Radio - 5G NR) standardı temel alındığı için veri boyutları ve bağlantıları büyük oranda belirlidir. Algoritma seçimi, paralelleştirme ve veri hacmini arttırma üzerine çalışmalar yapılmıştır. Donanım gerçeklemesi yapılırken karşılaşılan veri depolama, adresleme ve sıralama sorunlarına çözümler üretilmeye çalışılmıştır.

Döngüde FPGA (FPGA in the Loop - FIL), FPGA'de çalışması için bir donanım tanımlama diliyle (Hardware Description Language - HDL) yazılmış kodları MATLAB ortamı ile entegre ederek gerçek donanım üstünde çalışan kod ile yazılımdaki kodların beraber benzetiminin yapılması sağlayan doğrulama programıdır. HDL ile tasarım yaparken doğrulama yapmak çok önemli bir yer tutmaktadır ve FIL kullanılmadığı durumda herhangi bir bloğun doğrulamasını yapmak için test dosyaları

oluşturup veri grupları hazırlayarak benzetim yapılması gerekmektedir. FIL sayesinde MATLAB ortamında oluşturulan veriler örnek modelle aynı anda gerçek donanım üzerinde çalışan HDL koduyla kıyaslanarak sonuçları doğrulanabilmektedir.

5G NR standardındaki LDPC matrisleri farklı boyutlara ve farklı satır ağırlıklarına sahip oldukları için bu çalışmada tasarlanan LDPC eşlik denetim biriminin farklı sayıda giriş ile çalışabilmesi gerekmektedir. Bu nedenle FIL kullanılarak farklı sayıda girişler için MATLAB ortamında doğrulama yapılmış ve FPGA üzerinde çalıştırılarak test edilmiştir.

Bu çalışmada hem FIL ile doğrulama yaparak tasarım ve doğrulama süreçlerinin hızlandırılması, hem de donanıma uygun algoritmalar seçilerek karmaşıklığı düşük ve veri hacmi yüksek bir eşlik denetim birimi tasarlanması, eşlik denetim biriminin çalışmasına örnek göstermek amacıyla 5G NR standardına uygun bir üst seviye tasarımının yapılması amaçlanmıştır.

IMPLEMENTATION OF 5G COMPATIBLE
LOW DENSITY PARITY CHECK DECODER ON FPGA

SUMMARY

Nowadays, the ever-increasing digital data production and data need make high-speed wireless communication systems more and more important in order to transmit these data. The increase in the amount of transmitted data brings new requirements with it. The first of these requirements is to increase the speed of communication. The second requirement is that the data is transmitted to the receiver without being affected by the channel noise. These two requirements do not change, although new requirements can be added whether communication takes place between humans or machines, or via cellular networks or satellite. At this point, standards are established to meet the specified requirements. In 5G, which is a current standard for cellular communication, Low Density Parity Check (LDPC) codes as forward error coding have been proposed to meet these requirements in data channels. These channels are Physical Uplink Shared Channel (PUSCH) and Physical Downlink Shared Channel (PDSCH). On the other hand, in satellite communication, there are LDPC codes in standards such as Digital Video Broadcasting Satellite Second Generation (DVB-S2).

LDPC codes are defined by parity check matrices and consist of two types of base units. These units are called bit nodes and check nodes. Each row in the parity check matrix represents a check node, and each column represents a bit node. The parity check matrix consists of ones and zeros. If the value at the intersection of any row and column is 1, it means that there is a connection between the intersecting bit node and the check node at this point. In the 5G NR standard, LDPC codes are defined with 2 base graphs. A large number of LDPC parity matrices are obtained from base graphs by expanding them with lifting sizes to support different block lengths.

LDPC codes, by their nature, are suitable for flexible design and applications. Since they are suitable for working in different block sizes and working in parallel, their implementation with Field Programmable Gate Arrays (FPGA) becomes advantageous. LDPC codes can work with different decoding algorithms, therefore these algorithms should be examined for performance and compatibility with implementation on FPGA before hardware design is made. Since the LDPC decoder should have low utilization and high throughput, an appropriate algorithm should be selected. Among the LDPC decoding algorithms, sum-product algorithm, min-sum algorithm, offset min-sum algorithm and attenuated min-sum algorithm come to the fore. The sum-product algorithm is also known as belief-propagation algorithm and is the default decoding algorithm in the literature. In all of these algorithms, the operation performed on the bit nodes is exactly the same, while the operation on the control nodes differs. Although the sum-product algorithm is the algorithm that gives the highest performance for LDPC codes, the use of the hyperbolic tangent function

greatly increases both the computational complexity and the hardware implementation complexity. Therefore, hardware implementations given in the literature generally use the min-sum algorithm.

LDPC codes are usually defined by a parity check matrix. In the LDPC decoder design, this matrix determines the area of the data storage units and connections between the units in the design. The actual unit in the decoder where the algorithm runs is defined as the Check Node Unit (CNU). Since this study is based on the 5G New Radio standard, data sizes and connections are largely specific. Studies have been carried out on algorithm selection, parallelization and increasing the throughput. Solutions have been produced for data storage, addressing and scheduling problems encountered during hardware implementation.

Input data of the LDPC decoder must be arranged in accordance with the matrix structure and stored in the RAMs. Taking advantage of the adjustable RAM widths, multiple messages can be stored and accessed together. In this way, high throughput can be achieved with parallel working CNU designs. As a solution, bit shuffling and circular shift methods have been proposed while accessing RAMs in studies in the literature. By using the bit shift values stored in the ROMs according to the connections in the matrix, the data stored together in the RAMs can be distributed to the bit node units and check node units in the correct order during a single read. While the processed data is writing back to the RAMs, it is written back to the correct addresses by reverse shuffling or reverse bit shifting with the same values stored in the ROM. Appropriate check node units and access sequences are suggested to avoid conflicts in RAM access while performing these read and write operations. The work of the check nodes should be sequenced, taking into account their latency. Conflicts occur when trying to access the same address to read in the next layer before the data is written to RAM after the previous layer has been processed. As a result of these conflicts, the data that has not been updated is transferred to the next layer and the results are overwritten in the same address in RAM. To avoid this situation, access sequences need to be regulated. Access sequences vary for different LDPC matrices and hardware implementations.

As a result of the studies on the literature, it has been decided that the most suitable structure to be implemented in hardware can be obtained by using the min-sum algorithm. This algorithm is suitable to be run on FPGA in terms of both having low complexity and being suitable for sequential access. For the design to be made with fixed points, keeping the messages 6 bit wide was found suitable as a result of the performance tests made on the min-sum algorithm model. Since the check node messages are collected at the bit nodes, the bit width is defined 2 bits more at the bit nodes.

CNU receives messages from the bit nodes and check nodes sequentially. Messages are indicated by the data valid signal. The end of the incoming message block is indicated by the end-of-block sign. When the CNU is ready to receive data on its input, it sends out a ready signal. A corresponding message output is sent to each of the bit node and check node messages coming to the CNU input. In this way, both the bit node and the control node can be updated simultaneously. In order for the updates between the layers in the LDPC matrix to take place without interruption, the CNU must be able to

accept new data from the input while sending the valid data to the output. Otherwise, after updating the nodes of the previous layer, it is necessary to wait for all the inputs to be received and processed again in order to update the next layer. While the messages are given to the output, the new inputs received are processed in separate sequences, thus making the CNU operation uninterrupted.

FPGA in the Loop (FIL) is a validation program that integrates the codes written in a hardware description language (HDL) with the MATLAB environment to work in the FPGA, allowing the code running on the real hardware to be simulated together with the codes in the software. While designing with HDL, validation has a very important place and in case of not using FIL, it is necessary to simulate by creating test files and preparing data sets to validate any design block. By using the FIL, software model and HDL design running on hardware can be tested simultaneously by applying the data created in the MATLAB environment, and the results can be compared.

FIL can be used in different formats. By programming the FPGA via FIL with any HDL code, and sending the data from the MATLAB environment to the FPGA, the results can be observed and processed in the MATLAB environment. In this way, the processing time for big data can be shortened by taking advantage of the parallel operation feature of the FPGA. For validation and testing purposes, the model can be created in the MATLAB environment and run simultaneously with the HDL code on the FPGA, and the results of both the model and the hardware can be observed in the MATLAB environment. FIL was used for validation and testing purposes in this study.

Since the LDPC matrices in the 5G NR standard have different sizes and different row weights, CNU designed in this study should be able to work with different number of inputs. For this reason, verification was made in the MATLAB environment for different numbers of inputs using FIL and tested by running on FPGA.

In this study, it is aimed to accelerate the design and verification processes by verifying with FIL, and to design a low complexity and high throughput CNU by choosing algorithms suitable for the hardware, and to make a top-level design in accordance with the 5G NR standard in order to illustrate the operation of CNU. After the CNU design was validated with the FIL, a high-level architecture was designed for use within the LDPC Decoder. If all connections are to be processed at the same time, as many CNUs as the number of connections must be implemented on the FPGA. For this reason, the LDPC matrix was examined in a layered structure and each row was processed in serial layers. Reading and writing of each message in a layer is done serially. In order to provide high throughput, when a layer has finished all its readings, the next layer's readings are started without waiting for it to finish writing. When working in the order given in the 5G NR standard during the transitions between the layers, it is encountered that the next layer tries to reach the same address and sends the outdated data to the CNUs before the previous layer has updated the RAM yet. In this study, a different access order is proposed to resolve conflicts in RAM accesses. The order of access to addresses in reads within the layer does not affect the result. Based on this fact, the addresses are sorted in a unique way, preventing the outdated data from being read in the next layer.

Future studies will focus on error performance and try to implement more advanced algorithms such as offset min-sum and attenuated min-sum, which are built on the

min-sum algorithm. Since these improvements will only be made on the CNU, they can be verified with the FIL and added to the LDPC decoder design without changing the top-level architecture. The development and validation environment created with FIL will be used by making customizations on the model.

1. GİRİŞ

Bu bölümde konuya ilişkin bilgiler verilerek temel kavramlar açıklanmıştır ve bu çalışmanın amacından bahsedilmiştir.

LDPC kodlarının kullanımı, günümüz kablosuz haberleşme sistemlerinde giderek yaygınlaşmaktadır. Farklı uygulama alanları için oluşturulmuş standartlarda yer alması, LDPC kodları ile daha esnek tasarım yapabilme ihtiyacını arttırmaktadır. LDPC kodları eşlik denetim matrisleri ile tanımlanmaktadır. Bu matris değiştikçe farklı tasarım gereksinimleri doğurmaktadır.

Sayısal haberleşme sistemlerinin en temel bloklarından biri kanal kodlama bloğudur [1]. Kanal kodlama bloğunun amacı kanalda meydana gelen bozulmaları alıcı tarafta tespit edip düzeltmektir. Katlamalı kodlar ve blok kodlar olmak üzere sıklıkla kullanılan iki kanal kodlama çeşidi vardır. Katlamalı kodlar, çıkıştaki bitlerin, veri akışındaki mevcut bit ve az sayıda önceki bit üzerindeki mantıksal işlemlerle belirlendiği kodlardır. Blok kodlarda ise bilgi bitleri büyük bloklar halinde kodlanarak eşlik bitlerine karar verilir. Katlamalı kodlar arasında iteratif kod çözme özelliğiyle Turbo kodlar ön plana çıkmaktadır [2]. Kod çözme sırasında kanaldan gelen bilgi bitlerine karar vermeden önce eşlik bitleriyle birlikte karar verme algoritmaları geri beslemeli iterasyonlar halinde çalıştırılarak yüksek hata performansı elde edilebilmektedir [3]. LDPC kodları, iteratif olarak çalışan blok kodlarıdır. Bu sayede büyük veri bloklarının yüksek hata performansıyla çözülmesine imkan sağlamaktadır [4].

LDPC kod çözme algoritmaları arasında toplam-çarpım algoritması, min-toplam algoritması, dengelenmiş min-toplam algoritması ve zayıflatılmış min-toplam algoritması ön plana çıkmaktadır [1].

Bu tezin amacı, LDPC kodlarının donanım gerçeklemesi sırasında ihtiyaç duyulan en önemli birim olan eşlik denetim birimini 5G standardı için tasarlamaktır. 5G

NR standardında tanımlanan LDPC matrisleri kendi aralarında çeşitlilik gösterdikleri için eşlik denetim biriminin de esnek bir yapıda tasarlanması gerekmektedir. Bu esnek tasarım sayesinde farklı standartlar için de uyumlu çalışabilecek bir birim tasarlanması amaçlanmaktadır. Tasarım sırasında doğrulamayı ve donanım testlerini pratik bir şekilde yapabilmek için FIL kullanılarak tasarım ve doğrulama süreçlerinin de hızlandırılması amaçlanmıştır [5].

Tezin ikinci bölümünde 5G NR LDPC kodlar ve kod çözücüler ilgili bilgiler verilmiştir. Üçüncü bölümde donanım gerçekleme temelleri anlatılmıştır. Dördüncü bölümde literatür analizi ve üzerine yapılan çalışmalar, beşinci bölümde ise FPGA üzerinde kod çözücü tasarımı anlatılmıştır. Altıncı bölümde sonuçlardan ve gelecek çalışmalardan bahsedilmiştir.

2. 5G NR LDPC KODLAR VE KOD ÇÖZÜCÜLER

Bu bölümde LDPC kodlama ve kod çözme kavramları açıklanmıştır. 5G NR LDPC kodlarının özelliklerinden bahsedilmiştir.

2.1 LDPC Kodlama ve Kod Çözme

LDPC kodları eşlik denetim matrisleriyle tanımlanır ve iki tip temel birimden oluşurlar. Bu birimlere bit düğümleri ve denetim düğümleri denir. Eşlik denetim matrisindeki her satır bir denetim düğümünü, her sütun ise bir bit düğümünü ifade eder. Eşlik denetim matrisi 1'lerden ve 0'lardan oluşmaktadır. Herhangi bir satır ile sütunun kesiştiği noktadaki değer 1 ise bu noktada kesişen bit düğümü ve denetim düğümü arasında bağlantı olduğu anlamına gelir. Eşlik denetim matrisindeki bu bağlantılar Tanner Çizgesi denilen çizgelerle Şekil 2.1 ve Şekil 2.2'de örnek verildiği şekilde gösterilir [6]. Bit düğümleri sol tarafta daire biçiminde, denetim düğümleri sağ tarafta kare biçiminde gösterilmiştir.

$$H=\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Şekil 2.1 : LDPC Eşlik Denetim Matrisi Örneği

LDPC kodlayıcının amacı giriş bitlerine bakarak kod cümleleri üretmektir. Eşlik denetim matrisi H olmak üzere, üreteç matrisi G

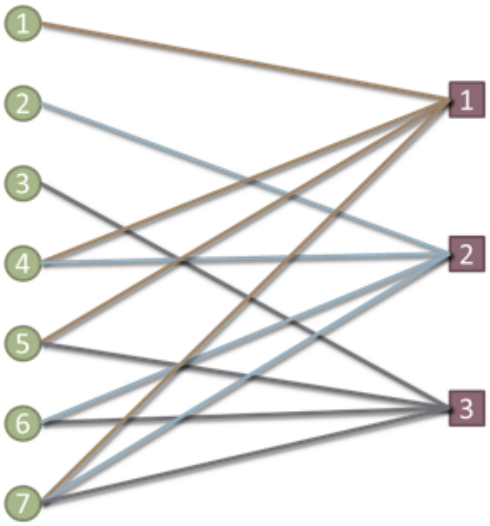
$$G \times H^{Transpoze} = 0 \tag{2.1}$$

biçiminde gösterilir. Giriş dizisi M, oluşturulacak kod cümlesi C olmak üzere

$$M \times G = C \tag{2.2}$$

şeklinde kodlama yapılır. Bu durumda kod cümlesinin

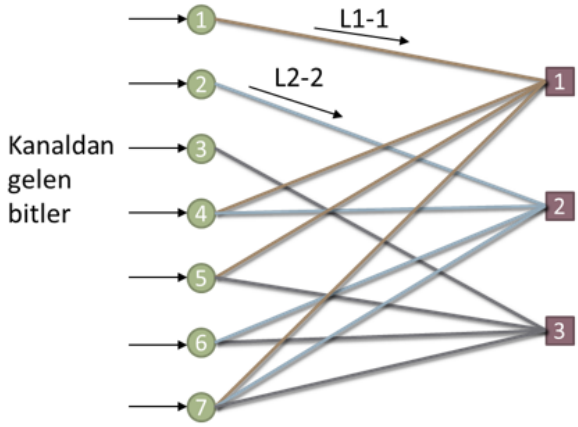
$$C \times H^{Transpoze} = 0 \tag{2.3}$$



Şekil 2.2 : Tanner Çizgesi Örneği

eşitliğini sağlaması gerekmektedir.

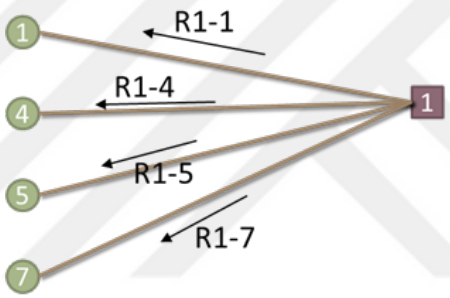
LDPC kod çözücü iteratif yapıda çalışmaktadır. Bit düğümleri ve denetim düğümleri arasındaki mesaj aktarımları ile bu iterasyonlar gerçekleşmektedir. Maksimum iterasyon sayısı, yapılacak olan mesaj aktarımlarının maksimum sayısını belirler. İterasyonlar sırasında bit düğümlerindeki değerler üzerinden ara kararlar verilerek Denklem 2.3'ü sağlayan geçerli bir kod cümlesine ulaşılırsa iterasyonlar sonlandırılarak kod cümlesinin doğru olduğuna karar verilir. Bit düğümlerinden denetim düğümlerine doğru olan mesaj aktarımı Şekil 2.3'te gösterildiği biçimdedir.



Şekil 2.3 : Bit Düğümlerinden Denetim Düğümlerine Mesaj Aktarımı

İterasyonlar başlamadan önce kanaldan gelen LLR değerleri bit düğümlerine yazılır. Denetim düğümlerinin hepsine sıfır değeri atanır. İterasyonlar iki adımda

gerçekleştirilir. İlk adımda bit düğümleri tuttukları mesajları bağlı oldukları denetim düğümlerine aktarır. Şekil 2.3'te aktarılan mesajlar L1-1 (birinci bit düğümünden birinci denetim düğümüne aktarılan mesaj), L2-2 (ikinci bit düğümünden ikinci denetim düğümüne aktarılan mesaj) biçiminde örneklenmiştir. İkinci adımda ise denetim düğümleri, bağlı olduğu bit düğümlerinden aldıkları mesajlara göre geri gönderecekleri mesajlara karar verirler. Bu mesajlar LLR değerlerine bakarak bitleri 0 veya 1 olduğuna karar verilebilmesi için önemlidir. Denetim düğümlerinden bit düğümlerine geri gönderilen mesajlar Şekil 2.4'te örneklenmiştir. Bu örnekte R1-1 birinci denetim düğümünden birinci bit düğümüne gönderilen mesajı, R1-4 birinci denetim düğümünden dördüncü bit düğümüne gönderilen mesajı, R1-5 birinci denetim düğümünden beşinci bit düğümüne gönderilen mesajı, R1-7 birinci denetim düğümünden yedinci bit düğümüne gönderilen mesajı temsil etmektedir.



Şekil 2.4 : Denetim Düğümlerinden Bit Düğümlerine Mesaj Aktarımı

Geri gönderilecek mesajları belirlemek için kullanılan algoritmalar arasında en iyi hata performansına sahip olanı Toplam-Çarpım Algoritmasıdır (Sum Product Algorithm - SPA). Bu algoritma

$$R = -sign \times tanh^{-1}(|A|/2) \tag{2.4}$$

biçiminde gösterilir. Mesajların pozitif veya negatif olmalarına göre işaretleri Denklem 2.4'te sign ile gösterilmiştir, mutlak değer ifadesinin içinde bulanan A ifadesi

$$A = tanh(|LLR_1|/2) + tanh(|LLR_2|/2) + \ldots + tanh(|LLR_N|/2) \tag{2.5}$$

olarak gösterilebilir. Denklem 2.5'te logaritmik olasılık oranları LLR ifadesiyle gösterilmiştir.

