



MILLİ
TEKNOLOJİ
HAMLESİ



ÇİP TASARIM YARIŞMASI
SAYISAL İŞLEMCI TASARIM KATEGORİSİ
ÖN TASARIM RAPORU

TAKIM ADI: Riskbeş

BAŞVURU ID: #1414559

2024

1. Giriş

Bu projede RV32IMAFBC_Zicsr komut setini destekleyen bir işlemci çekirdeği tasarlanması planlanıyor, çekirdeğimiz yüksek bir verime sahip olabilmesi için boru hattı mekanizması kullanılacaktır. Çekirdeğe bağlı biri buyruk biri veri için olmak üzere 2 tane 2KB boyutunda önbellek bulunacaktır. Bu önbellekler kendi içinde bloklara ayrılacaktır. Çekirdeğimiz dış dünya ile haberleşebilmesi için Wishbone arayüzü ve buna bağlı bir UART birimi eklenecektir. Çekirdeğin Tasarımı “Computer Organization and Design RISC-V Edition: The Hardware Software Interface”[\[1\]](#) kitabından ilham almıştır.

Raporun Tasarım isterleri bölümünde mevcut sistemin özelliklerinden ve onun üstüne tasarlanacak nihai tasarımın özelliklerinden bahsedildi, tahmini performans değerleri açıklandı ve en son çekirdeğimizi nasıl doğrulayacağımızı anlattık.

Tasarım Detaylarında ise tasarım detaylı bir şekilde anlatıldı, her birim işlevi ve tasarımını anlatıldı, hangi yöntem ne için seçildiği, diğer yöntemlere göre avantaj/dezavantaj durumlarından bahsedildi.

2. Tasarım İsterleri

2.1. Mevcut Tasarım

Mevcut tasarım, RV32IMC komut setini destekliyor.

5 aşamalı boru hattı kullanıyor. Boru hattının aşama sayısının fazla olması saat vuruş sıklığını düşürülmesini sağlar. Ancak bu sayı arttıkça, veri bağımlılığı ve dallanma gibi sebeplerden dolayı boru hattının yönetimi de zorlaşmaktadır. Bu nedenle **saat vuruş sıklığı / boru hattı karmaşıklığı** ikilemi arasında bu sayıyı seçtik[\[2\]](#).

Buyruk ve veri belleklerinin tamamen ayrıldığı saf Harvard mimarisine sahip.

C uzantısı dolayısıyla, toplam kod uzunluğu %25-%30 civarında düşer. Bunun da buyruk belleği miktarının %50 civarında bir artışla aynı seviyede avantajda olduğu tespit edilmiştir.[\[3\]](#)

Ancak C uzantısı bulunması dolayısıyla, işlemci hem 16 bit hem de 32 bit buyrukları destekliyor. 32 bit buyruklar hizalı olmak zorunda değil. Buyruk belleğine hizasız 32 bit erişim karmaşıklığı fazla arttırır. Bunun yerine buyruk belleğine her zaman hizalı 32 bit erişim yapılmıyor. Buyrukları düzungen getirebilmek için 3 durumlu bir durum makinesi ve bir kayıt yazmacı kullanılıyor. Bunun oluşturduğu alan artışı, bellek kullanımındaki kazançla karşılaştırıldığında oldukça küçüktür.

2.2. Yapılacak Geliştirmeler

Buyruk ve veri bellekleri yerine, buyruk ve veri önbellekleri konulacak; bu iki önbellek tek bir ana belleğe bağlı olacaktır. Bu şekilde saf Harvard mimarisinden, modifiye Harvard miarasine geçilecektir. [4]

Buyruk kümesine A, F, B ve Zicsr uzantıları eklenecektir.

İşlemciye bir UART birimi eklenecektir. Eklenecek UART birimiyle Wishbone arayüzü üzerinden iletişim kurulacaktır.

2.3. Yapılacak Testler

Çekirdeğimizi ilk önce bubble sort gibi basit algoritmalar ile test etmeyi planlıyoruz, daha sonra riscv'in formal testlerinden olan riscv-arch-test'ten[5] geçirmeyi planlıyoruz.

2.4. Performans Analizi

Tasarımımız boru hattına sahip olduğu için, kritik yol kısa olup 75MHz de çalıştırılmaya planlıyoruz. Bu işlemcinin her 13.3 ns'de bir buyruk tamamlaması anlamına gelir.

Not: Yanlış dallanmalar ve benzeri bazı boru hattı sorunlarından dolayı ortalama buyruk süresi artabilir. Yukarıdaki sayı en iyi durumu almaktadır.

Çekirdek çevrimlerin bir kısmını önbellek güncellenirken beklemek zorunda kalacaktır. Eğer bir önbellek güncellemesinin 25 çevrim süրdüğünü, ve ortalama da 200 çevrim de bir önbellek güncellemesi gerektiğini varsayırsak; bu çekirdeğin çevrimlerinin 1/8'inin kullanılmadığı anlamına gelir. Bu da saat periyodunun 65,625MHz inmesi, yani her buyruğun 15.23 ns sürmesiyle aynı durumdur.

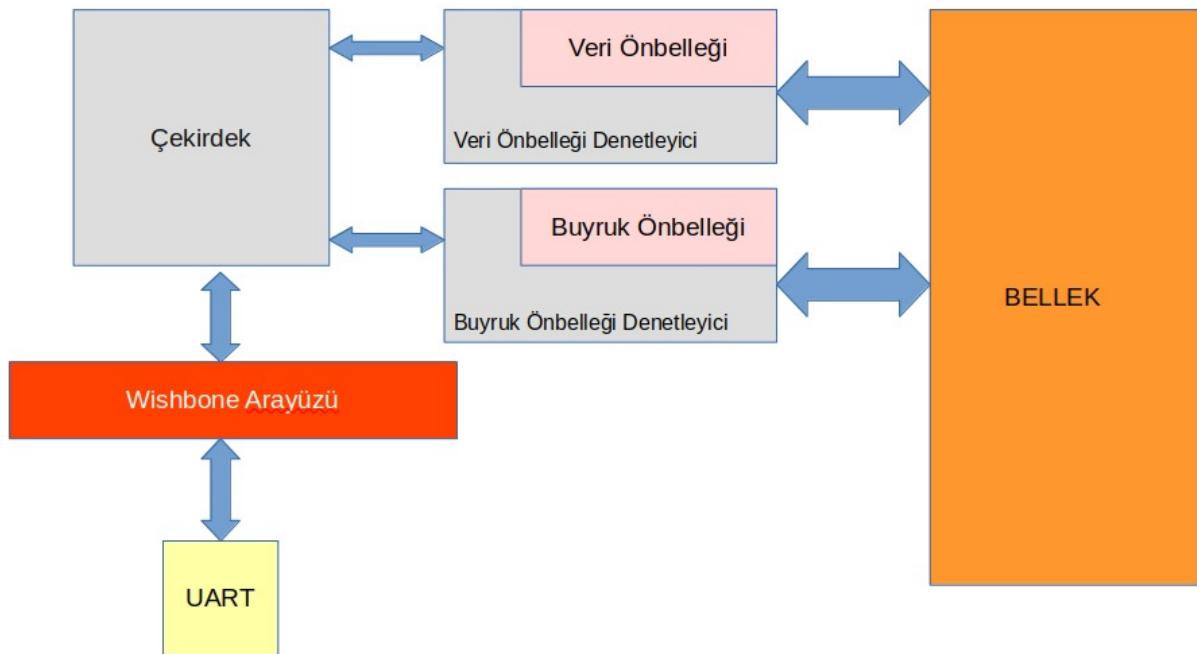
Not: Her önbellek güncellemesi sırasında çekirdeğin güncellenmesi gerekli olmayabilir. Böyle durumlarda sırasız yürütüm gibi optimizasyonlar kullanılması mümkündür. Bu tür optimizasyonların kullanılması planlanmamaktadır.

Boru hattının verimini (Throughput) hesaplamak için $Verim = (Aşama Sayısı + Buyruk sayısı - 1) * Çevrim periodu$ formülünü kullanırsak. ve Yukarı da belirlediğimiz saat frekansını baz alırsak N buyruklu bir programın çalışma süresi $(5 + N - 1) * 15.23 ns = 15.23N + 76.15 ns$ olmaktadır. N'in yüksek bir sayı olması durumunda bunu 15.23N ns'e yuvarlayabiliriz. Örneğin 1 milyon buyruklu bir program yaklaşık 15 ms sürecektir.

Not: Neredeyse bütün programlar koşullu dallanma buyrukları içерdiği için, programdaki toplam buyruk sayısını önceden hesaplamak zordur.

Bütün bunlara bağlı olarak, işlemcinin saniyede yaklaşık 6,7 milyon buyruk tamamlaması beklenir.