2.2 5G NR LPDC Kodları

5G NR standardında, Fiziksel Yukarı Bağlantı Paylaşımlı Kanal (Physical Uplink Shared Channel - PUSCH) ve Fiziksel Aşağı Bağlantı Paylaşımlı Kanal (Physical Downlink Shared Channel - PDSCH) veri haberleşmesi kanallarında LDPC kodları kullanılmaktadır [7]. LDPC kodları eşlik denetim matrisleriyle birlikte tanımlanır. Bu eşlik denetim matrisleri düşük yoğunluklu olarak tasarlanır. Düşük yoğunluklu olduklarından dolayı matristeki 1'lerin sayısı 0'larından sayısından oldukça azdır. 5G NR standardında LDPC kodları, 2 adet temel çizge ile tanımlanmışlardır. Farklı blok uzunluklarını destekleyebilmek için Tablo 2.1'de verilmiş olan Z yükseltme çarpanlarıyla genişletilerek temel çizgelerden çok sayıda LDPC eşlik matrisi elde edilmektedir [7].

Tablo 2.1 : Yükseltme Çarpanı Tablosu

İndis	Yükseltme Çarpanı
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

LDPC kodlama yapılırken gelen paketlere eşlik bitleri çizgelerin temel kodlama oranlarına göre eklenir [8]. Eklenen eşlik bitlerinin sayısı değiştirilerek farklı kodlama oranları elde edilir. Kod çözme tarafı standartta tanımlanmamıştır. Standartta verilen temel çizgeler üzerinden, alıcı tarafta farklı algoritmalar kullanılarak kod çözme işlemi uygulanabilir. Bu sayede farklı uygulamalar ve gerçeklemeler için uygun kod çözme yöntemlerini kullanmaya imkan tanınmıştır.

3. DONANIM GERÇEKLEME TEMELLERİ

Bu bölümde donanım gerçeklemesi için kullanılan temel yapılar ve işlemler anlatılmıştır. Bu çalışmada donanım gerçeklemek ve test etmek için kullanılmış olan FIL kavramı açıklanmıştır.

3.1 Donanım Yapısı

Donanım tanımlama dilleri (Hardware Description Language - HDL), elektronik devrelerin ve sayısal mantık devrelerinin yapısını ve davranışını tanımlamak için kullanılan bilgisayar dilleridir [9]. HDL kullanarak büyük tasarımları kolayca oluşturmak mümkün hale gelmektedir. Küçük tasarımları bile hızlıca üretebilmek için HDL kullanılmaktadır.

HDL tabanlı tasarımlar, gelişmelere ayak uydurmak için tasarımların değiştirilmesine ve yeniden kullanmasına olanak tanır. Cihazların fiziksel boyutları küçüldükçe, HDL tabanlı modelden daha iyi performansa sahip daha yoğun devreler sentezlenebilir [10]. Bu çalışmada yapılan tasarımlarda VHDL dili kullanılmıştır.

Günümüzde yeni tasarımları olabilecek en kısa sürede ürün haline getirmek çok büyük bir öneme sahiptir. Bu nedenle geliştirme sürelerinin kısalması önemli bir ihtiyaçtır. Bu ihtiyacı karşılamak için hızlı ve düşük maliyetli bir şekilde prototipler geliştirilebilmesi gerekmektedir. Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array - FPGA), bu soruna çözüm oluşturmaktadır. FPGA, programlanabilir mantık blokları, giriş-çıkış blokları ve ara bağlantılardan oluşan sayısal tümleşik devrelerdir [11]. FPGA, HDL kullanılarak hızlı bir şekilde programlanabildiği ve paralel çalışabilme özelliğine sahip olduğu için yaygın olarak kullanılmaktadır.

3.2 Veri Gösterimi ve İşlemesi

Bu çalışmada veri gösterimi için ikiye tümleyen gösterimi kullanılmıştır. Donanım gerçeklemesi için kullanılan temel işlemler olan kuantalama ve bit kaydırma işlemleri bu bölümde anlatılmıştır.

3.2.1 İkiye tümleyen gösterimi

Bu çalışmada tasarlanan eşlik denetim biriminde yapılan işlemlerde ikiye tümleyen (Two's Complement) veri gösterimi kullanılmıştır. İkiye tümleyen, bilgisayarların sayıları ifade etme şeklidir. İkiye tümleyen gösteriminde bir sayının negatifini göstermek için sayı ikili düzende yazılır, 1'ler 0'larla, 0'lar 1'lerle değiştirilir ve sonuca bir eklenir [12]. Örneğin, -28 sayısını ifade etmek için önce 28 sayısı ikiye tümleyen halinde yazılır ve aşağıdaki adımlar izlenir:

- 0 0 0 1 1 1 0 0
- 1'ler 0'larla, 0'lar 1'lerle değiştirilir.
- 1 1 1 0 0 0 1 1
- Sonuca bir eklenir.
- 1 1 1 0 0 1 0 0

İkiye tümleyen gösteriminde toplama ve çıkarma basit bir şekilde yapılabilir. Bu sayede toplama ve çıkarma devreleri birleştirilebilir, aksi takdirde ayrı işlemler olarak ele alınmaları gerekir [12].

3.2.2 Kuantalama işlemi

Eşlik denetim biriminde gerçekleşen işlemlerde veri hassasiyetini kaybetmemek için ara çıktılarda bit genişliğinin artmasına izin verilecek şekilde bir tasarım yapılmıştır. Veri depolamak için tasarımda ayrılan alanın büyüklüğü önceden belirlenmiş bir miktara sahip olduğundan eşlik denetim birimi çıkışa veri gönderirken içerideki

işlemlerde artan bit genişliğini kuantalamak zorundadır. Bunun için öncelikle kuantalama işlemi uygulanacak verinin en anlamlı bitine bakılarak ikiye tümleyen gösterimine göre pozitif mi yoksa negatif mi olduğuna karar verilir. Sonrasında sayının mutlak değerine bakılarak sayı gösterimlerine göre ifade edilebilecek en büyük ve en küçük sayıların sınırını aşıp aşmadığına bakılır. Aşıyorsa maksimum veya minimum değer olarak ikiye tümleyen biçiminde çıkışa verilir. Diğer durumlarda ifade edilecek sayı, tamsayı ve kesirli kısımları dikkate alınarak istenilen bit genişliğinde çıkışa verilir.

3.2.3 Bit kaydırma işlemi

Tasarımdaki veri yerleşiminden dolayı eşlik denetim biriminin girişine veri depolama birimlerinden doğru verileri aktarmak ve eşlik denetim biriminin çıkışındaki verileri doğru bir şekilde yazmak için bit kaydırma işlemi uygulanması gerekmektedir. Bu işlem için Barrel Shifter yapısı kullanılmıştır. Böylelikle ardışıl bir mantık devresi kullanılmadan bit kaydırma işlemi kombinezonsal mantık devreleriyle gerçeklenebilmiştir. İstenilen bit kaydırma miktarına göre girişine gelen bitleri sola kaydırarak bu işlemi aşağıdaki gibi gerçekleştirmektedir:

- Girişe gelen dizi :
- **1 1 0** 1 0 0 1 0
- Bit kaydırma miktarı : 3
- Yedinci bitten beşinci bite kadar olan bitler en sağa yerleştirilir ve kalan bitler soluna birleştilir:
- 1 0 0 1 0 **1 1 0**
- Böylece üç bit kaydırma işlemi yapılır.

3.3 FPGA-in-the-Loop

FIL, mevcut herhangi bir HDL kodunu MATLAB veya Simulink ortamında geliştirilmekte olan modellere entegre edebilen ve test senaryolarını FPGA üzerinde

HDL tasarımına uygulayabilen bir benzetim aracıdır [5]. FIL benzetimi yapabilmek için MATLAB ürünlerinin yanında bir FPGA tasarım yazılımı, FPGA kartı ve kart ile bilgisayar arasındaki bağlantıyı sağlayacak ethernet, JTAG veya PCIe arayüzü gerekmektedir [13]. Bu çalışmada Xilinx Zedboard kullanıldığı için bilgisayar ile FPGA arasında JTAG bağlantısı kullanılmıştır. FPGA tasarım yazılımı olarak kullanılan Vivado, MATLAB ortamına komut penceresinde aşağıdaki kod çalıştırılarak entegre edilmiştir:

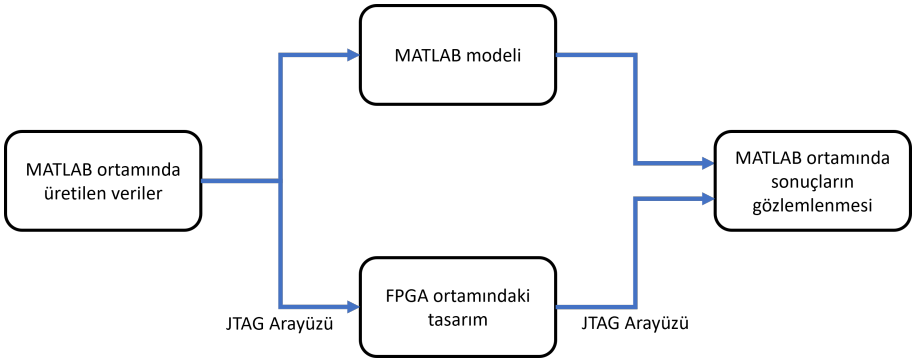
```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',
'C:\Xilinx\Vivado\2021.1\bin\vivado.bat');
```

FPGA tasarım yazılımı entegre edildikten sonra FILWizard açılarak benzetimi yapılmak istenen HDL kodları eklenir ve üst seviyede bulunan dosya seçilir. Bir sonraki adımda modülün giriş-çıkışları için veri, saat, saat seçme, sıfırlama türlerinden biri seçilir. Sıfırlama ve saat seçme işaretleri aktif-düşük veya aktif-yüksek olarak seçilebilir. Veri olarak seçilen tür dışındaki işaretleri MATLAB otomatik olarak sürer. Veri türündeki sinyaller kullanıcı tarafından sürülür. Bu sayede modülün bütün işaretler MATLAB ortamında kontrol edilebilir. Bir sonraki adımda çıkış işaretlerinin veri türleri seçilir. Tüm seçimler yapıldıktan sonra FILWizard, FPGA tasarım yazılımını kullanarak FPGA projesini oluşturur ve FPGA programlamak için gerekli komut dosyası ile HDL tasarımını modelde kullanmak için gereken FIL sınıfını çıktı olarak verir.

FIL, farklı biçimlerde kullanılabilir. Herhangi bir HDL kodu ile FIL aracılığıyla FPGA programlanıp MATLAB ortamından gelen veriler FPGA'e gönderilerek sonuçlar yine MATLAB ortamında gözlemlenip işlenebilir. Bu sayede FPGA'in paralel çalışabilme özelliğinden yararlanılarak büyük veriler için işleme süresi kısaltılabilir. Doğrulama ve test amaçlı kullanımda ise MATLAB ortamında model oluşturularak FPGA üzerindeki HDL kodu ile aynı anda çalıştırılabilir ve hem modelin hem donanımın sonuçları MATLAB ortamında gözlemlenebilir [14].

Bu çalışmada FIL doğrulama ve test amaçlı kullanılmıştır. Bu sayede tasarım sırasında testlerin yapılması kolaylaşmıştır. Donanım tasarımına başlamadan önce MATLAB ortamında model oluşturulmuş, yazılan HDL kodları FIL sayesinde

modelle kıyaslanarak hatalar tespit edilmiştir. Kullanılan test senaryosu Şekil 3.1'de gösterilmiştir.



Şekil 3.1 : FIL Test Senaryosu

FIL ile çalışılırken donanıma gönderilecek verilerin sabit noktalı olması gerekmektedir. Bu nedenle MATLAB ortamında oluşturulan veriler HDL kodunun girişlerinin bit genişliklerine uygun olacak şekilde sabit noktalı veri tipine dönüştürülmelidir. Kontrol işaretleri de giriş verileriyle birlikte sürülerek çıkış verileri MATLAB çalışma alanında incelenebilir veya işlenmeye devam edilebilir.

4. DONANIM UYUMLU LDPC KOD ÇÖZME

LDPC kod çözücülerde donanım gerçeklemesi yapılırken teorik çalışmalarda karşılaşılmayan sorunlar ortaya çıkmaktadır. Donanım mimarisine uygun tasarım yapılırken bellek yönetimleri ve veri tipleri kod çözücünün çalışmasını etkilemektedir. Bu bölümde LDPC kod çözücülerin donanım gerçeklemesi için uygun algoritmalar, veri tipleri ve kod çözme çizelgeleri üzerine yapılan çalışmalar anlatılmıştır.

4.1 Literatürdeki Kod Çözme Algoritmaları

LDPC kodların çözümü konusunda literatürde sıklıkla kullanılan dört adet algoritma bulunmaktadır: toplam-çarpım algoritması, min-toplam algoritması, dengelenmiş min-toplam algoritması ve zayıflatılmış min-toplam algoritması [1,15]. Bunlar arasında toplam-çarpım algoritması aynı zamanda kanı yayılımı algoritması olarak da bilinir ve literatürde varsayılan kod çözme algoritmasıdır. Bu algoritmaların hepsinde bit düğümlerinde yapılan işlem birebir aynı iken denetim düğümlerinde yapılan işlem fark etmektedir. Buna göre, aşağıda da özetlendiği üzere, toplam çarpım algoritması düğüme gelen mesajları 0.5 ile çarpıp hiperbolik tanjantını alır. Elde edilen mesajları toplayarak ters hiperbolik tanjant alarak çıkış mesajını hesaplar. Bu işlem oldukça karmaşık görünse de aslında gelen K-1 mesajı birleştirerek K. kenar için çıkış mesajını hesaplar. Bu sırada, arka planda, K. kenarın bağlı olduğu bit düğümünün 0 veya 1 olması için gerekli tüm kombinasyonları değerlendirmiş olur. Örneğin, K. kenardaki çıkış mesajının 0 olması için diğer K-1 girişin toplamının 0 olması gerekmektedir. Bu da oldukça çok sayıda bit kombinasyonlarına karşılık gelmektedir. Toplam-çarpım algoritmasındaki karmaşık giriş-çıkış ilişkisi bu kombinasyonların toplamlarına denk gelmektedir.

Toplam-çarpım algoritması LDPC kodlar için en yüksek başarımı veren algoritma olsa da hiperbolik tanjant işlevinin kullanımı hem hesap karmaşıklığını hem de donanım gerçekleme karmaşıklığını çok arttırmaktadır. Bu nedenle, literatürde verilen donanım

gerçeklemeleri genellikle min-toplam algoritmasını kullanır. Min-toplam algoritması toplam-çarpım algoritmasındaki giriş-çıkış özelliklerini basitçe modellemeyi hedefler. Şöyle ki, toplam-çarpım algoritmasının K-1 girişinin K. çıkış üzerine etkisi incelendiğinde, K-1 girişten mutlak değer olarak en düşük olanının tüm işlemde baskın olduğu ve K. çıkış için genellikle bu değerin çıktığı görülebilir. Bu gözleme dayanarak, hiç hiperbolik tanjant hesabı yapılmadan girişler arasındaki en düşük değer doğrudan çıkış mesajı olarak belirlenebilir. Min-toplam algoritması

$$R = sign \times \min_{1 \leq i \leq N} (|LLR_i|) \tag{4.1}$$

şeklinde ifade edilebilir. Bu şekilde kullanım yeteri kadar toplam-çarpım algoritması başarımına benzemekteyken donanım gerçeklemesini çok büyük oranda sadeleştirmektedir. Gelen mesajların karmaşık işlem devreleri veya tablolar aracılığıyla hiperbolik tanjantları alınacağına sadece en küçük mutlak değere sahip olanın belirlenmesi yetebilmektedir.

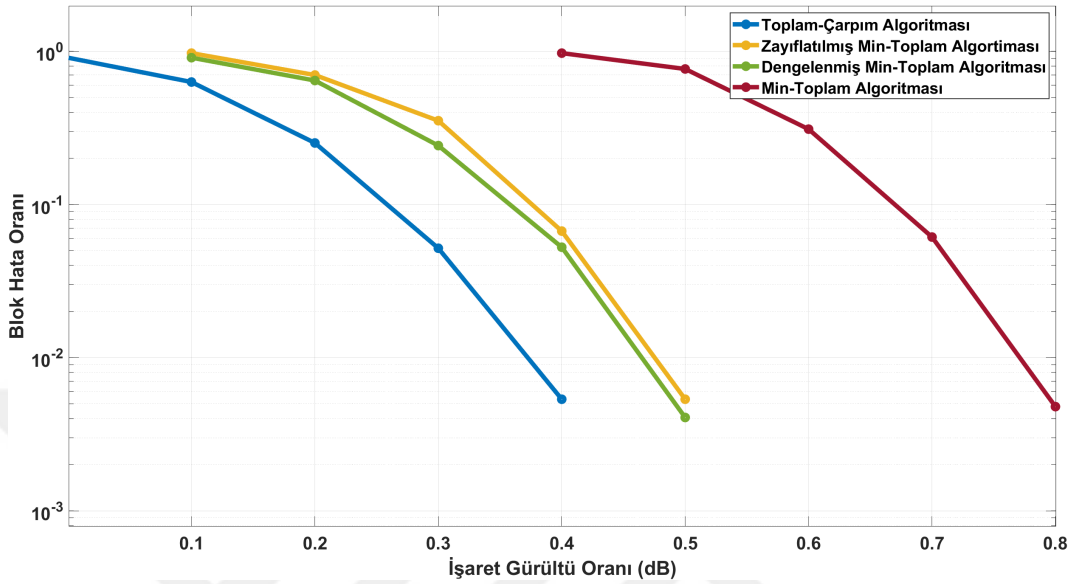
Min-toplam algoritmasının başarımını düşüren bir durum bulunmaktadır. Buna göre, toplam-çarpım algoritmasına benzemesiyle ilgili olan gözlemde bir varsayım bulunmaktadır. İlgili benzeme sadece gelen mesajlardan biri diğer K-2 mesaja göre çok daha küçükse geçerlidir. Eğer en küçük mesaj değerine yakın olan başka mesajlar da varsa bu durumda en küçük mesajı çıkış mesajı olarak seçmek aynı eşdeğerliği sağlamamaktadır. Bu durumu düzeltmek için literatürde iki farklı algoritma daha önerilmiştir. Bu algoritmalar, yine gelen mesajlardan en küçüğünü belirledikten sonra hatayı en aza indirmek için bu mesajı birden küçük bir sayıyla çarparak veya bir değer çıkararak zayıflatırlar. Böylece, algoritma çıkışının toplam-çarpım algoritmasına biraz daha yaklaşmasını sağlarlar. Söz konusu algoritmalar dengelenmiş ve zayıflatılmış min-toplam algoritmalarıdır. Dengelenmiş min-toplam algoritması

$$R = sign \times \max(\min_{1 \leq i \leq N} (|LLR| - Co), 0) \tag{4.2}$$

ifadesiyle gösterilebilir. Denklem 4.2'de hatayı en aza indirmek için en küçük mesajdan çıkarılan sayı Co ile gösterilmiştir. Zayıflatılmış min-toplam algoritması ise

$$R = sign \times Ca \times \min_{1 \leq i \leq N} (|LLR_i|) \tag{4.3}$$

biçiminde ifade edilir. Denklem 4.3'te en küçük mesaj ile çarpılan zayıflatma katsayısı Ca ile gösterilmiştir. İlgili algoritmaların benzetimleri gerçekleştirilmiş ve örnek olarak ele alınan bir LDPC kodu için blok hata oranının işaret gürültü oranına oranı cinsinden Şekil 4.1'deki sonuçlara ulaşılmıştır.