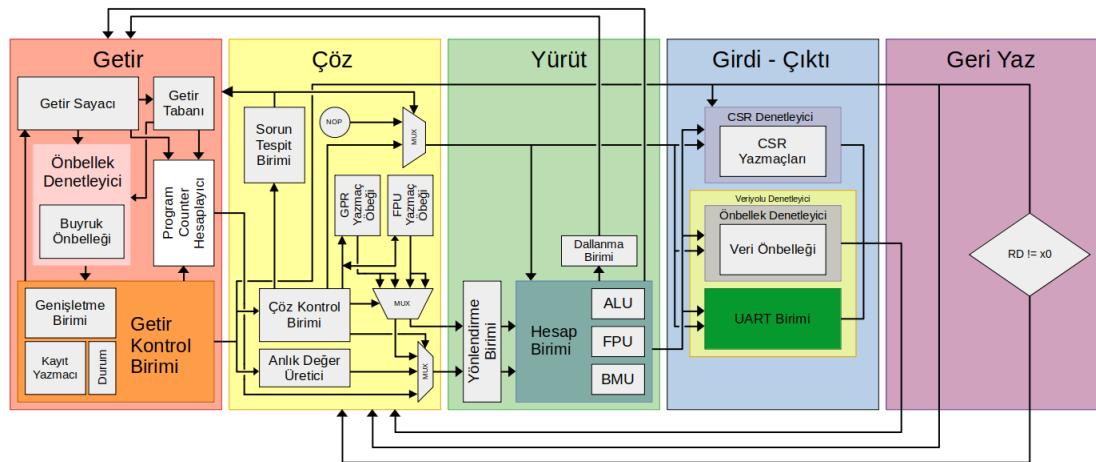
3. Tasarım Detayları



Şekil 1: Sistemin Blok Diyagramı

Şekil 1'de sistemin genel yapısı gösterilmiştir.

3.1. Boru Hattı



Şekil 2: Boru Hattı Tasarımı

Şekil 2'deki diyagramda tasarlardığımız boru hattının aşama ve birimleri gösterilmiştir. Her yeni buyruk getir aşamasında başlar, ve geri yaz aşamasında tamamlanır. Belirtilemesi gereklidir ki, önbellek denetleyiciler ve UART birimi çekirdeğin parçası değildir, ancak okunurluğu artırmak için diyagrama eklenmiştir.

3.1.1. Getir aşaması: Getir aşaması buyruk önbelleği denetleyicisiyle iletişim kurar. Getir Sayacı ismini verdigimiz yazmaç buyruk önbelleginde erişilen yeri gösterir. Getir Tabanı ismini verdigimiz

yazmaç ise kullanılan buyruk önbelleği bloğunun ana bellekteki yerini gösterir. Bu yazmaç değiştirse, önbellekte yeni bir bloğa geçilmesi veya önbelleğin güncellenmesi gerekir.

Buyruk Önbelleği Denetleyici: Buyruk önbelleğiyle iletişim sağlar. Detaylı bilgi Bölüm 3.2'dedir.

Getir Kontrol Birimi: Bu birim kendi içinde bir genişletme birimi, durum makinesi ve bir kayıt yazmacı içerir. Kullandığımız komut setinde 16 bitlik ve 32 bitlik olmak üzere iki tür buyruk vardır. Genişletme birimi 16 bitlik buyrukları 32 bitlik karşıtlarıyla değiştirir. Buyruk uzunluğunun belirsiz olması sonucu, 32 bitlik buyrukların hizasız olma ihtimali vardır. Ancak buyruk önbelleği sadece hizalı erişimleri destekler. Bu yüzden buyrukların düzgün okunabilmesi için 3 durumlu ve bir kayıt yazmacı bulunan bir durum makinesi kullanılır.

Kayıt yazmacı her çevrimde önbellekten okunan verinin üst 14 bitini kaydeder. Bu 14 bit, geciktirilmiş okuma durumunda kullanılır.

Durum makinesinin üç durumunun adları sırasıyla: Hizalı okuma, üst yarı okumadır ve geciktirilmiş okumadır. Hizalı okuma PC yazmacının 4'ün katı olduğu durumdur. Üst yarı okuma ise, PC yazmacının 4'ün katı olmadığı, 16 bitlik buyruk okuduğumuz durumdur. Bu durumlarda buyruk rahatlıkla okunabilir. Geciktirilmiş okuma, PC yazmacının 4'ün katı olmadığı, 32 bitlik buyruk okuduğumuz durumdur. Bu durumda, bu çevrimde okunan alt 16 bit kayıt yazmacındaki 14 bit ile birleştirilir. Reset sinyali sonrası ilk durum hizalı okumadır.