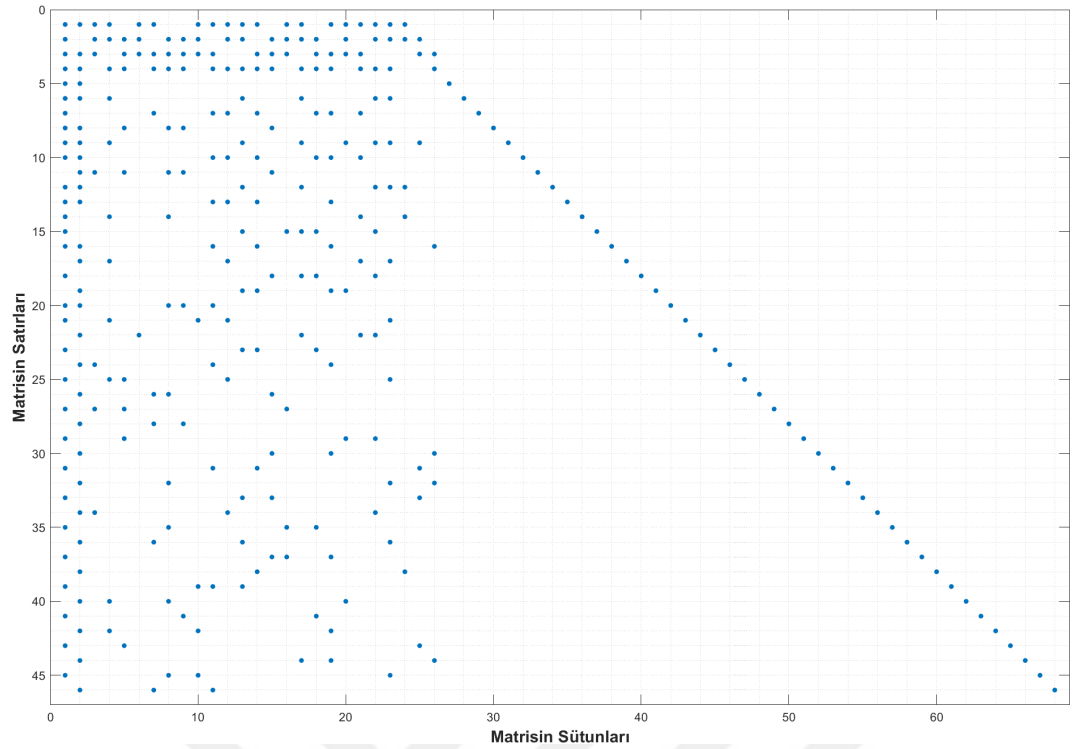


Şekil 4.1 : Literatürdeki Kod Çözme Algoritmalarının Benzetim Sonuçları

Şekil 4.1'e göre, standart "Min-Toplam Algoritması" nın "Toplam-Çarpım Algoritması" na göre kaybı yaklaşık 0.4 dB iken denenen diğer versiyonlar bu kaybın 0.3 dB'lik kısmını geri kazanmaya izin vermekte ve çok daha düşük karmaşıklıklarına rağmen toplam-çarpım algoritmasının başarımına göre sadece 0.1 dB kayıp göstermektelerdir. FPGA üzerinde donanım gerçeklemesi yapılırken öncelikli olarak min-toplam algoritması değerlendirilecek olup bu tasarımın geçerlilik testleri sonrasında dengelenmiş veya zayıflatılmış versiyona kolayca geçilebilecektir.

4.2 Kod Çözme Çizelgelerinin Karşılaştırılması

5G NR standardında tanımlanan LDPC kodların eşlik denetim matrisleri özel bir yapıya sahiptir. Özellikle donanım gerçeklemesi sırasında paralel işleme özelliği sağlasın diye ilgili matris öncelikle bir temel çizge olarak tanımlanır. Örneğin, standartta tanımlanmış iki adet temel çizgeden biri 46 satır, 68 sütundan oluşur ve Tablo 4.1'de gösterilmiştir. Şekil 4.2, Tablo 4.1'in mavi noktalar -1'den farklı olan değerleri göstermek üzere görselleştirilmiş halidir.



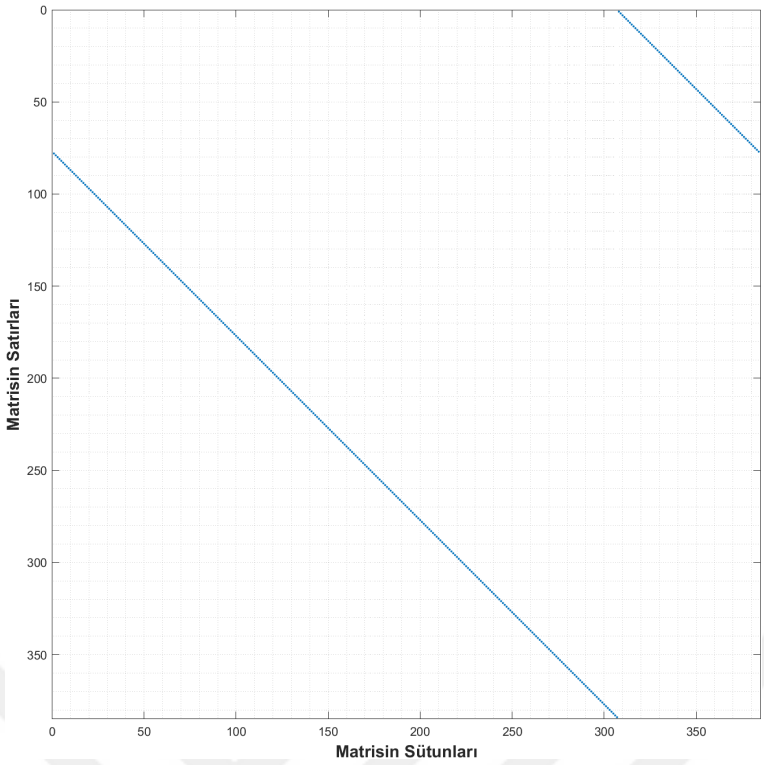
Şekil 4.2 : Tablo 4.1'de gösterilen 5G NR Temel Çizge 1 Matrisi

Bu temel çizgeden tam boyutlu bir eşlik denetim matrisi oluşturma, şekilde noktayla gösterilmiş yerleri Z×Z boyutlu kaydırılmış birim matrislerle değiştirerek, boş yerleri ise yine Z×Z boyutlu sıfır matrislerle değiştirerek elde edilir [16]. Örneğin, pratik bir örnek için standartta belirlenen tablolardan Z=384 seçildiğinde Şekil 4.4'teki tam boyutlu bir eşlik denetim matrisi elde edilir. Burada, görüleceği gibi her bir kare ya 384×384 sıfır matrisini ya da 384×384 kaydırılmış birim matrisini gösterir. Şekil 4.3'te, temel çizge matrisindeki ilk değer olan 307 değeri için 384 yükseltme çarpanıyla elde edilen kaydırılmış 384×384 birim matris örnek olarak gösterilmiştir.

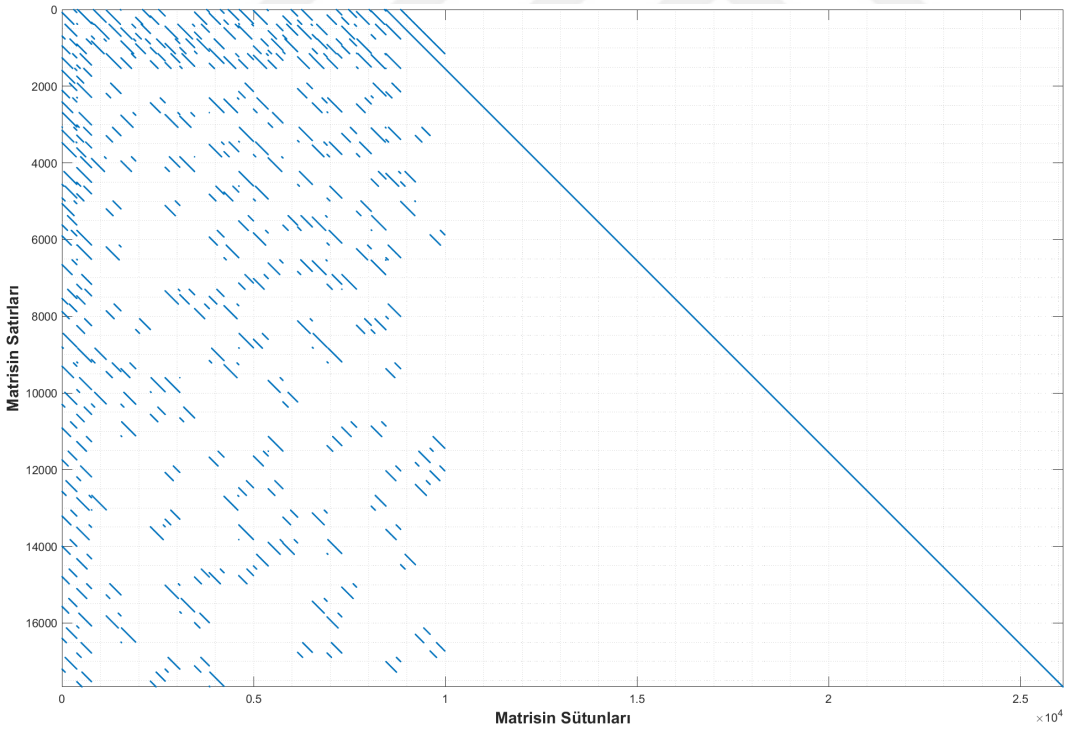
Elde edilen matrisin yapısına bakıldığında denetim düğümlerinin 46 adet, her biri 384 satırdan oluşan katmandan oluştuğu görülebilir. Bu katman içerisindeki 384 denetim düğümünün donanım tarafından eş zamanlı işlenebildiği varsayılır. Bu konu göz önünde tutulduğunda kod çözme için birden fazla olanak elde edilir. Bunlardan birincisi, literatürde paralel (parallel, flooding schedule) çizelge olarak adlandırılan çizelgedir [17]. Paralel çizelgede kodun katmanlı yapısı göz ardı edilir ve tüm 46×384=18.432 denetim düğümünün eş zamanlı olarak işlendiği varsayılır. Bilgisayarda bu mümkün olsa da donanımda gerçekleştirmesi mümkün olmayan bir çizelgedir. Bir diğer teknik ise kodun katmanlı yapısını göz önünde tutar ve her

Tablo 4.1 : 5G NR Temel Çizge 1 Matrisi Kaydırma Çarpanı Değerleri

307	19	50	369	-1	181	216	-1	-1	317	288	109	17	357	-1	215	106	-1	242	180	330	346	1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
76	-1	76	73	288	144	-1	331	331	178	-1	295	342	-1	217	99	354	114	-1	331	-1	112	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
205	250	328	-1	332	256	161	267	160	63	129	-1	-1	200	88	53	-1	131	240	205	13	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
276	87	-1	0	275	-1	199	153	56	-1	132	305	231	341	212	-1	304	300	271	-1	39	357	1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
332	181	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
195	14	-1	115	-1	-1	-1	-1	-1	-1	-1	-1	166	-1	-1	-1	241	-1	-1	-1	-1	51	157	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
278	-1	-1	-1	-1	-1	257	-1	-1	-1	1	351	-1	92	-1	-1	-1	253	18	-1	225	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	62	-1	-1	316	-1	-1	333	290	-1	-1	-1	-1	-1	114	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
307	179	-1	165	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	-1	-1	39	-1	-1	224	-1	368	67	-1	170	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
366	232	-1	-1	-1	-1	-1	-1	-1	-1	321	133	-1	57	-1	-1	-1	303	63	-1	82	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	101	339	-1	274	-1	-1	111	383	-1	-1	-1	-1	-1	354	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
48	102	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	-1	-1	-1	47	-1	-1	-1	-1	188	334	115	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
77	186	-1	-1	-1	-1	-1	-1	-1	-1	174	232	-1	50	-1	-1	-1	-1	74	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
313	-1	-1	177	-1	-1	-1	266	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	115	-1	-1	370	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
142	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	248	-1	-1	137	89	347	-1	-1	-1	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
241	2	-1	-1	-1	-1	-1	-1	-1	-1	210	-1	-1	318	-1	-1	-1	-1	55	-1	-1	-1	-1	-1	-1	269	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	13	-1	338	-1	-1	-1	-1	-1	-1	-1	57	-1	-1	-1	-1	-1	-1	-1	-1	289	-1	57	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
260	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	303	-1	81	358	-1	-1	-1	375	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	130	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	163	280	-1	-1	-1	-1	132	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
145	213	-1	-1	-1	-1	-1	344	242	-1	197	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
187	-1	-1	206	-1	-1	-1	-1	-1	264	-1	341	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	59	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	205	-1	-1	-1	102	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	328	-1	-1	-1	213	97	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
30	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11	233	-1	-1	-1	22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	24	89	-1	-1	-1	-1	-1	-1	-1	61	-1	-1	-1	-1	-1	-1	-1	27	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
298	-1	-1	158	235	-1	-1	-1	-1	-1	-1	339	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	234	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	72	-1	-1	-1	-1	17	383	-1	-1	-1	-1	-1	-1	312	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
71	-1	81	-1	76	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	136	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	194	-1	-1	-1	-1	194	-1	101	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
222	-1	-1	-1	19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	244	-1	274	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	252	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1	-1	-1	147	-1	-1	-1	-1	-1	-1	78	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
159	-1	-1	-1	-1	-1	-1	-1	-1	-1	229	-1	-1	260	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	90	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	100	-1	-1	-1	-1	-1	215	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	258	-1	-1	256	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
102	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	201	-1	175	-1	-1	-1	-1	-1	-1	-1	-1	-1	287	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	323	8	-1	-1	-1	-1	-1	-1	-1	-1	361	-1	-1	-1	-1	-1	-1	-1	-1	-1	105	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
230	-1	-1	-1	-1	-1	-1	148	-1	-1	-1	-1	-1	-1	-1	202	-1	312	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	320	-1	-1	-1	-1	335	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	266	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
210	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	313	297	-1	-1	21	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	269	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	82	-1	-1	-1	-1	-1	-1	-1	-1	-1	115	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
185	-1	-1	-1	-1	-1	-1	-1	-1	177	289	-1	214	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	258	-1	93	-1	-1	-1	346	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	297	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
175	-1	-1	-1	-1	-1	-1	-1	37	-1	-1	-1	-1	-1	-1	-1	-1	312	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
-1	52	-1	314	-1	-1	-1	-1	-1	139	-1	-1	-1	-1	-1	-1	-1	-1	288	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
113	-1	-1	-1	14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	218	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
-1	113	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	132	-1	114	-1	-1	-1	-1	-1	-1	168	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
80	-1	-1	-1	-1	-1	-1	78	-1	163	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	274	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
-1	135	-1	-1	-1	-1	149	-1	-1	-1	15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

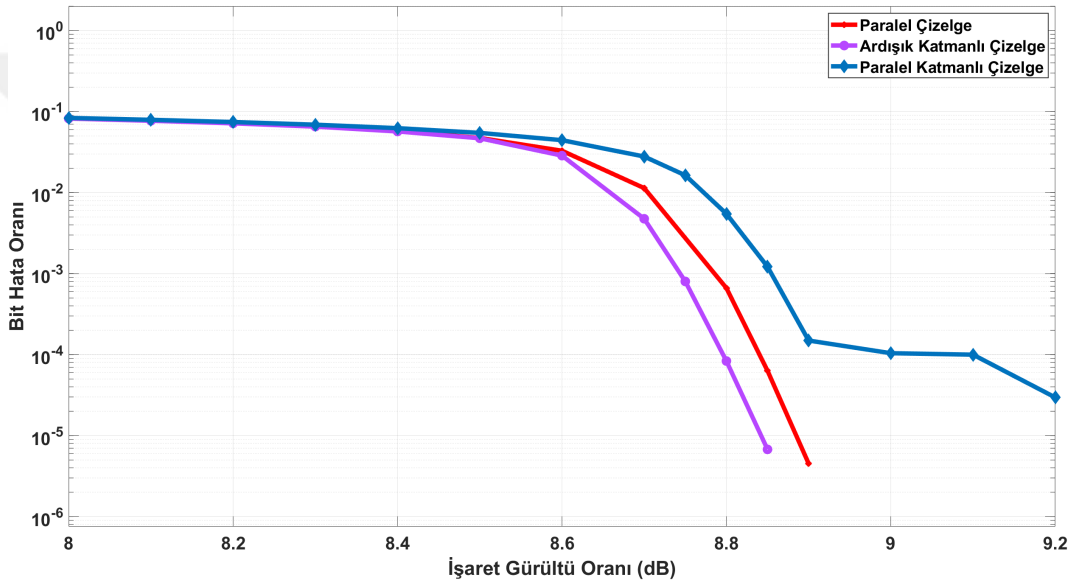


Şekil 4.3 : Tablo 4.1'de gösterilen 5G NR Temel Çizge 1 Matrisinin 384 Yükseltme Çarpanı ve 307 Kaydırma Çarpanı ile Genişletilmiş (1,1) Elemanı



Şekil 4.4 : Tablo 4.1'de gösterilen 5G NR Temel Çizge 1 384 Yükseltme Çarpanı ile Genişletilmiş Matris

biri 384 denetim düğümünden oluşan katmanları birer birer işler. Bunu yaparken eğer tüm katmanların paralel işlendiğini varsayarsak ortak bit düğümlerine erişmeye çalışan katmanların mesajlarının çakışması nedeniyle kod çözücünün başarımı düşer. Bunun yerine, donanımda gerçekleştirilmesi de daha makul olan ardışık katmanlı çizelge düşünülebilir [18,19]. Bu çizelgede, bir katman işlendikten sonra elde edilen mesajlar belleğe yazılır ve sonraki katmanın işlenmesine geçilir. Aynı yineleme içerisinde, önceki katmanın ürettiği taze bilgi yeni katmanda da kullanıldığından bu kod çözme çizelgesinin hata başarımı paralel çizelgeden bile iyidir. İlgili sonuçlar test düzeneğinden bit hata oranının işaret gürültü oranına oranı cinsinden Şekil 4.5'teki gibi elde edilmiştir:

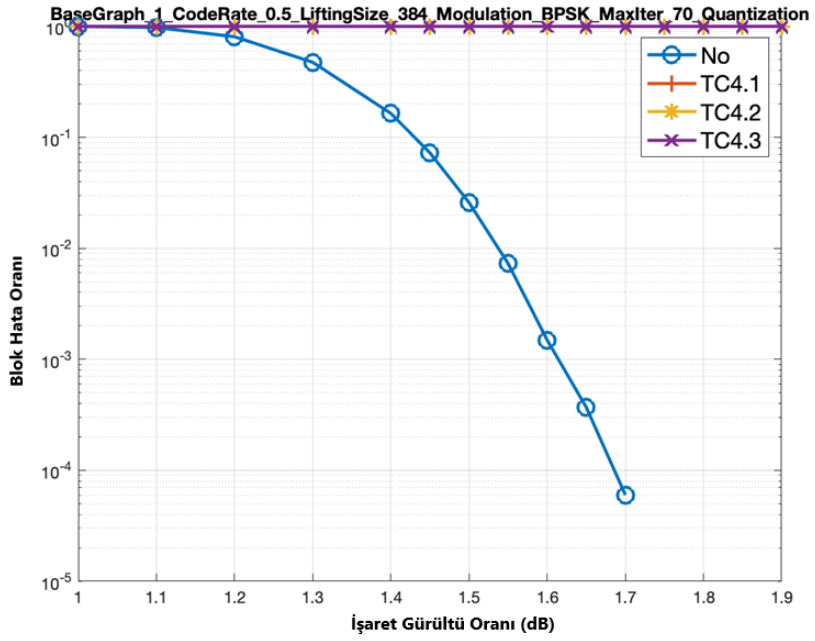


Şekil 4.5 : Paralel, Paralel Katmanlı ve Seri Katmanlı Çizelge Karşılaştırılması

4.3 Kayan Noktalı ve Sabit Noktalı Kod Çözme Karşılaştırılması

LDPC algoritmaları genellikle kayan noktalı olarak belirtilmelerine rağmen donanım üzerinde sabit noktalı olarak tasarlanmaları gerekmektedir [20]. Bu bölümde gerçekleştirilen çalışmaların amacı kod çözücünün FPGA gerçeklemeleri sırasında nasıl bir sabit noktalı veri gösteriminin tercih edilmesi gerektiğini ve nasıl bir başarım beklenebileceğini önceden öngörebilmektir. FPGA gerçekleme sırasında ikiye tümleyen veri gösterim formatı kullanılacaktır. Buna göre TCa.b biçiminde tanımlayacağımız sistem bir sayıyı göstermek için toplam a tane bit kullanacak olup, bu bitlerin b tanesi sayının kesirli kısmını göstermek, bir biti ise sayının işaretini

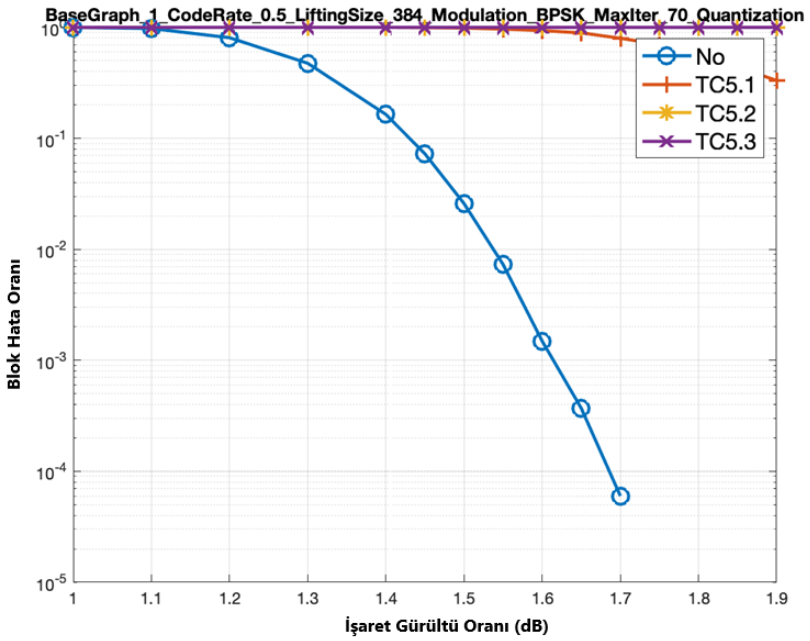
göstermek için kullanılacaktır. Bu da tamsayı kısmı için a-b-1 tane bit bırakmaktadır. Bit ve denetim düğümleri arasındaki mesajların TCa.b gösterimiyle kullanıldığı kod çözücüde bit düğümlerin toplam mesajını göstermek için TCa+2.b gösterimi kullanılacaktır. Böylece daha büyük tam sayılı mesajların da problemsiz gösterilmesi amaçlanmaktadır. Min-toplam algoritmasının bit düğümlerinde gerçekleştirdiği işlemlerde (5G standardına göre) yaklaşık 20 mesaj toplanabildiğinden, bu toplamı sağlıklı gösterebilmek için bu ekstra iki bite izin verilmiştir. Çeşitli gösterim biçimleri için kayan noktalı ve sabit noktalı bilgisayar benzetim sonuçları aşağıda sunulmuş, her birinden sonra ilgili sonuçlar yorumlanmıştır. Bilgisayar benzetimleri sırasında standartta belirtilen birinci temel çizge ele alınmış, kodlama oranı R=1/2 oranına uydurulmuş, Z=384 parametresi ile üretilmiştir. Bilgisayar benzetimleri BPSK modülasyonu ile AWGN kanal üzerinde gerçekleştirilmiş ve kod çözücüye en fazla 70 yineleme yapma hakkı verilmiştir. Sonuçlar blok hata oranının işaret gürültü oranına oranı cinsinden Şekil 4.6'da sunulmuştur.



Şekil 4.6 : 4 Bit Sabit Noktalı Kod Çözme

Şekil 4.6'da görüldüğü üzere toplam a=4 bit ile gösterilmeye çalışılan mesajlar kod çözücüyü çok zayıflatmakta ve kayan noktalı kod çözücüye göre kabul edilemeyecek derecede fazla hataya neden olmaktadır. Kod çözücünün çalışmadığı söylenebilir. Bunun nedeni toplam 4 bitin nasıl dağıtılırsa dağıtılsın mesajları göstermeye

yetmemesidir. Örneğin, TC4.1 gösteriminde gösterilebilecek en büyük sayı +4.5'tir ve bu sayı kod çözücü içerisinde dolaşan mesajları çok ağır biçimde kısıtlamaktadır.



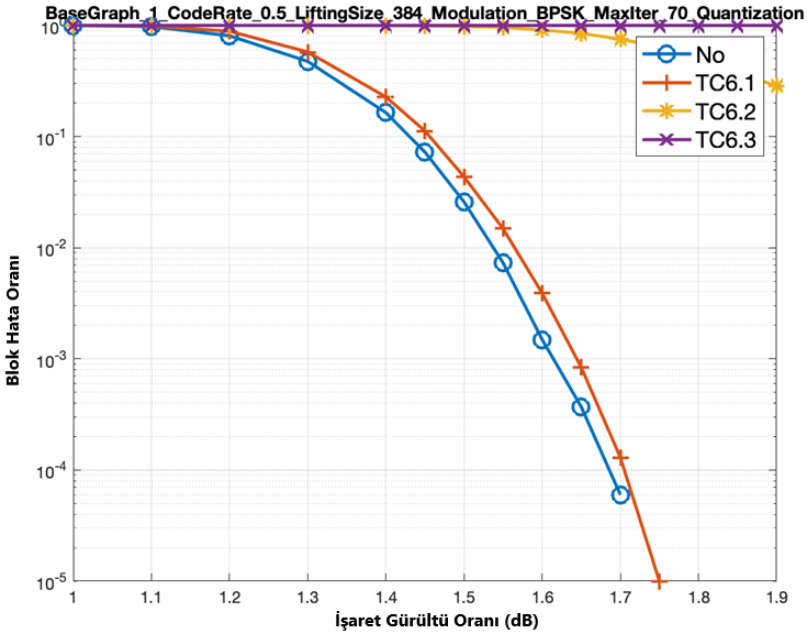
Şekil 4.7 : 5 Bit Sabit Noktalı Kod Çözme

Şekil 4.7'deki TC5.x gösterimine gelindiğinde görüldüğü gibi TC5.1 diğer aile üyelerine göre kod çözücünün bir nebze çalışmasına olanak verse de yine kabul edilemez kayıplar verdirmektedir.

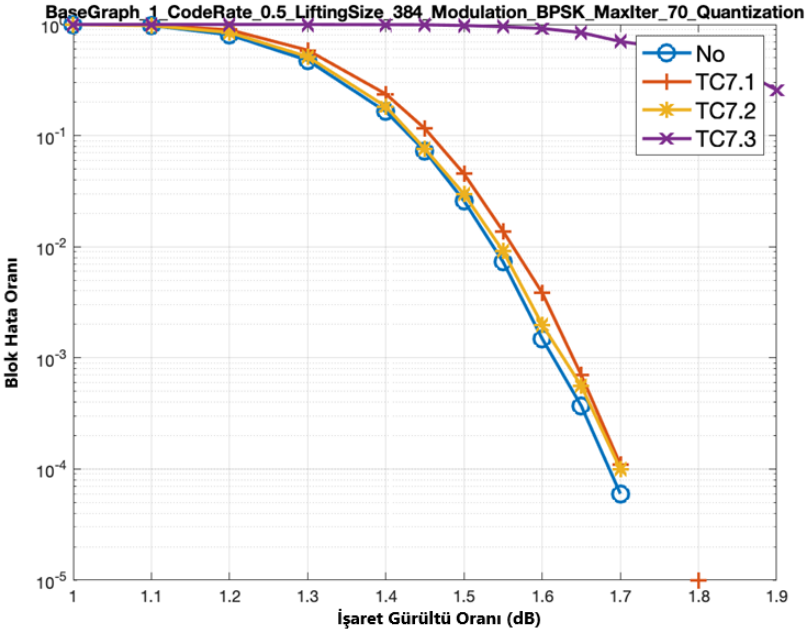
Şekil 4.8'deki TC6.x gösterimine çıktığımızda ilk kez kabul edilebilir başarımlara yaklaştığımızı görebiliriz. TC6.1 gösterimi kayan noktalı kod çözücüye göre yaklaşık 0.05 dB kayıp verse de en azından hata eğrisini takip edebilmektedir. TC6.2 ve TC6.3 gösterimleri eldeki 6 bitin çok fazla kısmını kesirli sayı için kullandıklarından dolayı tamsayı kısmının kısıtlanmasını engelleyememektedirler.

Şekil 4.9'daki TC7.x gösterimlerinde TC7.1 ve TC7.2 gösterimlerinin kayan noktalı sonuçlara yeteri kadar yakın oldukları gözlemlenmektedir. TC7.2 gösteriminin kayan noktaya en yakın sonuçları vermesi bu kod için kod çözücünün 1 işareti biti + 4 tamsayı biti + 2 kesir biti ile belirlenen sınırları beğendiğini göstermektedir. Bu gösterimde gösterilebilecek en yüksek sayı +16.75'tir.

Şekil 4.10'da ve Şekil 4.11'de görülebileceği üzere TC8.x ve TC9.x ailelerinin başarımları neredeyse aynıdır. Önceki TC7.x ailesiyle karşılaştırıldığında artık TC8.2, TC8.3, TC9.2, ve TC9.3 gösterimlerinin yeterli hassasiyete sahip olup kayan

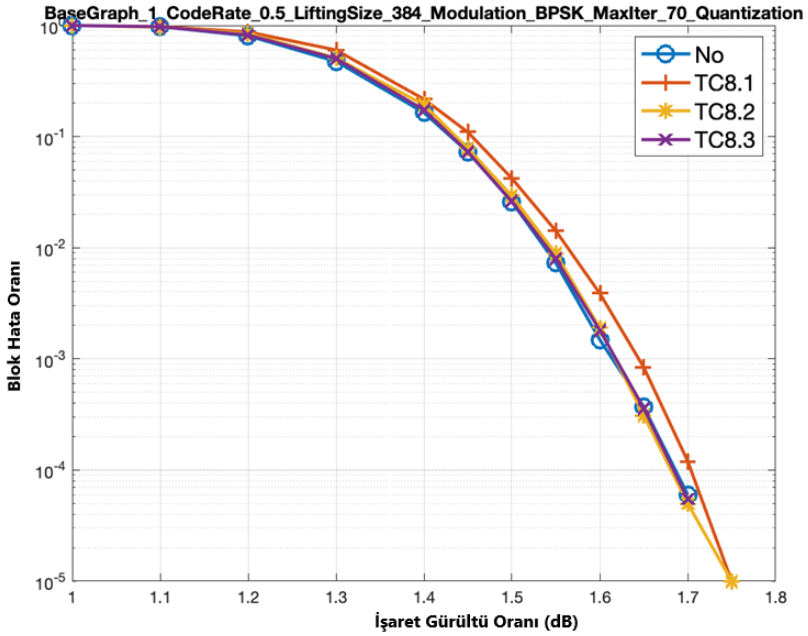


Şekil 4.8 : 6 Bit Sabit Noktalı Kod Çözme

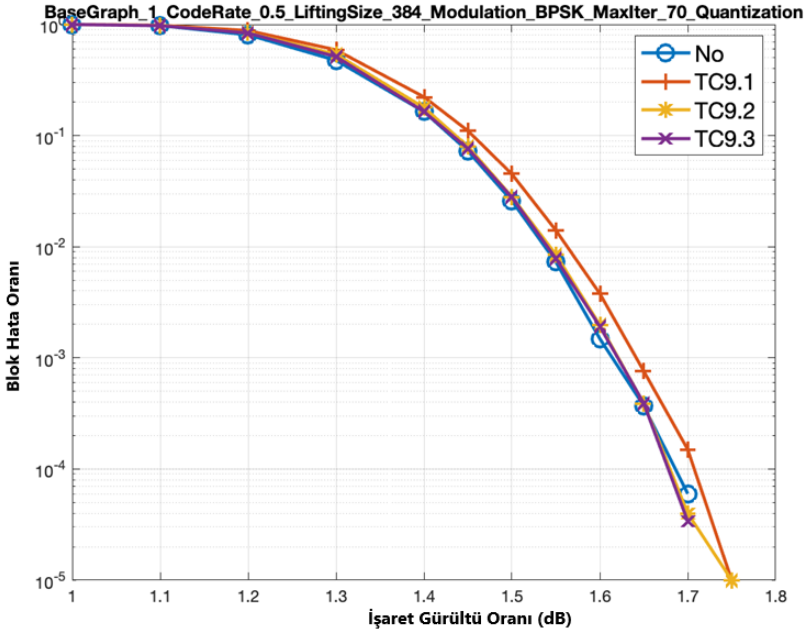


Şekil 4.9 : 7 Bit Sabit Noktalı Kod Çözme

noktalı kod çözücünün başarımı tamamen yakaladığı görülmektedir. TC8.1 ve TC9.1 gösterimlerinde tamsayılar için çok sayıda bit ayrılmış olmasına rağmen kesirli kısım için tek bit ayrılmasının hata eğrisinde bir miktar kaymaya neden olduğu da gözlenmiştir.



Şekil 4.10 : 8 Bit Sabit Noktalı Kod Çözme



Şekil 4.11 : 9 Bit Sabit Noktalı Kod Çözme

Elde edilen bu benzetim sonuçlarına göre donanım gerçekleme sırasında elde bulunacak imkanlara göre TC6.1 veya TC7.2 gösterimlerinden birinin seçilmesinin hata başarımını korumak adına doğru olacağı sonucuna ulaşılmıştır.

5. FPGA ÜZERİNDE KOD ÇÖZÜCÜ TASARIMI

Bu bölümde LDPC kod çözücünün donanıma uygun bir şekilde tasarımının yapılması ve eşlik denetim biriminin FIL ile test edilmesi süreçleri anlatılmıştır. LDPC kod çözücü yapısında bulunan bit düğümü güncellemeleri donanım gerçeklemesinde mesajların toplanmasıyla yapılabildiği için eşlik denetim biriminin içinde tasarlanmıştır ve bu bölümde anlatılan bellek erişim yöntemiyle gerçekleştirilmiştir.

5.1 Literatürdeki Donanım Gerçeklemeleri

LDPC kod çözücüler için literatürde önerilmiş farklı kablosuz haberleşme standartlarına uygun donanım tasarımları bulunmaktadır. Bu tasarımların odaklandıkları ortak noktalar verilerin bellekte saklanması, bellek erişimleri ve paralel çalışma konularıdır. LDPC kod çözücünün girişine gelen verilerin matris yapısına uygun olarak sıralanarak RAM'lerde saklanması gerekmektedir. RAM genişliklerinin ayarlanabilir olmasından faydalanarak birden çok mesaj bir arada saklanabilir ve erişilebilir [21]. Bu sayede paralel çalışan CNU tasarımlarıyla yüksek veri hacmi sağlanabilir. RAM'leri verimli kullanmak için yapılan bu uygulama sonucunda bellek erişimleri için yeni stratejiler kullanılması gerekmektedir. Çözüm olarak literatürdeki çalışmalarda RAM'lere erişirken bit karıştırma ve çembersel bit kaydırma yöntemleri önerilmiştir [22]–[24]. Matristeki bağlantılara göre ROM'larda saklanan bit kaydırma değerleri kullanılarak RAM'lerde birlikte saklanan veriler tek okuma esnasında doğru sıralamayla bit düğümü birimlerine ve denetim düğümü birimlerine dağıtılabilir. İşlenen veriler RAM'lere geri yazılırken ise yine ROM'da saklanan aynı değerlerle ters karıştırma veya ters bit kaydırma yapılarak tekrar doğru yerlere yazılır. Bu okuma ve yazma işlemleri yapılırken RAM erişimlerinde çakışma olmaması için uygun denetim düğümü birimleri ve erişim sıralamaları önerilmiştir [25]. Denetim düğümlerinin gecikmeleri dikkate alınarak çalışmalarının sıralanması gerekmektedir. Bir katmanın çalışması sonrası veriler yazılmadan önce bir sonraki katmandaki okumalarda aynı

adrese erişilmeye çalışılıyorsa çakışmalar oluşur. Bunların sonucunda güncellenmemiş veriler bir sonraki katmana aktarılır ve sonuçlar RAM'de aynı adresin üstüne yazılır. Bu durumu önlemek için erişim sıralamalarının düzenlenmesi gerekir. Farklı LDPC matrisleri ve donanım gerçeklemeleri için erişim sıralamaları değişiklik göstermektedir.

5.2 Donanıma Uygun CNU Yapısı

Literatür üzerine yapılan çalışmalar sonucunda donanımda gerçeklemek için en uygun yapının min-toplam algoritması kullanılarak elde edilebileceğine karar verilmiştir. Bu algoritma hem düşük karmaşıklığa sahip olması hem de sıralı erişime uygun olması açısından FPGA üzerinde çalıştırılmaya uygundur. Sabit noktalı yapılacak tasarım için mesajların 6 bit genişliğinde tutulması modelde yapılan performans testleri sonucunda uygun bulunmuştur. Bu bit genişliğine göre belirlenen parametreler Tablo 5.1'de gösterilmiştir.

Tablo 5.1 : CNU Parametre Listesi

Parametre	Değer	Açıklama
SO_WIDTH	8	SO girişinin bit genişliği
EXT_WIDTH	6	Extrinsic girişinin bit genişliği

Bit düğümlerinden gelen mesajlar SO_WIDTH, denetim düğümlerinden gelen mesajlar ise EXT_WIDTH olarak adlandırılmıştır. Denetim düğümü mesajları, bit düğümlerinde toplandığı için bit düğümlerinde bit genişliği 2 bit daha fazla tanımlanmıştır. Bu değerlerin parametrik tanımlanması sayesinde istenildiği durumda arttırılıp azaltılarak performans-alan kullanımı arasında tercihler yapılabilmektedir.

CNU girişinde bit düğümlerinden ve denetim düğümlerinden gelen mesajları sıralı bir şekilde almaktadır. Anlamlı mesajlar veri geçerli işaretiyle belirtilmektedir. Gelen mesaj bloğunun sonuna gelindiği ise blok sonu işareti ile belirtilmektedir. CNU girişine veri almaya hazır olduğunda dışarıya bir hazır işareti göndermektedir. Sıfırlama sinyali asenkron ve aktif düşük olacak şekilde tasarlanmıştır. CNU girişine gelen bit düğümü ve denetim düğümü mesajlarının her birini karşılık birer SO ve mesaj çıkışı göndermektedir. Bu sayede hem bit düğümünün hem de denetim düğümünün güncellemesi aynı anda yapılabilmektedir. Mesajlar üretilirken

bit düğümü güncellemesi için ayrı bir zaman planlaması yapılmasına ve donanım harcamasına gerek kalmamaktadır.

CNU giriş ve çıkış işaretleri ve açıklamaları Tablo 5.2'de gösterilmiştir.

Tablo 5.2 : CNU Giriş ve Çıkış İşaretleri

İsim	Yön	Genişlik	Açıklama
i_clk	Giriş (I)	1	Saat işareti
i_rstn	Giriş (I)	1	Sıfırlama işareti
i_eof	Giriş (I)	1	Blok sonu işareti
i_valid	Giriş (I)	1	Veri geçerli işareti
i_SO	Giriş (I)	SO_WIDTH	Giriş verisi
i_extrinsic	Giriş (I)	EXT_WIDHT	Giriş verisi
o_SO	Çıkış (O)	SO_WIDTH	Çıkış verisi
o_message	Çıkış (O)	EXT_WIDHT	Çıkış verisi
o_valid	Çıkış (O)	1	Veri geçerli işareti
o_ready	Çıkış (O)	1	CNU hazır işareti

Yüksek veri hacmi elde edebilmek için CNU içindeki işlemler belirli adımlarda paralelleştirilerek tasarlanmıştır. Geçerli ilk giriş çifti CNU girişine geldikten sonra ilk aşama bu iki girişin birbirinden çıkarılarak sonucun çıkarma dizisi olarak adlandırılan dizide tutulmasıdır. İkinci aşamada, birinci aşamadaki işlemden sonra elde edilen sonucun işareti biti işaret dizisinde tutulur. Bütün girişler alınırken bir sayaç tutularak bu dizilerin hangi elemanında tutulacağı sayaca göre belirlenir. Elde edilen değerin mutlak değeri alınır ve bu değer de mutlak değer dizinde tutulur ve böylece ikinci aşama tamamlanmış olur. Üçüncü aşamada mutlak değerler arasından en küçük olan iki tanesini bulmak için sürekli bir kıyaslama yapılır. Her giriş alındığında en küçük mutlak değer elde edilip edilmediğine bakılır ve buna göre yazmaçlar güncellenir. Blok sonu işareti gelene kadar bu şekilde işlemler ve kıyaslamalar yapılmaya devam edilir. 5G NR Temel Çizge 1 için en fazla 19 adet giriş kıyaslanmaktadır. Bu nedenle diziler de 19 elemana sahip olacak şekilde tasarlanmıştır.

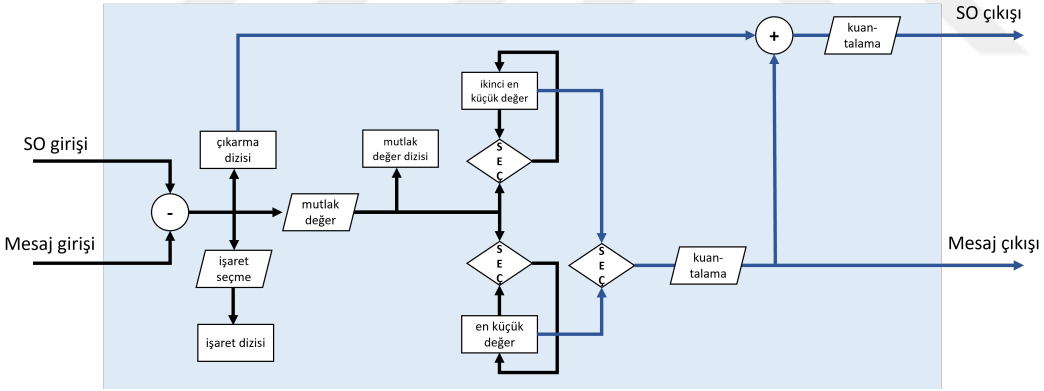
Blok sonu işareti geldikten ve girişlerin işlenmesi tamamlandıktan sonra çıkışların üretilmesi aşamasına geçilir. En küçük değeri tutan yazmaçlardaki değer, her bir giriş için tutulmuş olan mutlak değer dizisiyle kıyaslanarak hangi çıkışın en küçük değere karşılık olan çıkış olduğu tespit edilir. Gelen bütün girişlerin işaretlerinin çarpımı, o

anda çıkışa verilecek veri ile girişin işaret yazmacında tutulmuş işaretiyle çarpılarak çıkışın işaretine karar verilir ve mesaj aşağıdaki biçimde oluşturulur:

- En küçük mutlak değere sahip giriş için oluşturulan mesaj :
- (İşaret biti)(En küçük ikinci mutlak değer)
- Diğer girişler için oluşturulan mesaj :
- (İşaret biti)(En küçük mutlak değer)

Denetim düğümü güncellemesi için üretilecek olan mesaj bir önceki adımdaki sonucun kuantalanmasıyla elde edilmiş olur. Bu mesaj ilk aşama sonucunda elde edilen çıkarma dizisindeki ilgili eleman ile toplanıp kuantalanarak bit düğümü güncellemesi için gerekli mesaj da elde edilmiş olur.

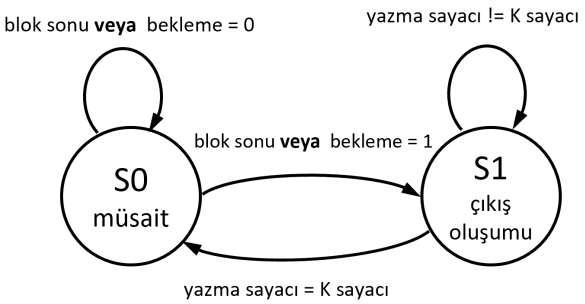
Girişlerin alındığı sıraya uygun olarak oluşturulmuş olan mesajlar çıkışa verilir ve geçerli işaretiyle belirtilir. Bu düzende çalışan CNU donanım yapısı Şekil 5.1'de gösterilmiştir.



Şekil 5.1 : CNU Donanım Yapısı

LDPC matrisindeki katmanlar arasındaki güncellemelerin kesintisiz olarak gerçekleşebilmesi için CNU çıkışa geçerli verileri gönderirken girişten yeni veri kabul edebilmelidir. Aksi durumda ilk katmanın düğümleri güncellendikten sonra ikinci katmanı güncelleyebilmek için tekrar bütün girişlerin alınıp işlenmesinin beklenmesi gerekmektedir. Bu amaçla Şekil 5.2'deki durum makinesi oluşturulmuştur.

Her giriş alındığında değeri arttırılan sayaç yazmacında blok sonu işareti ile birlikte elde edilen değer K sayacının değerini belirler ve çıkışlar üretilirken yazma sayacının



Şekil 5.2 : CNU Durum Makinesi

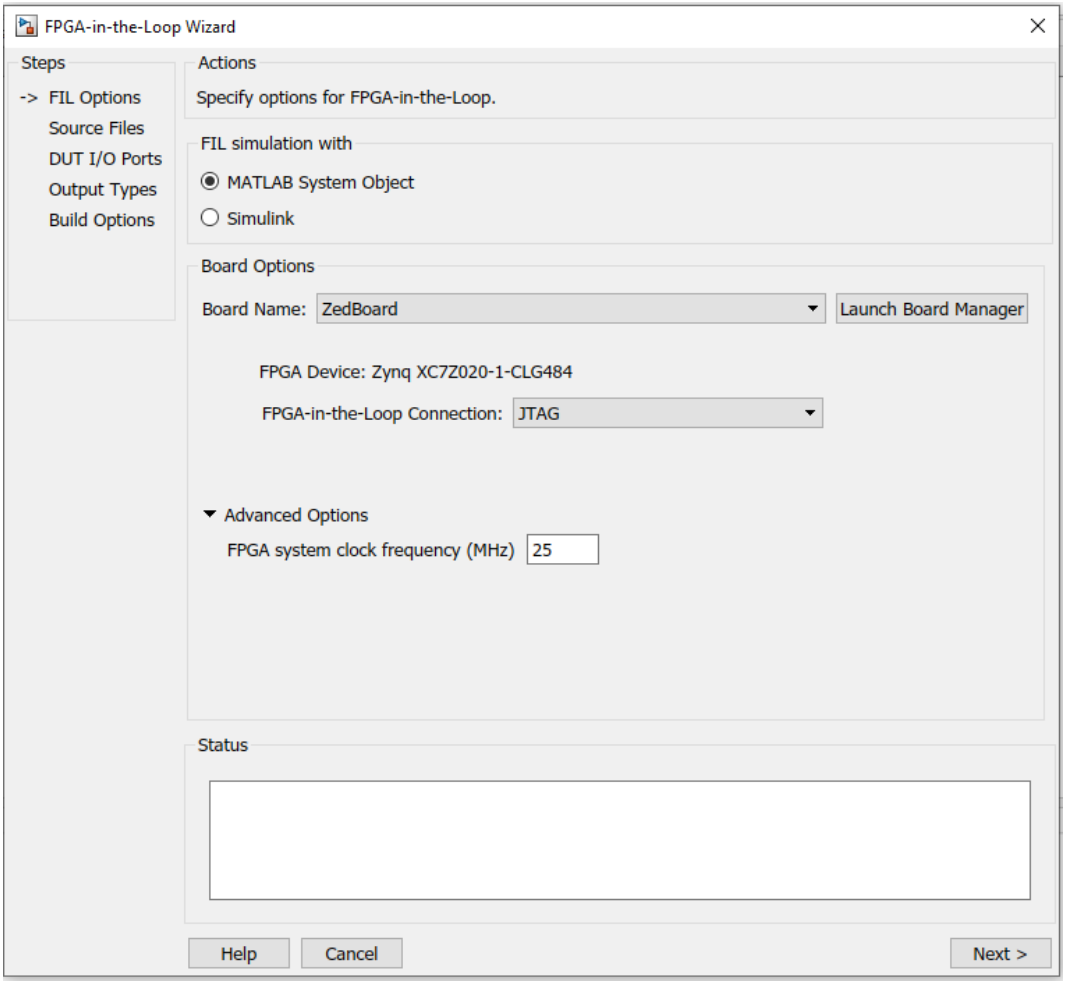
sayacağı değeri hesaplamak için kullanılır. Bu sayede hem dizilerin elemanları bu sayaca göre seçilir hem de uygun sayıda çıkış üretilmiş olur. Çıkış verilirken yeni girişler alındığında bekleme işareti ile üçüncü bir giriş alınması engellenir ve hazır işareti sıfır değerine çekilir. Bekleme durumu bittiğinde hazır işareti verilir ve yeni girişler alınmaya devam edilir.

5G NR LDPC matrislerinin yapısından dolayı CNU girişe gelen blokların uzunlukları farklılık göstermektedir. Örneğin, 5G NR Temel Çizge 1'de en fazla 19, en az 3 elemanlı bloklar gelebilmektedir. Arka arkaya gelen iki bloğun ilkinin eleman sayısı 19, ikincisinin eleman sayısı 3 olduğu durumda daha ilk bloğun çıkışlarının üretilmesi tamamlanmadan ikinci bloğun girişlerinin işlenmesi tamamlanmaktadır. Böyle durumlarda verilerinin kaybolmaması için çıkışlar verilmeye başlamadan ilgili yazmaçlar yedeklenir ve ikinci bloğun girişlerin işlenmesinin bittiğini ve dizileri doldurduğu belirten bir bekleme işareti oluşturulur. Bu işaret aktif olduğu sürece CNU hazır işareti düşürülür. İlk bloğun çıkışları tamamlanıp ikinci bloğun çıkışları verilmeye başlandığında CNU hazır işareti tekrar aktif olur ve girişten geçerli veriler alınabilir. Bu sayede bütün CNU, bütün LDPC matrisleriyle uyumlu çalışabilmesine imkan sağlayacak bir kontrol mekanizmasına sahip olmuş olur.

5.3 FIL Destekli CNU Tasarımı

CNU tasarımı sırasında min-toplam algoritmasının ve zamanlama yapılarının çalışmasını doğrulayabilmek için FIL kullanılmıştır. Bu sayede hızlı bir şekilde doğrulama yapılarak tasarımın FPGA üzerinde çalışması gözlemlenebilmiştir.

FILWizard başlatıldığında Şekil 5.3'te gösterilen ekranda konfigürasyonlar yapılmıştır. FIL testleri için Zedboard kullanılmıştır. MATLAB ortamı ile FPGA arasındaki bağlantı JTAG ile sağlanmıştır. FPGA sistem saat frekans 25 MHz olarak seçilmiştir. FIL benzetimi, daha esnek bir çalışma ortamı sağladığından MATLAB sistem objesi ile yapılmıştır.

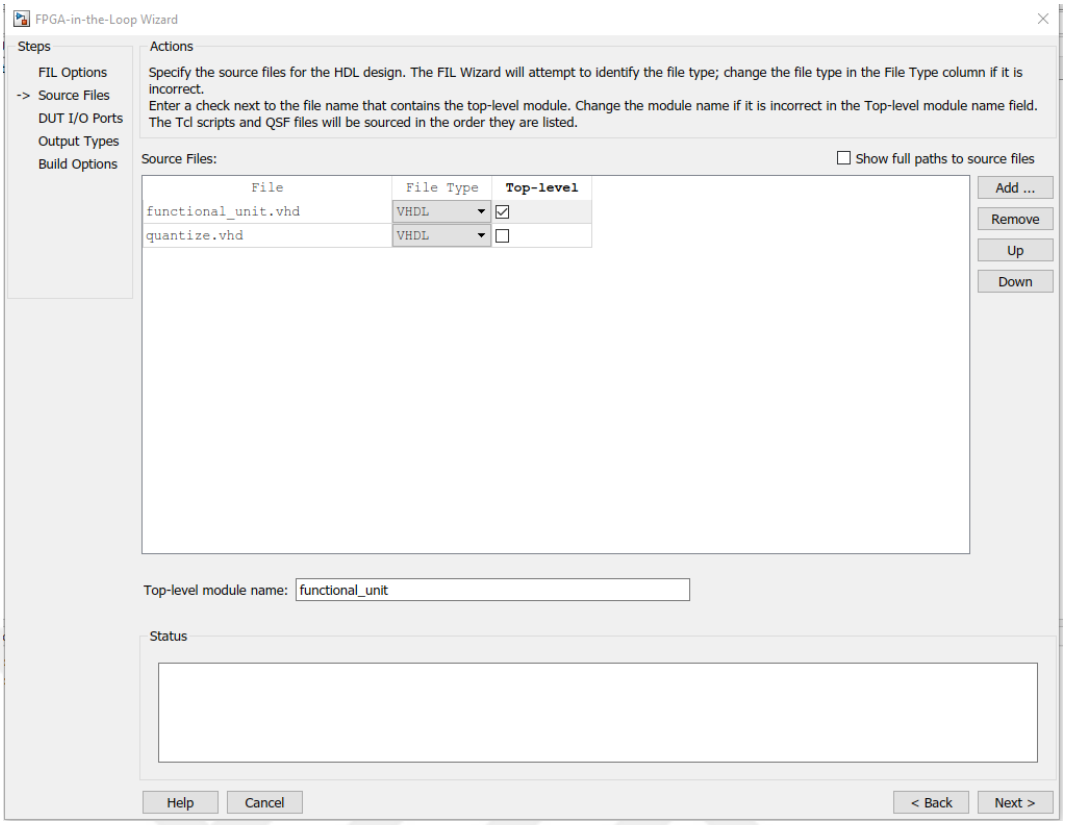


Şekil 5.3 : FIL Birinci Adım

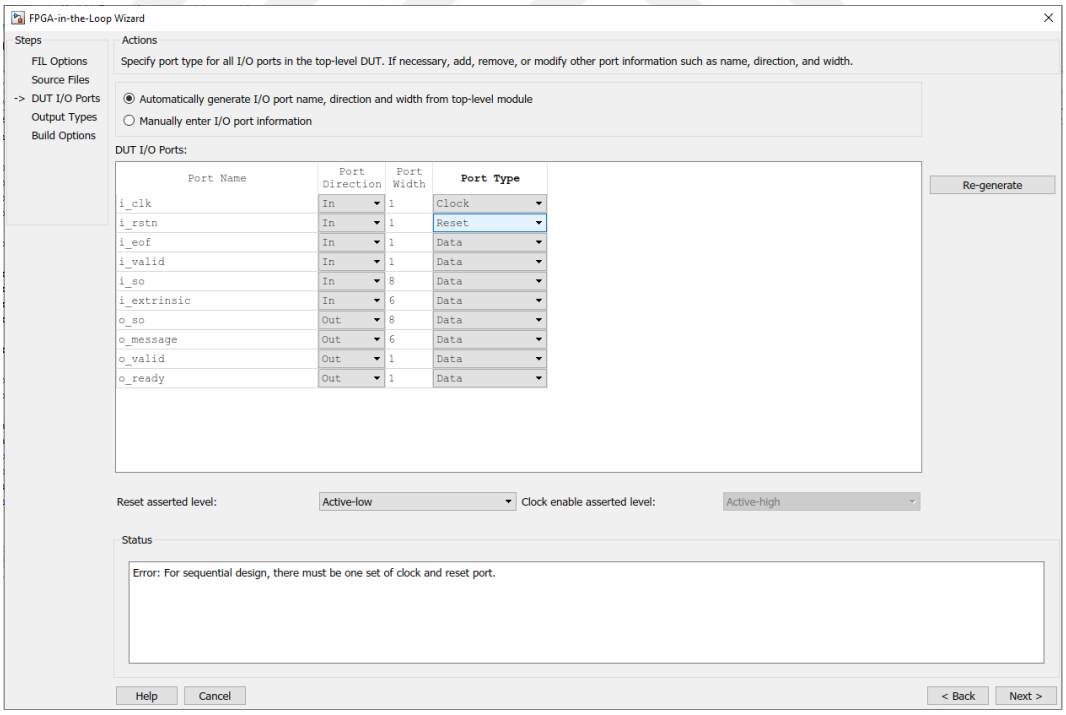
Bir sonraki adımda CNU için yazılmış olan VHDL kodları eklenerek üst seviyede olan dosya Şekil 5.4'te gösterildiği gibi seçilmiştir.

Şekil 5.5'te gösterildiği şekilde CNU kodunun giriş çıkışları için veri tipleri seçilmiştir. Saat işareti, sıfırlama işareti ve veri işareti olarak üç ayrı türde ayarlanmıştır. Sıfırlama işareti tasarımda kullanıldığı şekilde aktif düşük olarak seçilmiştir. Bit genişliklerini FIL tespit ederek göstermiştir.

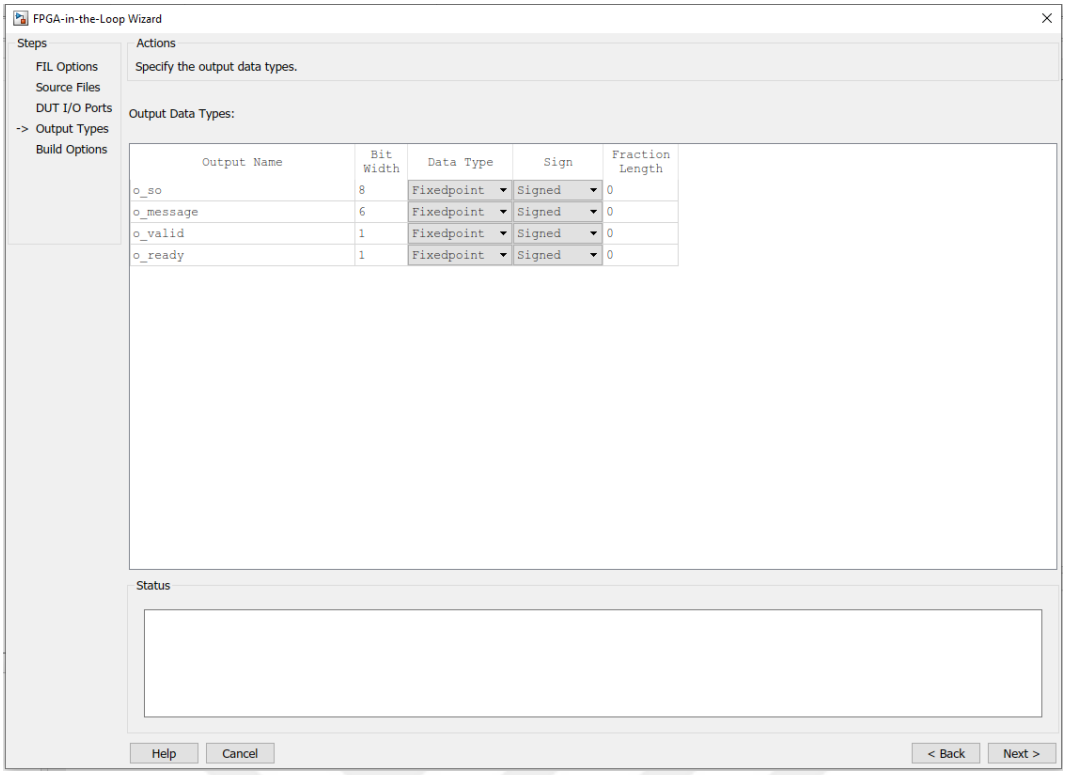
Şekil 5.6'da CNU çıkışlarının veri tipleri ve işaretleri belirlenmiştir. Kesirli kısımlar min-toplam algoritmasında bir değişiklik yaratmadığı için sıfır olarak bırakılmıştır.



Şekil 5.4 : FIL İkinci Adım



Şekil 5.5 : FIL Üçüncü Adım



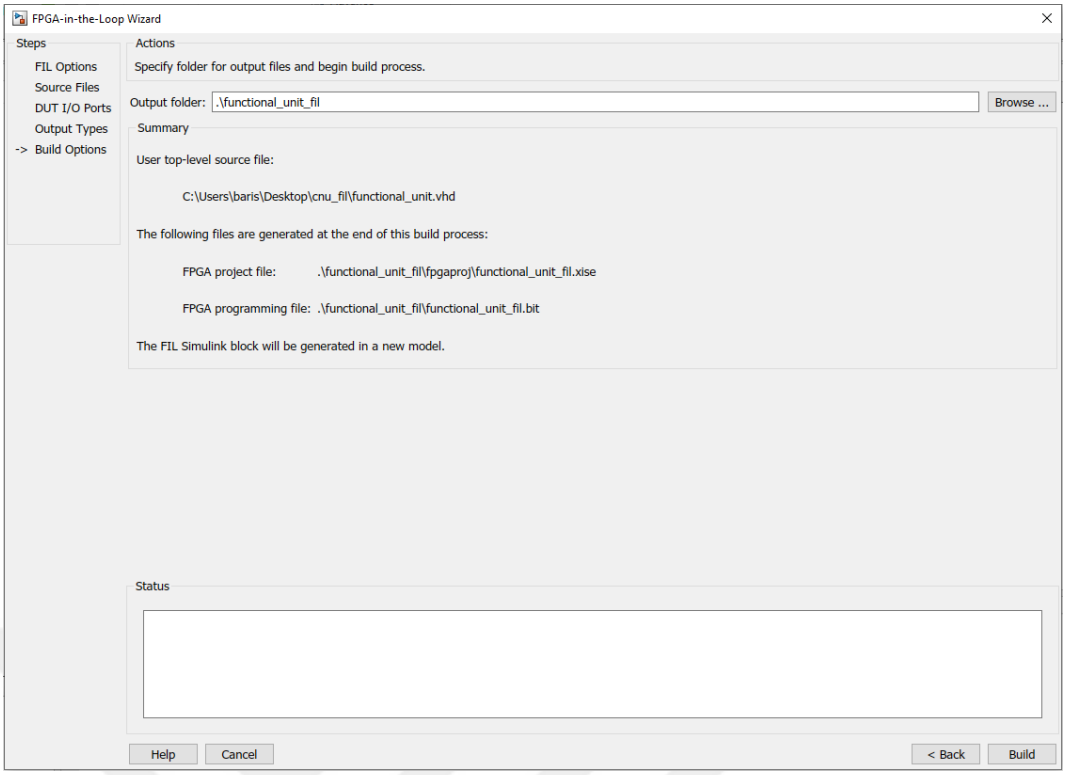
Şekil 5.6 : FIL Dördüncü Adım

Son adımda ise Şekil 5.7'de görüldüğü gibi FIL kodlarının oluşturulacağı dosya seçilerek kodlar üretilmiştir.

Zedboard bilgisayara JTAG kablosuyla bağlanarak MATLAB tarafından algılanmıştır. FPGA programlamak için üretilen kod çalıştırılarak CNU kodu ve JTAG bağlantısı için gerekli bloklar FPGA'e yüklenmiştir. Oluşturulan FIL sınıfı kullanılarak giriş üretmek ve FIL ile test edip modelle kıyaslamak için gereken kod yazılmıştır. CNU FIL kodları EK A'da paylaşılmıştır. Üretilen CNU FIL Sınıfı EK B'de, CNU FPGA Programlama FIL Kodu ise EK C'de paylaşılmıştır.

CNU FIL kodları ile yapılan bir test için uygulanan girişler Şekil 5.8'de gösterilmiştir. Girişler sabit noktalı olarak verilmiştir. İlk sütunda bulunan girişler bit düğümlerinden gelen verilerdir. İkinci sütunda bulunan girişler ise denetim düğümlerinden gelen verilerdir. Bu veriler her saat darbesinde sırayla ve geçerli işaretiyle CNU girişine gönderilmiştir. Son verilerle birlikte blok sonu işareti de gönderilmiştir.

Şekil 5.9'da FIL sonuçları gösterilmiştir. Sonuçlar yine sabit noktalı olarak elde edilmiştir. FPGA üzerinde gerçeklenmiş olan CNU, bit düğümlerinden ve denetim düğümlerinden gelen verilerin farkını almıştır. Bu farkların mutlak değerleri üzerinden



Şekil 5.7 : FIL Beşinci Adım

```
model_and_hardware_inputs =

    -4    -1
    10     2
     3     9

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 8
        FractionLength: 0
```

Şekil 5.8 : FIL Girişleri

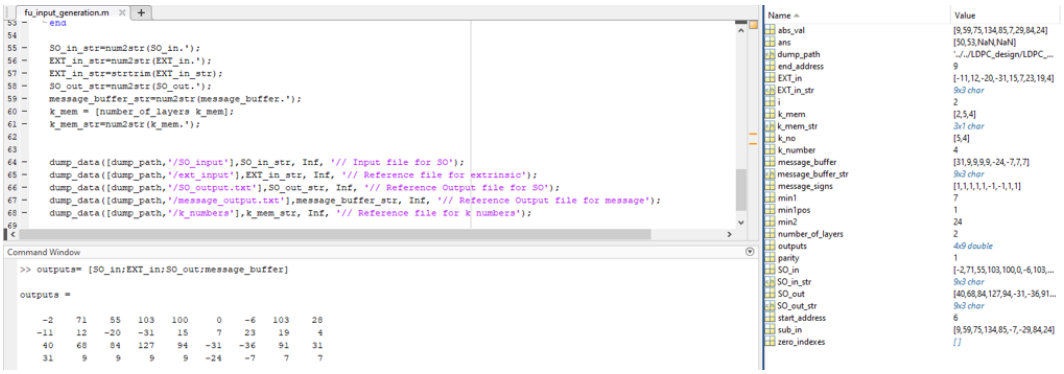
çalışan min-toplam algoritması sonucunda en küçük mesaj 3, ikinci en küçük mesaj ise 6 olarak belirlenmiştir. Girişlerin işaretlerine göre pozitif veya negatif olarak belirlenen mesajlar üçüncü ve dördüncü sütunda gösterilmiştir. Üçüncü sütundaki mesajlar FIL ile FPGA üzerinden gelen mesajlar, dördüncü sütundaki mesajlar ise MATLAB modelinin sonucunda elde edilen mesajlardır. Bu mesajlar en başta elde edilen bit düğümlerinin ve denetim düğümlerinin farklarıyla toplanarak bit düğümü güncellemesi de yapılır. Bu sonuçlar donanım için birinci sütunda, model için ikinci sütunda gösterilmiştir.

```
model_hardware_comparison =

    -9    -9    -6    -6
    11    11     3     3
    -9    -9    -3    -3

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 8
        FractionLength: 0
```

Şekil 5.9 : FIL Sonuçları

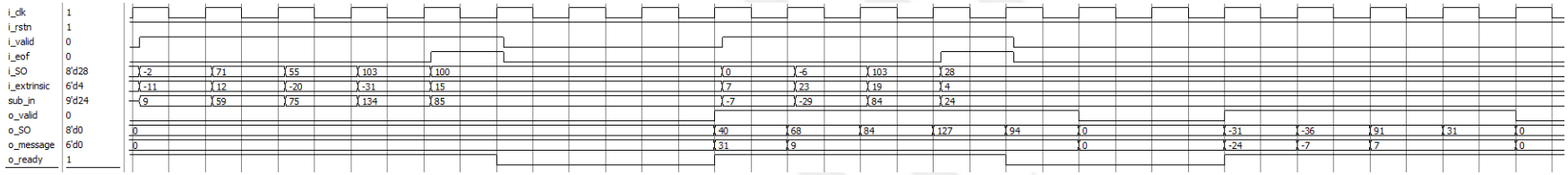
MATLAB ekranında görülen Şekil 5.8'deki ve Şekil 5.9'daki girişler ve çıkışlar incelendiğinde donanım üzerinde gerçeklenmiş CNU ile MATLAB ortamındaki modelin birbirleriyle bit-uyumlu olarak çalıştığı doğrulanabilmektedir. Bu sayede hem model tasarımı hem de model ve donanım doğrulama aynı ortamda gerçekleştirilebilmiştir.

FIL ile CNU doğrulaması yapıldıktan sonra klasik yöntemlerle de doğrulama yapmak için bir VHDL test dosyası oluşturulmuştur. Oluşturulan bu dosya EK D'de paylaşılmıştır. VHDL test dosyası ile yapılan testlerde MATLAB modelinden dosyaya yazılan girişler kullanılmıştır. MATLAB modeli sonuçları Şekil 5.10'da gösterilmiştir. Donanım benzetimi sonuçları Şekil 5.11'de gösterilmiştir.



Şekil 5.10 : MATLAB Ortamında CNU Modeli Sonuçları

Şekil 5.10'da gösterilen MATLAB ortamında giriş ve çıkış verileri gözlemlenmiş ve dosyalara yazılmıştır. Böylece donanım için gereken test dosyalarında kaynak olarak bu dosyalar verilebilmektedir. MATLAB ortamı ile donanım benzetimi arasındaki bağlantı bu şekilde kurulmaktadır. Bu yöntemle yapılan donanım benzetimi sonuçları Şekil 5.11'de verilmiştir.



Şekil 5.11 : CNU Donanım Benzetimi Sonuçları

Fonksiyonel doğrulaması tamamlanan CNU, Vivado ortamında 200 MHz saat frekansında sentezlenerek alan kullanımı incelenmiştir. Sonuçlar Tablo 5.3'te gösterilmiştir.

Tablo 5.3 : CNU Sentez Sonuçları

İsim	LUT	Yazmaç	F7 Mux	F8 Mux	Bonded IOB	BUFGCTRL
CNU	413	645	48	10	34	1

5.4 LDPC Kod Çözücü Üst Seviye Mimarisi

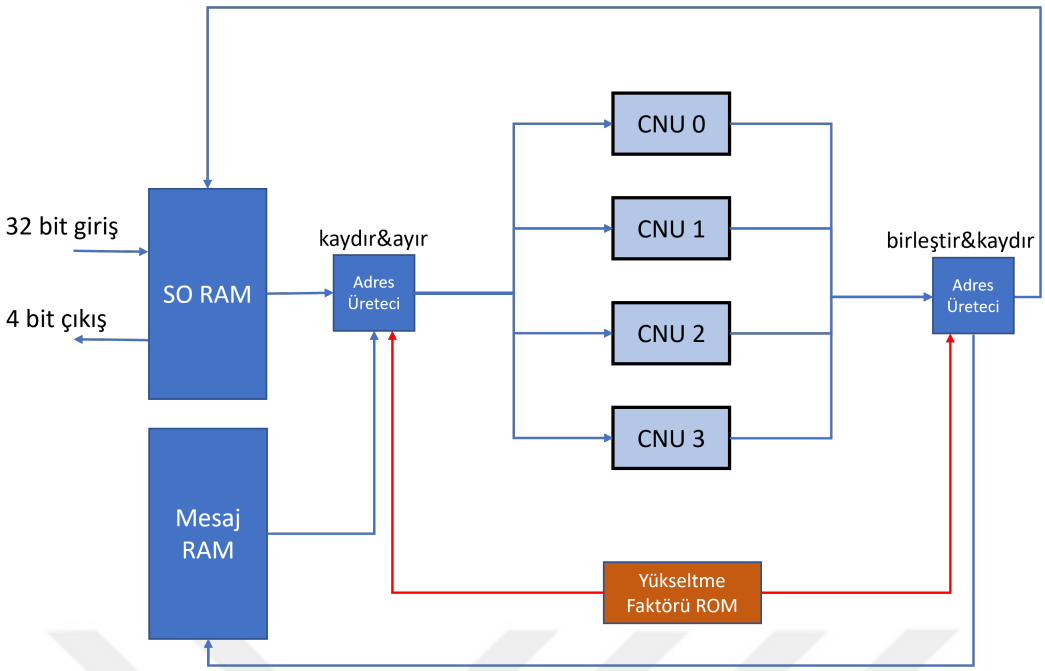
CNU tasarımı, FIL ile doğrulandıktan sonra LDPC Kod Çözücü içerisinde kullanılmak üzere bir üst seviye mimarisi tasarlanmıştır. Bu tasarım için 5G NR LDPC Temel Çizge 1, yükseltme çarpanı 4 olan matris seçilmiştir. Bu matristeki -1 dışında herhangi bir değere sahip olan bütün elemanlar bir bağlantıyı işaret etmektedir. Eğer bütün bağlantılar aynı anda işlenmek istenirse bağlantı sayısı kadar CNU, FPGA üzerinde gerçeklenmelidir. Temel Çizge 1, yükseltme çarpanı 4 olan matriste bu bağlantı sayısı 1264'tür, en büyük yükseltme çarpanında ise bu sayı 121344 olabilmektedir. Bu kadar fazla sayıda CNU gerçeklemek alan kullanımı açısından Tablo 5.3'te bir tane CNU'nun alan kullanımı incelendiğinde uygulanabilir değildir. Bu nedenle LDPC matrisi katmanlı bir yapıda incelenerek her bir satır seri katmanlar halinde işlenmiştir. Yükseltme çarpanı 4 olduğu için temel çizgenin her bir satırı 4×4 boyutunda bir matris oluşturmaktadır. Bu 4×4 boyutlu matrisin paralel olarak işlenmesi mümkündür. Bu amaçla FPGA üzerinde sadece 4 adet paralel çalışan CNU gerçeklenmiştir. Bit düğümlerindeki ve denetim düğümlerinde verileri depolamak için RAM kullanılmıştır. CNU'ların paralel çalışabilmesi için hafıza erişimi önemli bir yer tutmaktadır. Tasarlanan mimaride kanaldan gelen veriler dörtlü gruplar halinde birleştirilerek RAM satırlarına yazılmıştır. Bu sayede aynı saat darbesinde okunan veriler ayrılarak CNU girişlerine gönderilebilmektedir. LDPC matrisindeki 68 sütundan her biri RAM'de bir satıra denk gelmektedir. 4 veri birleştirildiği için yükseltme çarpanı 4 ile oluşan matristeki 272 sütun 68 satıra sığdırılabilmektedir. LDPC matrisi kaydırılmış birim matrislerden oluştuğu için veriler RAM'den okunduktan sonra CNU'lara gönderilirken bu birim matrislere göre bit kaydırma işlemi yapılması gerekmektedir. Bit düğümünde saklanan veriler 8 bit olduğu için 4 tanesinin birleştirilmesiyle 32 bitlik bir veri elde

edilmektedir. Bu verinin 2 kere kaydırılması gerekiyorsa aşağıda gösterildiği biçimde bit kaydırma işlemi uygulanır:

- RAM'den okunan veri:
- 01010101000000001111111**10101010**
- LDPC matrisinde ilgili sütundaki sayı: 2
- Bit kaydırma miktarı:
- 2*8 = 16
- Bit kaydırma işleminin sonucu:
- 1111111**10101010**0101010100000000
- Birinci CNU girişi:
- 11111111
- İkinci CNU girişi:
- **10101010**
- Üçüncü CNU girişi:
- 01010101
- Dördüncü CNU girişi:
- 00000000

Denetim düğümlerinden gelen mesajlar ise aynı şekilde altılı bit grupları halinde kaydırılırlar. RAM'leri ve CNU'ları içeren üst seviye mimarisi Şekil 5.12'de gösterilmiştir.

LDPC kod çözücünün donanım gerçeklemesinde adres üretimi için seri katmanlı çizelge uygulanmıştır. Temel çizgenin her katmanı 4 satır, 272 sütun içeren bir matrise karşılık gelmektedir. Her katmandaki 4 satır, 4 CNU ile paralel olarak işlenmiştir. Bir



Şekil 5.12 : LDPC Kod Çözücü Üst Seviye Mimarisi

katmandaki her girişin okunması ve yazılması seri olarak yapılmıştır. Bu 4 paralel işlemde yapılacak okuma ve yazma işlemlerinin sayısı katman içinde birbirine eşittir. 5G NR LDPC Temel Çizge 1 için her bir katmandaki bağlantı sayısı Tablo 5.4'te ve Tablo 5.5'te verilmiştir.

Temel çizgede her bağlantının kendi kaydırma miktarı bulunmaktadır. Bu miktara göre bit kaydırma işlemi yapılarak CNU girişlerine veriler gönderilmektedir. Yüksek veri hacmi sağlamak için bir katman bütün okumalarını bitirdiğinde yazmasını bitirmesini beklemeden bir sonraki katmanın okumaları yapılmaya başlanmaktadır. Katmanlar arasındaki geçişler sırasında 5G NR standardında verilen düzende çalışıldığında bir önceki katman henüz RAM'i güncellemeden bir sonraki katmanın aynı adrese erişmeye çalışması ve CNU'lara güncellenmemiş verileri göndermesi durumuyla karşılaşılmaktadır. Bu çalışmada, RAM erişimlerindeki çakışmaları çözmek için farklı bir erişim sıralaması önerilmiştir. Katman içindeki okumalarda adreslere erişim sıralaması sonucu etkilememektedir. Bu sayede, adresler özgün bir biçimde sıralanarak güncellenmemiş verilerin bir sonraki katmanda okunmalarının önüne geçilmiştir. Önerilen erişim sıralaması Tablo 5.6'da ve Tablo 5.7'de verilmiştir.

Tablo 5.4 : İlk 21 Katman İçin Temel Çizge 1 Bağlantı Sayıları

Katman Numarası	Bağlantı Sayısı
1	19
2	19
3	19
4	19
5	3
6	8
7	9
8	7
9	10
10	9
11	7
12	8
13	7
14	6
15	7
16	7
17	6
18	6
19	6
20	6
21	6

Önerilen erişim sıralaması sayesinde beşinci katman dışındaki bütün katmanlardaki çakışmalar önlenebilmektedir. Beşinci katmanda sadece üç okuma yapıldığı için de sıralama yapmaya yetmemektedir. Bu değerler ROM yapılarında tutularak iterasyon sırasında sırayla erişilmektedir. Çakışmalar yoğunluklu olarak ilk 21 katman arasında gerçekleştiği için erişim sıralaması özellikle bu katmanlarda uygulanmıştır.

Denetim düğümü verileri her katman için ayrı üretildiğinden bit düğümlerinin verileri gibi birbirlerinin üzerine yazılmamaktadır. Bu nedenle ayrı ayrı olarak RAM'de 316 satırda tutulmaktadır. Her bağlantının oluşturduğu mesajlar için bir adres üretilmektedir. Bilgi bitlerinin ilk iki grubu kanaldan gönderilmediği için bu bitler yerine sıfır değeri RAM'lere yazılır. LDPC kod çözücü çalışırken birinci iterasyon denetim düğümlerinde henüz mesajlar bulunmamaktadır. İlk iterasyonda bu mesajlar yerine sıfır değeri verilerek sonuçlar RAM'de tutulur. Sonraki iterasyonlar adres üretilerek RAM'e erişilir ve denetim düğümü mesajları CNU'lara iletilir. İterasyonlar tamamlandığı bit düğümlerinde güncellenen verilerin işaret bitlerine bakılarak sert

Tablo 5.5 : Kalan Katmanlar İçin Temel Çizge 1 Bağlantı Sayıları

Katman Numarası	Bağlantı Sayısı
22	6
23	5
24	5
25	6
26	5
27	5
28	4
29	5
30	5
31	5
32	5
33	5
34	5
35	5
36	5
37	5
38	4
39	5
40	5
41	4
42	5
43	4
44	5
45	5
46	4

karar mekanizması çalıştırılır. Karar verilen bitler 4 bit birleştirilerek LDPC kod çözücü çıkışına verilir ve geçerli işaretiyle belirtilir.

LDPC kod çözücünün üst seviye kodlarını doğrulamak için bir VHDL test dosyası oluşturulmuştur. Oluşturulan bu dosya EK E'de paylaşılmıştır. VHDL test dosyası ile yapılan testlerde MATLAB modelinden dosyaya yazılan girişler kullanılmıştır. LDPC kod çözücü iki iterasyon için ayarlanarak çalıştırılmıştır. Uygulanan giriş dosyası EK F'de paylaşılmıştır. MATLAB modeli ve donanım sonuçları birbiriyle bit uyumlu olarak elde edilmiş ve Tablo 5.8'de gösterilmiştir. Donanım benzetimi sonuçları ve zamanlamaları Şekil 5.13, Şekil 5.14 ve Şekil 5.15'te gösterilmiştir.

Şekil 5.13'te LDPC kod çözücünün girişine gelen verileri RAM'e yazma aşaması gösterilmiştir. Girişler bir saat darbesi geciktirilerek RAM'e yazma sinyali ayarlanmaktadır ve blok sonu işareti gelene kadar yazılmaya devam etmektedir.

Tablo 5.6 : İlk 21 Katman İçin Önerilen Erişim Sıralaması

Katman Numarası	Bit Düğümü Sıralaması
1	1, 3, 6, 10, 16, 20, 7, 11, 14, 19, 21, 4, 12, 13, 17, 22, 23, 24, 2
2	5, 1, 3, 6, 10, 16, 20, 8, 9, 15, 18, 25, 4, 12, 13, 17, 22, 23, 24
3	2, 5, 1, 3, 6, 10, 16, 20, 8, 9, 15, 18, 7, 11, 14, 19, 21, 26, 25
4	4, 2, 5, 1, 12, 13, 17, 22, 23, 8, 9, 15, 18, 7, 11, 14, 19, 21, 26
5	1, 2, 27
6	4, 13, 1, 17, 22, 23, 28, 2
7	7, 11, 12, 1, 14, 18, 19, 21, 29
8	2, 5, 8, 9, 1, 15, 30
9	4, 2, 13, 17, 20, 1, 22, 23, 25, 31
10	11, 12, 2, 14, 18, 19, 21, 32, 1
11	3, 5, 8, 2, 9, 15, 33
12	1, 13, 17, 22, 2, 23, 24, 34
13	11, 1, 12, 14, 19, 35, 2
14	4, 8, 1, 21, 24, 36
15	13, 16, 17, 1 18, 22, 37
16	2, 11, 14, 19, 26, 38, 1
17	4, 12, 21, 23, 39, 2
18	1, 15, 17, 18, 22, 40
19	2, 13, 14, 19, 20, 41
20	1, 8, 9, 11, 42, 2
21	4, 10, 12, 23, 43, 1

Şekil 5.14'te LDPC kod çözücünün çıkışlarının oluşturulması aşaması gösterilmiştir. RAM'de güncellenmiş mesajların işaretlerine bakılarak sert karar mekanizması çalıştırılmaktadır. Dört adet mesaj bir arada saklandığı için tek okuma yapılarak her seferinde dört bit çıkış verilebilmektedir.

Şekil 5.15'te LDPC kod çözücü iterasyonları sırasında RAM adreslerinin değişimi gösterilmiştir. Tablo 5.6'da önerilen erişim sıralamasına uygun olarak adresleme yapıldığı gözlemlenmektedir. RAM'deki ilk adres sıfırdan başladığı için Tablo 5.6'daki ve Tablo 5.7'deki adreslerin bir eksiği biçiminde adresleme yapılmaktadır. RAM'de okuma yapmak bir saat darbesi sürdüğü için CNU girişine geçerli işareti verilmeden 1 saat darbesi öncesinde adresler oluşturulmaktadır.

LDPC kod çözücü üst seviye giriş ve çıkış işaretleri ve açıklamaları Tablo 5.9'da gösterilmiştir.

LDPC kod çözücü iki iterasyon yaparak çalıştırıldığında ilk geçerli girişi almasından sonra geçerli çıkış vermesine kadar 1219 saat darbesi geçmiştir. Tek iterasyon yaptığı

Tablo 5.7 : Kalan Katmanlar İçin Önerilen Erişim Sıralaması

Katman Numarası	Bit Düğümü Sıralaması
22	2, 6, 17, 21, 22, 44
23	1, 13, 14, 18, 45
24	2, 3, 11, 19, 46
25	1, 4, 5, 12, 23, 47
26	2, 7, 8, 15, 48
27	1, 3, 5, 16, 49
28	2, 7, 9, 50
29	1, 5, 20, 22, 51
30	2, 15, 19, 26, 52
31	1, 11, 14, 25, 53
32	2, 8, 23, 26, 54
33	1, 13, 15, 25, 55
34	2, 3, 12, 22, 56
35	1, 8, 16, 18, 57
36	2, 7, 13, 23, 58
37	1, 15, 16, 19, 59
38	2, 14, 24, 60
39	1, 10, 11, 13, 61
40	2, 4, 8, 20, 62
41	1, 9, 18, 63
42	2, 4, 10, 19, 64
43	1, 5, 25, 65
44	2, 17, 19, 26, 66
45	1, 8, 10, 23, 67
46	2, 7, 11, 68

durumda kod çözme işlemi 655 saat darbesi sürmektedir. LDPC kod çözücünün çıkışına verdiği toplam bilgi biti sayısı I ile gösterilirse

$$I = 22 \times Z \tag{5.1}$$

şeklinde ifade edilebilir. Denklem 5.1'de yükseltme çarpanı Z ile gösterilmiştir. Tasarlanan LDPC kod çözücü üst seviyesi mimarisi ile elde edilebilecek veri hacmi

$$VeriHacmi = \frac{22 \times Z \times f}{655 + (564 \times (IterasyonNo - 1))} \tag{5.2}$$

biçimde belirlenebilir. Denklem 5.2'de veri hacmi hesaplanırken saat frekansı f, LDPC kod çözücünün yaptığı toplam iterasyon sayısı İterasyonNo ile ifade edilmiştir. Karşılaştırma yapabilmek amacıyla 400 MHz saat frekansında, kodlama oranı 0.324

Tablo 5.8 : LDPC Kod Çözücü Model ve Donanım Benzetimi Sonuçları

Satır Numarası	Sonuç Bitleri
1	1100
2	1000
3	0110
4	1000
5	1100
6	1111
7	1001
8	0011
9	1110
10	1101
11	1001
12	1100
13	0100
14	0011
15	0111
16	1110
17	0100
18	0110
19	1011
20	1010
21	0110
22	1101

Tablo 5.9 : LDPC Kod Çözücü Üst Seviye Giriş ve Çıkış İşaretleri

İsim	Yön	Genişlik	Açıklama
i_clk	Giriş (I)	1	Saat işareti
i_rstn	Giriş (I)	1	Sıfırlama işareti
i_eof	Giriş (I)	1	Blok sonu işareti
i_valid	Giriş (I)	1	Veri geçerli işareti
i_ready	Giriş (I)	1	Hazır işareti
i_data	Giriş (I)	32	Giriş verisi
o_eof	Çıkış (O)	1	Blok sonu işareti
o_valid	Çıkış (O)	1	Veri geçerli işareti
o_ready	Çıkış (O)	1	Kod Çözücü hazır işareti
o_data	Çıkış (O)	4	Çıkış verisi

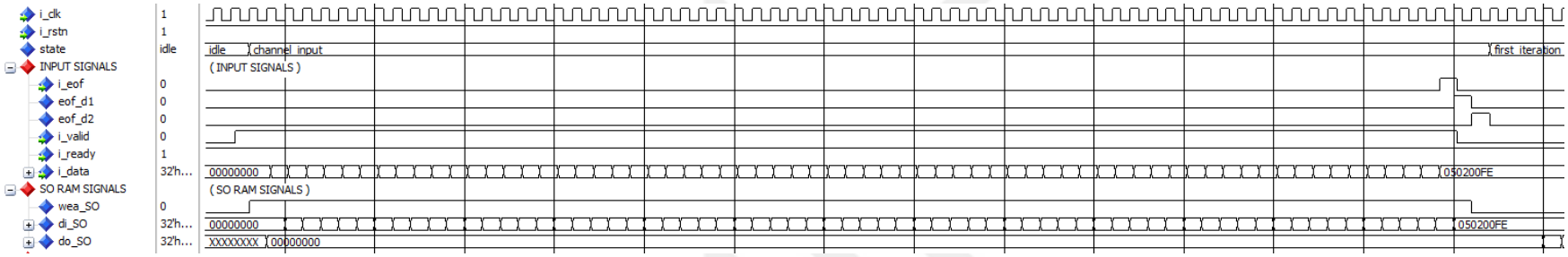
ve 8 iterasyon seçilerek elde edilebilecek veri hacimlerinin [26] ile kıyaslaması Tablo 5.10'da verilmiştir. [27] bütün kod çözme zincirinin veri hacmini gösterdiği için 268 MHz saat frekansında, verilen en yüksek kodlama oranında ve 8 iterasyon için olan sonuçlar paylaşılmıştır. Yükseltme çarpanı 4 için gerçekleme yapılmıştır. Yükseltme çarpanının artması ile Şekil 5.12'de blok diyagramı gösterilen gerçeklemede paralel

çalışan CNU sayısı artmaktadır. Frekansın düşmesine sebep olacak kritik yola herhangi bir ek gelmeyecektir. Bu sebeple, frekans değeri korunarak yükseltme çarpanının dörtten büyük olduğu diğer gerçeklemeler için Denklem 5.2 kullanılarak veri hacmi hesabı yapılmıştır.

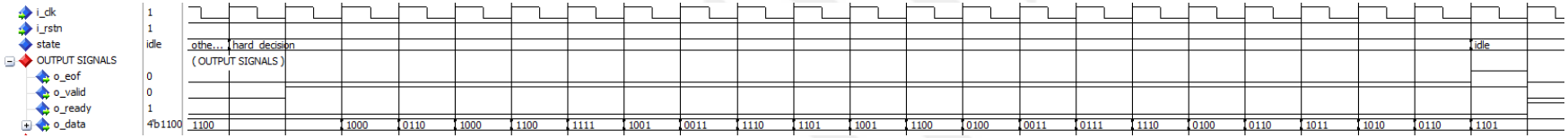
Tablo 5.10 : LDPC Kod Çözücü Veri Hacmi Kıyaslaması

	Yükseltme Çarpanı	Veri Hacmi (Gbps)
Bu çalışma	4	0.0076
	12	0.0229
	32	0.0612
	64	0.1224
	128	0.2447
	192	0.3670
	256	0.4894
	384	0.7341
[26]	32	0.149
	64	0.265
	128	0.424
	256	0.432
	384	0.417
[27]	12	0.027
	192	0.430
	384	0.696

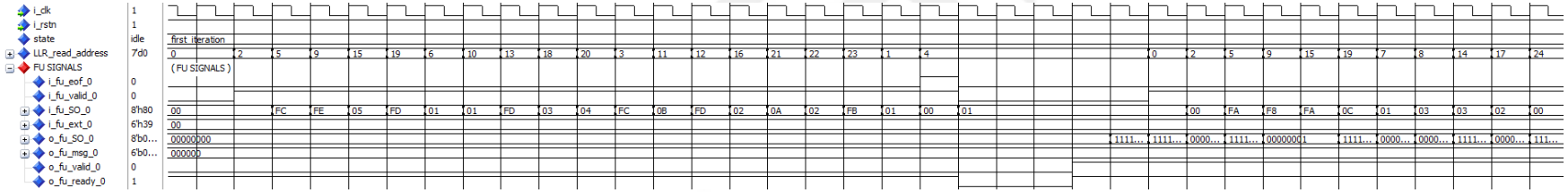
Tablo 5.10'da görüldüğü üzere FIL destekli bir şekilde tasarlanan CNU ile oluşturulan mimari yükseltme çarpanının büyük olduğu durumlarda yüksek veri hacimlerine ulaşabilmektedir. Şekil 5.16'da görüldüğü gibi 240'tan küçük yükseltme çarpanlarında [26] gerçeklemesinden düşük veri hacimlerine sahip olmasına rağmen 240 yükseltme çarpanından itibaren daha yüksek veri hacmine ulaşılabilmektedir. 192'den küçük ve 288'den büyük yükseltme çarpanlarında [27] gerçeklemesinden daha yüksek veri hacmine ulaşılabilmektedir.



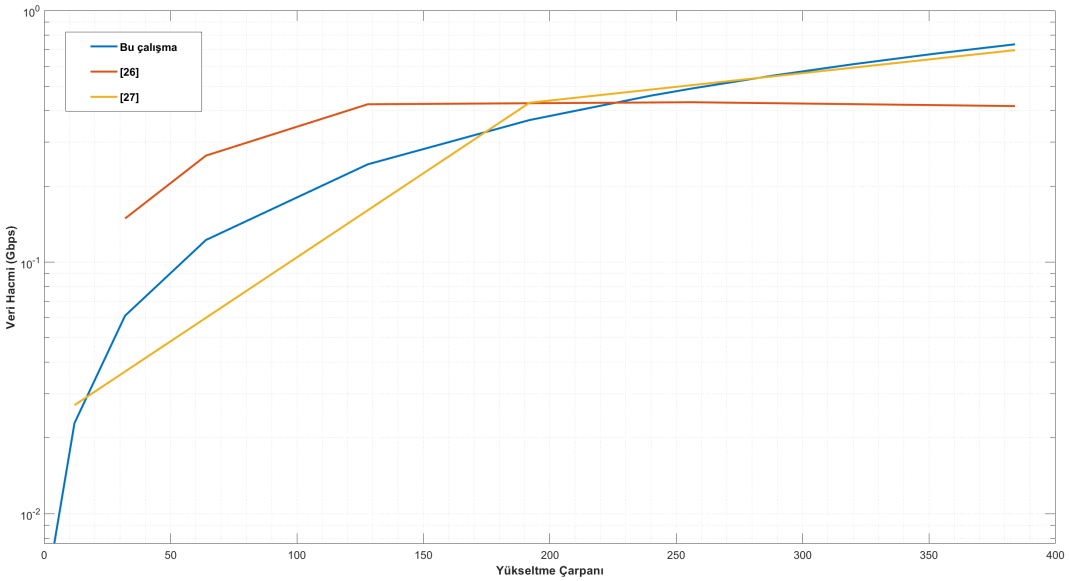
Şekil 5.13 : LDPC Kod Çözücüde Verilerin Alınması



Şekil 5.14 : LDPC Kod Çözücüde Çıkışların Oluşturulması



Şekil 5.15 : LDPC Kod Çözücü Adreslerin Oluşturulması



Şekil 5.16 : LDPC Kod Çözücü Gerçeklemelerinde Yükseltme Çarpanına Bağlı Olarak Veri Hacminin Değişimi

6. SONUÇLAR

Tez kapsamında 5G NR standardına uygun çalıştırılabilmek üzere CNU ve LDPC kod çözücü tasarımı yapılmıştır. Tasarlanan CNU, FIL ile donanım üzerinde doğrulanmıştır. Bu sayede doğrulama sonuçları MATLAB ortamında kolay bir şekilde gözlemlenebilmiş ve MATLAB modeli ile karşılaştırılabilmiştir. Yüksek veri hacmi ile çalışabilmesi için paralelleştirme çalışmaları yapılmıştır.

Donanım üzerinde gerçeklenecek kod çözücü için uygun kod çözme algoritmaları ve çizelgeleri hakkında bilgi edinilmiştir. Farklı algoritmaların performansları ve karmaşıklıkları kıyaslanarak gerçeklemeye uygunlukları belirlenmiştir. Gerçeklemek için karar verilen algoritma min-toplam olarak seçilmiştir. Hem hata performansı hem de karmaşıklık açısından incelendiğinde en uygun algoritma olarak ortaya çıkmaktadır. Kod çözme çizelgeleri arasında FPGA üzerinde çalışmaya en uygun olanı seri-sıralı çizelge olarak belirlenmiştir. Paralel çizelgeyi gerçeklemek için tam paralellik gerekmektedir. Alan kullanımı açısından bu çizelge çok fazla ihtiyaç yaratması açısında tercih edilmemiştir. Paralel-sıralı çizelge ise bellek erişimleri açısından sorun yaratmaktadır ve güncel verilere erişimi kısıtlamaktadır. Seri-sıralı çizelge ile hem yüksek paralellik hem de bu tezde yapılan sıralama sayesinde bellek çakışmalarına yol açmadan bellek erişimleri sağlanabilmiştir.

5G NR Temel Çizge 1 yükseltme çarpanı 4 için yapılan üst seviye mimari tasarımı ile bütün 5G LDPC matrislerinin gerçeklenebilmesi için uygun bir yapı oluşturulmuştur. Bu yapıda elde edilen paralellik sayesinde büyük yükseltme çarpanlarıyla birlikte artan yüksek veri hacimlerine ulaşılabilmiştir. Tasarlanan CNU, sadece 5G NR standardındaki matrislerle değil, bütün LDPC matrisleriyle çalışabilmesi için esnek bir yapıda tasarlanmıştır. Girişindeki veri geçerli işareti ve blok sonu işareti sayesinde istenilen uzunluktaki blokları işleyebilir ve hazır sinyali ile zamanlamasını ayarlayabilir. Böylece farklı standartlarda kullanılan LDPC kod çözücüler için temel blok olarak kullanılmaya uygun hale getirilmiştir.

Çalışmalar sonucunda ortaya çıkan CNU tasarımı geliştirmeye açıktır. Hata performansına odaklanarak min-toplam algoritması üzerine kurulmuş olan dengelenmiş min-toplam ve zayıflatılmış min-toplam gibi daha gelişmiş algoritmaların gerçeklenmesine çalışılacaktır. Bu geliştirmeler sadece CNU üzerinde yapılacağı için FIL ile doğrulanarak üst seviye mimarisini değiştirmeden LDPC kod çözücü tasarımına eklenebilecektir. Model üzerinde özelleştirmeler yapılarak FIL ile oluşturulan geliştirme ve doğrulama ortamı kullanılacaktır.

Bu çalışmada,

- 5G standardına uygun bir şekilde çalıştırılabilmek üzere CNU ve LDPC kod çözücü tasarımı yapılmıştır.
- FPGA üzerinde ve MATLAB modeli ile karşılaştırmalı doğrulama ortamı oluşturulmuştur.
- Donanıma uygun algoritmalar ve sabit noktalı bit genişlikleri belirlenmiştir.

Gelecek çalışmalarda,

- Dengelenmiş min-toplam ve zayıflatılmış min-toplam gibi algoritmaların gerçeklenmesi,
- FIL ile doğrulanarak üst seviye mimarisini değiştirmeden LDPC kod çözücü tasarımına eklenmesi hedeflenmektedir.

KAYNAKLAR

[1] **Marchand, C.** (2010). *Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards*, Université de Bretagne Sud, `https://hal.archives-ouvertes.fr/tel-01151985`.

[2] **Ryan, W.E.** *ve diğerleri* (1998). A turbo code tutorial.

[3] **Berrou, C.**, **Glavieux, A. ve Thitimajshima, P.** (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1, *Proceedings of ICC '93 - IEEE International Conference on Communications*, cilt 2, s.1064–1070 vol.2.

[4] **Gallager, R.G.** (1963). *Low-density parity-check codes*, Ph.D. dissertation, MIT, Cambridge, MA.

[5] **URL-1**. `https://www.mathworks.com/help/hdlverifier/ref/hdlverifier.filsimulation-system-object.html`, erişim tarihi: 06.05.2022.

[6] **Tanner, R.** (1981). A recursive approach to low complexity codes, *IEEE Transactions on Information Theory*, *27*(5), 533–547.

[7] **3GPP** (2022). NR; Multiplexing and channel coding, **Technical Specification (TS) 38.212**, 3rd Generation Partnership Project (3GPP), `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214`, version 17.1.0.

[8] **Olmos, P.**, **Liu, Y. ve Mitchell, D.** (2021). Low-Density Parity-Check (LDPC) Codes for 5G Communications, s.1–23.

[9] **Hartenstein, R.W.** (1987). *Hardware description languages*, North-Holland.

[10] **Ciletti, M.D.** (2011). *Advanced Digital Design with the Verilog HDL Second Edition*, Pearson.

[11] **Brown, S.D.**, **Francis, R.J.**, **Rose, J. ve Vranesic, Z.G.** (2012). *Field-programmable gate arrays*, cilt180, Springer Science & Business Media.

[12] **Finley, T.** Two's Complement (2000), *URL https://www. cs. cornell. edu/~ tomf/notes/cps104/twoscomp. html*.

[13] **URL-2**. https://www.mathworks.com/help/hdlverifier/ug/verify-hdl-implementation-of-pid-controller-using/-fpga-in-the-loop.html, erişim tarihi: 06.05.2022.

[14] **Zadek, P.**, **Koczor, A.**, **Golek, M.**, **Łukasz Matoga ve Penkala, P.** (2015). Improving Efficiency of FPGA-in-the-Loop Verification Environment, *IFAC-PapersOnLine*, *48*, 180–185.

[15] **Asakura, S.**, **Tanahashi, M.**, **Nakamura, M.**, **Okano, M. ve Tsuchida, K.** (2016). Hardware implementation of spatially coupled LDPC codes for broadcasting, *2016 International Symposium on Information Theory and Its Applications (ISITA)*, s.211–215.

[16] **Bae, J.**, **Abotabl, A.**, **Lin, H.P.**, **Song, K.B. ve Lee, J.** (2019). An overview of channel coding for 5G NR cellular communications, *APSIPA Transactions on Signal and Information Processing*, *8*.

[17] **Chang, Y.M.**, **Casado, A.I.V.**, **Chang, M.C.F. ve Wesel, R.D.** (2008). Lower-Complexity Layered Belief-Propagation Decoding of LDPC Codes, *2008 IEEE International Conference on Communications*, s.1155–1160.

[18] **Hocevar, D.** (2004). A reduced complexity decoder architecture via layered decoding of LDPC codes, *IEEE Workshop onSignal Processing Systems, 2004. SIPS 2004.*, s.107–112.

[19] **Marchand, C.**, **Conde-Canencia, L. ve Boutillon, E.** (2013). HIGH-SPEED CONFLICT-FREE LAYERED LDPC DECODER FOR THE DVB-S2, -T2 AND -C2 STANDARDS, *2013 IEEE Workshop on SIgnal Processing Systems (SISP'2013)*, France, s.1–6, https://hal.archives-ouvertes.fr/hal-00848822.

[20] **Shaker, S.W.** (2011). DVB-S2 LDPC finite-precision decoder, *13th International Conference on Advanced Communication Technology (ICACT2011)*, s.1383–1386.

[21] **Chen, X.**, **Kang, J.**, **Lin, S. ve Akella, V.** (2011). Memory System Optimization for FPGA-Based Implementation of Quasi-Cyclic LDPC Codes Decoders, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *58*(1), 98–111.

[22] **Loi, K. ve Ko, S.B.** (2011). Improvements on the design and implementation of DVB-S2 LDPC decoders, *Computers Electrical Engineering*, *37*, 1137–1146.

[23] **Gomes, M.**, **Falcao, G.**, **Silva, V.**, **Ferreira, V.**, **Sengo, A. ve Falcao, M.** (2007). Flexible Parallel Architecture for DVB-S2 LDPC Decoders, *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, s.3265–3269.

[24] **Petrović, V.L.**, **Marković, M.M.**, **Mezeni, D.M.E.**, **Saranovac, L.V. ve Radošević, A.** (2020). Flexible High Throughput QC-LDPC Decoder With Perfect Pipeline Conflicts Resolution and Efficient Hardware Utilization, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *67*(12), 5454–5467.

[25] **Marchand, C.**, **Dore, J.B.**, **Conde-Canencia, L. ve Boutillon, E.** (2009). Conflict resolution for pipelined layered LDPC decoders, *2009 IEEE Workshop on Signal Processing Systems*, s.220–225.

[26] **Xilinx**. *LogiCORE IP Product Guide LDPC Encoder/Decoder v2.0 for Vivado Design Suite*, `https://docs.xilinx.com/v/u/en-US/pb052-ldpc`, erişim tarihi: 19.05.2022.

[27] **Intel**. *5G LDPC-V Intel® FPGA IP User Guide Updated for Intel® Quartus® Prime Design Suite: 22.1 IP Version: 4.0.0*, `https://www.intel.com/content/www/us/en/docs/programmable/683670/22-1-4-0-0/about-the-5g-ldpc-v.html`, erişim tarihi: 26.05.2022.

EKLER

EK A : CNU FIL Kodları
EK B : CNU FIL Sınıfı
EK C : CNU FPGA Programlama FIL Kodu
EK D : CNU VHDL Test Dosyası
EK E : LDPC Kod Çözücü Üst Seviye VHDL Test Dosyası
EK F : LDPC Kod Çözücü Üst Seviye Benzetimi Girişleri

EK A : CNU FIL Kodları

```
% FIL INSTANTIATION
cnu= functional_unit_fil;

% INPUT GENERATION
so_1= fi(-4,1,8,0);
so_2= fi(10,1,8,0);
so_3= fi(3,1,8,0);
so_pad= fi(0,1,8,0);

so_in = [so_pad; so_1; so_2; so_3; so_pad; so_pad; so_pad; so_pad;
so_pad; so_pad; so_pad];
so_model = [ so_1; so_2; so_3];

ext_1= fi(-1,1,6,0);
ext_2= fi(2,1,6,0);
ext_3= fi(9,1,6,0);
ext_pad= fi(0,1,6,0);

ext_in = [ext_pad; ext_1; ext_2; ext_3;ext_pad;ext_pad;ext_pad;
ext_pad;ext_pad;ext_pad;ext_pad];
ext_model = [ ext_1; ext_2; ext_3];

eof = [0; 0; 0; 1;0;0;0;0;0;0;0];

eof = fi(eof,0,1,0);

valid = [0; 1; 1; 1;0;0;0;0;0;0;0];

valid = fi(valid,0,1,0);

inputs = [so_in ext_in valid eof];

% MATLAB MODEL
start_address = 1;
end_address    =3;

sub_in(start_address:end_address)  = ...
so_model(start_address:end_address) - ...
ext_model(start_address:end_address);
abs_val(start_address:end_address) = ...
abs(sub_in(start_address:end_address));

[min1,min1pos] = min(abs_val(start_address:end_address));

min2 = min(abs_val([start_address:(start_address+min1pos-2) ...
(start_address+min1pos):end_address]));

message_signs(start_address:end_address) = ...
sign(sub_in(start_address:end_address));
zero_indexes=find(~message_signs(start_address:end_address));
```

```
message_signs(start_address+zero_indexes-1)=1;
parity = prod(message_signs(start_address:end_address));

message_buffer(start_address:end_address) = min1;
message_buffer(start_address-1+min1pos) = min2;

message_buffer(start_address:end_address) = ...
parity*message_signs(start_address:end_address).* ...
message_buffer(start_address:end_address);
message_buffer(start_address:end_address) = ...
fi(message_buffer(start_address:end_address),1,6,0);

SO_out(start_address:end_address)= ...
sub_in(start_address:end_address) ...
+message_buffer(start_address:end_address);
SO_out(start_address:end_address) = ...
fi(SO_out(start_address:end_address),1,8,0);

%REAL HARDWARE FIL
[o_so, o_message, o_valid, o_ready] = cnu(eof,valid,so_in,ext_in);
outputs = [o_so o_message o_valid o_ready];

%RESULTS
SO_output_comparison = [o_so(8:10) transpose(SO_out)];
message_output_comparison = ...
[o_message(8:10) transpose(message_buffer)];
model_hardware_comparison = ...
[SO_output_comparison message_output_comparison];
model_and_hardware_inputs = ...
[inputs(2,1) inputs(2,2); inputs(3,1) ...
inputs(3,2); inputs(4,1) inputs(4,2)];
```

EK B : CNU FIL Sınıfı

```
classdef (StrictDefaults)functional_unit_fil <
hdlverifier.FILSimulation
%functional_unit_fil is a filWizard generated
%    class used for FPGA-In-the-Loop
%    simulation with the 'functional_unit' DUT.
%    functional_unit_fil connects MATLAB
%    with a FPGA and cosimulate with it by
%    writing inputs in the FPGA and reading outputs from the FPGA.
%
%    MYFIL = functional_unit_fil
%
%    Step method syntax:
%
%    [out1, out2, ...] = step(MYFIL, in1, in2, ...)
%    connect to the FPGA,
%    write in1, in2, ... to the FPGA and read out1, out2, ... from
%    the FPGA
%
%    functional_unit_fil methods:
%
%    step         - See above description for use of this method
%    release      - Allow property value and input characteristics
%                   changes, and release connection to FPGA board
%    clone        - Create functional_unit_fil object with same
%                   property values
%    isLocked     - Locked status (logical)
%    programFPGA - Load the programming file in the FPGA
%
%    functional_unit_fil properties:
%
%    DUTName                    - DUT top level name
%    InputSignals               - Input paths in the HDL code
%    InputBitWidths             - Width in bit of the inputs
%    OutputSignals              - Output paths in the HDL code
%    OutputBitWidths            - Width in bit of the outputs
%    OutputDataTypes            - Data type of the outputs
%    OutputSigned               - Sign of the outputs
%    OutputFractionLengths      - Fraction lengths of the outputs
%    OutputDownsampling         - Downsampling factor and phase
%                                 of the outputs
%    OverclockingFactor         - Overclocking factor of the hardware
%    Connection                 - Parameters for the connection
%                                 with the board
%    FPGAVendor                 - Name of the FPGA chip vendor
%    FPGABoard                  - Name of the FPGA board
%    FPGAProgrammingFile        - Path of the Programming file
%                                 for the FPGA
%    ScanChainPosition          - Position of the FPGA in the
%                                 JTAG scan chain
%
```

```
%    File Name: functional_unit_fil.m
%    Created: 05-May-2022 21:04:42
%
%    Generated by FIL Wizard

%#codegen

    properties (Nontunable)
        DUTName = 'functional_unit';
    end

    methods
        function obj = functional_unit_fil

            %THE FOLLOWING PROTECTED PROPERTIES
            %ARE SPECIFIC TO THE HW DUT
            %AND MUST NOT BE EDITED
            %(RERUN THE FIL WIZARD TO CHANGE THEM)
            obj.InputSignals = char('i_eof','i_valid',...
            'i_so','i_extrinsic');
            obj.InputBitWidths = [1,1,8,6];
            obj.OutputSignals = char('o_so','o_message',...
            'o_valid','o_ready');
            obj.OutputBitWidths = [8,6,1,1];
            obj.Connection = char('JTAG','libmwrtiostream_xjtag',
            'FPGAInstr=000010;FPGAInstr2=000011;FPGAInstr3=100010;...
            FPGAInstr4=100011;InstrLenBefore=4;InstrLenAfter=0;...
            TckFrequency=66.000000','');
            obj.FPGAVendor = 'Xilinx';
            obj.FPGATool   = 'Xilinx Vivado';
            obj.FPGABoard = 'ZedBoard';
            obj.ScanChainPosition = 2 ;

            %THE FOLLOWING PUBLIC PROPERTIES ARE RELATED
            %TO THE SIMULATION
            %AND CAN BE EDITED WITHOUT RERUNING THE FIL WIZARD
            obj.OutputSigned = [true,true,false,false];
            obj.OutputDataTypes = char('fixedpoint',...
            'fixedpoint','fixedpoint','fixedpoint');
            obj.OutputFractionLengths = [0,0,0,0];
            obj.OutputDownsampling = [1,0];
            obj.OverclockingFactor = 1;
            obj.FPGAProgrammingFile = ...
            'C:\Users\baris\Desktop\cnu_fil\...
            functional_unit_fil\functional_unit_fil.bit';
        end
    end
end
```

EK C : CNU FPGA Programlama FIL Kodu

```
function functional_unit_programFPGA
%functional_unit_programFPGA is a filWizard
%    generated function used to load the
%    programming file for the 'functional_unit'
%    HDL in the 'ZedBoard' FPGA board.
%
%    File Name: functional_unit_programFPGA.m
%    Created: 05-May-2022 21:04:42
%
%    Generated by FIL Wizard

filProgramFPGA('Xilinx Vivado', ...
'C:\Users\baris\Desktop\cnu_fil\functional_unit_fil\ ...
functional_unit_fil.bit',2);


end
```

EK D : CNU VHDL Test Dosyası

```
library ieee;
use ieee.Std_Logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

library ldpc_lib;
use ldpc_lib.functional_unit_comp_pack.all;

library common_lib;
use common_lib.ldpc_constants_pack.all;

entity tb_functional_unit is
  generic(
    dirname    : string  := "./testcases/cnu_test"
  );
end entity;

architecture arch of tb_functional_unit is

file ftype_SO_in    : TEXT open READ_MODE is dirname&"/SO_input";
file ftype_ext_in   : TEXT open READ_MODE is dirname&"/ext_input";
file ftype_k        : TEXT open READ_MODE is dirname&"/k_numbers";
file ftype_dmp_SO   : text open WRITE_MODE is dirname&"/SO_dmp.txt";
file ftype_dmp_msg  : text open WRITE_MODE is dirname&"/msg_dmp.txt";

  constant SYS_PERIOD          : time := 10 ns;

  signal clk      : std_logic := '0';
  signal rstn     : std_logic := '1';

  signal tb_eof        : std_logic := '0';
  signal tb_valid      : std_logic := '0';
  signal tb_SO         : std_logic_vector(SO_WIDTH-1 downto 0);
  signal tb_extrinsic  : std_logic_vector(EXT_WIDTH-1 downto 0);
  signal tb_SO_out     : std_logic_vector(SO_WIDTH-1 downto 0);
  signal tb_message    : std_logic_vector(EXT_WIDTH-1 downto 0);
  signal tb_out_valid  : std_logic;
  signal tb_ready      : std_logic;

begin

  clk_gen: process
  begin
    clk <='1';
    wait for SYS_PERIOD/2;
    clk <='0';
    wait for SYS_PERIOD/2;
  end process;

  sys_rstn_prc: process
```

```
  begin
    wait for 5*SYS_PERIOD;
    rstn <= '0';
    wait for 5*SYS_PERIOD;
    rstn <= '1';
    wait;
  end process;

  test_process : process

  variable line_SO       : line;
  variable line_ext      : line;
  variable line_k        : line;
  variable integer_SO    : integer;
  variable integer_ext   : integer;
  variable integer_k     : integer;
  variable integer_layer : integer;

begin
 wait until rstn = '1';
 wait until clk'event and clk = '1';
 readline(ftype_k, line_k);
 read(line_k,integer_layer);
 tb_eof       <= '0'
 for j in 0 to (integer_layer-1) loop
   if j = 4 then
     wait for 20*SYS_PERIOD;
   end if;
   readline(ftype_k, line_k);
   read(line_k,integer_k);
   tb_valid <= '1';
   readline(ftype_SO_in, line_SO);
   read(line_SO,integer_SO);
   tb_SO <= std_logic_vector(to_signed(integer_SO,SO_WIDTH));
   readline(ftype_ext_in, line_ext);
   read(line_ext,integer_ext);
   tb_extrinsic<=std_logic_vector(to_signed(integer_ext,EXT_WIDTH));
   for k in 0 to (integer_k-3) loop
   wait for SYS_PERIOD;
   readline(ftype_SO_in, line_SO);
   read(line_SO,integer_SO);
   tb_SO <= std_logic_vector(to_signed(integer_SO,SO_WIDTH));
   readline(ftype_ext_in, line_ext);
   read(line_ext,integer_ext);
   tb_extrinsic<=std_logic_vector(to_signed(integer_ext,EXT_WIDTH));
     end loop;
   wait for SYS_PERIOD;
   tb_eof <= '1';
   readline(ftype_SO_in, line_SO);
   read(line_SO,integer_SO);
   tb_SO <= std_logic_vector(to_signed(integer_SO,SO_WIDTH));
   readline(ftype_ext_in, line_ext);
   read(line_ext,integer_ext);
   tb_extrinsic<=std_logic_vector(to_signed(integer_ext,EXT_WIDTH));
   wait for SYS_PERIOD;
   tb_valid <= '0';
```

```
    tb_eof <= '0';
    wait for 3*SYS_PERIOD;
  end loop;
  wait;
end process;

  dump_process: process(clk)
      variable line_dumped_SO      : line;
      variable line_dumped_message : line;
    begin
    if(clk'event and  clk ='1') then
      if(tb_out_valid = '1' ) then
        write(line_dumped_SO,to_integer(signed(tb_SO_out)));
        writeline(ftype_dmp_SO,line_dumped_SO);
        write(line_dumped_message,to_integer(signed(tb_message)));
        writeline(ftype_dmp_msg,line_dumped_message);
      end if;
    end if;
    end process;

    dut : functional_unit port map(
          i_clk           => clk,
          i_rstn          => rstn,
          i_eof           => tb_eof,
          i_valid         => tb_valid,
          i_SO            => tb_SO,
          i_extrinsic     => tb_extrinsic,
          o_SO            => tb_SO_out,
          o_message       => tb_message,
          o_valid         => tb_out_valid,
          o_ready         => tb_ready
          );

end architecture;
```

EK E : LDPC Kod Çözücü Üst Seviye VHDL Test Dosyası

```
library ieee;
use ieee.Std_Logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

library ldpc_lib;
use ldpc_lib.functional_unit_comp_pack.all;
use ldpc_lib.LDPC_decoder_top_comp_pack.all;

library common_lib;
use common_lib.ldpc_constants_pack.all;
use common_lib.common_comp_pack.all;

entity tb_top_level is
  generic(
    dirname    : string  := "./testcases/bg_1_inputs"
  );
end entity;

architecture arch of tb_top_level is

file ftype_SO_in  : TEXT open READ_MODE is
dirname&"/top_level_inputs";
file ftype_dmp_hd : text open WRITE_MODE is
dirname&"/hard_decision.txt";

  constant SYS_PERIOD          : time := 10 ns;

  signal clk      : std_logic := '0';
  signal rstn     : std_logic := '1';

  signal tb_eof        : std_logic := '0';
  signal tb_valid      : std_logic := '0';
  signal tb_SO         : std_logic_vector(31 downto 0) :=
  "00000000000000000000000000000000";
  signal tb_data_out   : std_logic_vector(3 downto 0);
  signal tb_out_valid  : std_logic;
  signal tb_ready      : std_logic;
  signal tb_in_ready   : std_logic := '1';
  signal tb_out_eof    : std_logic;

begin

  clk_gen: process
  begin
    clk <='1';
    wait for SYS_PERIOD/2;
    clk <='0';
    wait for SYS_PERIOD/2;
```

```
end process;

-- System Reset generation
sys_rstn_prc: process
begin
  wait for 5*SYS_PERIOD;
  rstn <= '0';
  wait for 5*SYS_PERIOD;
  rstn <= '1';
  wait;
end process;

test_process : process

variable line_SO        : line;
variable line_k         : line;
variable integer_SO     : std_logic_vector(31 downto 0);


begin
  wait until rstn = '1';
  wait until clk'event and clk = '1';
  wait for 2 ns;
  tb_eof       <= '0';
  tb_in_ready  <= '1';
  wait for SYS_PERIOD;
  for j in 0 to (66) loop
    tb_valid     <= '1';
    readline(ftype_SO_in, line_SO);
    read(line_SO,integer_SO);
    tb_SO        <= integer_SO;
    wait for SYS_PERIOD;
  end loop;
  tb_eof       <= '1';
  readline(ftype_SO_in, line_SO);
  read(line_SO,integer_SO);
  tb_SO        <= integer_SO;
  wait for SYS_PERIOD;
  tb_valid     <= '0';
  tb_eof <= '0';
  wait;

end process;

dump_process: process(clk)
    variable line_dumped_hd       : line;
  begin
  if(clk'event and  clk ='1') then
     if(tb_out_valid = '1' ) then
    write(line_dumped_hd,tb_data_out);
    writeline(ftype_dmp_hd,line_dumped_hd);
   end if;
    end if;
  end process;
```

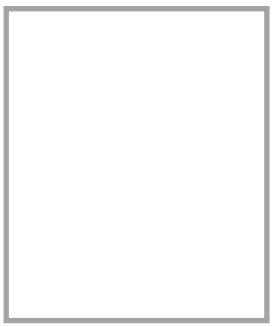
```
    dut :  LDPC_decoder_top port map (
           i_clk        => clk,
           i_rstn       => rstn,

           i_eof        => tb_eof,
           i_valid      => tb_valid,
           i_ready      => tb_in_ready,
           i_data       => tb_SO,

           o_eof        => tb_out_eof,
           o_valid      => tb_out_valid,
           o_ready      => tb_ready,
           o_data       => tb_data_out
           );

end architecture;
```

EK F : LDPC Kod Çözücü Üst Seviye Benzetimi Girişleri

```
00000000000000000000000000000000
00000000000000000000000000000000
00000100111111001111101000000001
11111011000000010000101100001001
11111010111111110000010000000001
00000000111110001111100011111110
11111010000000010000001111111010
00000001000000111111111000000001
11111010111111000000000100000011
11111010111110110000010111111001
11111101000000100000001011111011
11111101111111010000010000000111
00000110111111000000001000000101
00000011000010011111011011110111
00000010111111001111101111111011
11111110111110111111110100001100
00001000111111000000101000000110
00000110111110000000000000000100
11111001000001001111110000000001
11111110000000100000000100000011
00000001111110111111110000000101
11111011111111010000001011111111
11111011000001010000100111111010
11111111111111011000001010000001
11111100000000110000010111111001
11111110111110110000001111111101
11111001000000011111101011111101
00000001111101111111100111110110
11111011000000001111110111111000
11111100000000010000000011111011
00000100111101111111110111110111
11111001000000101111101100000001
11111111000001110000000011110100
00000011111110111111101000001010
00000011000000100000011000000000
00001001111111110000001011111101
00000110000000000000010111111100
00000010111110010000001011111110
00000101111110010000010011111000
00000001000001101111101000001010
00000110000001110000010011111100
00000111000000110000010011111100
00000100000000110000010100001001
11111011000001101111100111111001
00000010111110111111110011110111
00001000111101111111011100000101
11111011000000101111111000000010
11111111000001011111101011111000
11111010000001011111110100000111
00001011000001001111101011111100
```

```
11111100111111011111110100001001
11111100111111010000001111111110
11110111111110111111101100000000
00000001000000011111110100000010
00000010111111010000010111111111
00000110111111111111110000001001
00000101111111101111101100000010
11111101111110100000011100000110
00000001000000001111110011111010
00001000000000101000000000000100
11111110000010100000001111111001
11111100000000100000100011111010
11110110000001000000001000000001
00000100111110101111101100000111
00000100000000101111110111111101
00000110111110111111101111111001
11111100111111000000010111111011
00000101000000100000000011111110
```

ÖZGEÇMİŞ



Adı SOYADI: Barış BİLGİLİ

ÖĞRENİM DURUMU:

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

MESLEKİ DENEYİMLER VE ÖDÜLLER:

- 2020 yılından beri Yonga Teknoloji Mikroelektronik firmasında sayısal tasarım üzerine çalışıyor.

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- B. Bilgili, C. Yamaneren, K. Vatansever, U. Çoltu and B. Örs, "System on Chip Design with Vivado High-Level Synthesis Tool," 2019 11th International Conference on Electrical and Electronics Engineering (ELECO), 2019, pp. 1047-1050, doi: 10.23919/ELECO47770.2019.8990595.