Dallanma olmadığı sürece, yukarıdaki durum makinesi her çevrimde bir buyruk okur. Ancak dallanma durumunda hizasız 32 bit buyrukları okumak için iki çevrim gerekmektedir. Bu durumda bir çevrim için NOP buyruğu getirilir.

Program Counter Hesaplayıcı: Şu anki Getir Tabanı, Getir Sayacı ve Getir Kontrol Birimi'nin durumuna göre PC değerini hesaplar.

3.1.2. Çöz aşaması: Bu aşamada kullanılacak yazmaçlar okunur, anlık değerler üretılır, buyruk için gerekli denetim sinyalleri üretilir. Yazmaçlar, anlık değerler veya PC'den hangileri buyruk için gerekliyse bir sonraki aşamaya gönderilir.

Çöz Kontrol Birimi: Denetim sinyallerin üretildiği birim.

Yazmaç Öbekleri: Yazmaç değerlerinin saklandığı öbekler. Genel amaçlı yazmaçlar ve FPU yazmaçları ayrı öbeklerde tutulur. Her iki öbek de, çevrim başına iki okuma işlemi ve bir yazma işlemi yapabilir.

Anlık Değer Üretici: Sabit sayı içeren komutlardan sayımı çekip 32 bit uzunluğa genişleten birim.

Sorun Tespit Birimi: Eğer bir önceki buyrukta girdi işlemi varsa, ve bu buyruk bu işlemin sonucunu kullanırsa 1 çevrim kadar boşluk oluşturmak gereklidir. Bu yapılmazsa Girdi-Çıktı aşamasında veri yüklenirken, bu buyruk EX aşamasına varmış olur. Veri yönlendirme birimi bunu çözemediği için araya 1 NOP buyruğu eklenmesi gereklidir.

3.1.3. Yürüt Aşaması: Aritmetik-Lojik İşlemlerin gerçekleştiği aşamadır, hafıza işlemleri için gereken adres bu aşama da hesaplanmaktadır. Ayrıca dallanma buyruklarının alınıp alınmayacağı da burada hesaplanır.

ALU: Matematiksel, Lojik, çarpma ve bölme işlemlerinin gerçekleştiği birim.

Not: Çarpma ve bölme işlemleri şu an ALU'nun içinde olsa dahi, bunların ayrı bir birime taşınması planlanmaktadır. Bu birimde yapılan işlemler birden fazla çevrim alabilecek, çarpma ve bölme işlemlerini yavaşlatmasını karşın; çevrim süresinin düşük tutulması ile, diğer işlemler hızlandırılacaktır.

FPU: Kayan nokta işlemlerinin gerçekleştiği birim.

BMU: B uzantısındaki buyrukların sonuçlarının hesaplandığı birim.

Dallanma Birimi: Dallanma komutlarında dallanmanın gerçekleşip veya gerçekleşmediği belirleyen birim.

Yönlendirme Birimi: Bu ve bundan sonraki aşamalarındaki sonuç yazmaclarını ve "yazmaca yaz" sinyallerini alır. Eğer bu aşamalardaki buyruk, sonuç yazmacına bir değer yazacaksa; çöz aşamasından gelen değerler yerine bu değer matematiksel birimlerin ilgili girdilerine yönlendirilir.

3.1.4. Girdi - Çıktı Aşaması: Bu aşamada veri önbelleğine, UART birimi yazmaclarına veya CSR'lere veri yazılır veya onlardan veri okunur.

CSR Denetleyici: CSR'lere veri yazar ve onlardan veri okur. CSR'lere veri yazma sırasında yan etki oluşturma ihtimali vardır, bunu kontrol etmek için RS1 değeri incelenecaktır. Veri okuma sırasında etki oluşturma ihtimali standart CSR'lerde bulunmadığı için, tasarımda bunun hiçbir zaman olamayacağı var sayılmıştır. CSR'lerin buyruk değişimine etkisi olabildiğince bu ve Geri Yaz aşamasına kısıtlı olacaktır. Bunun mümkün olmadığı durumlarda bekleme yapılacaktır.

UART birimi: Bu birim ile iletişim için Wishbone arayüzü kullanılacaktır[6]. Boru hattının IO aşamasında 0x80000000'in altında olan adreslere yapılan okuma-yazma işlemleri UART birimine gönderilecektir. UART birimi yonga dışıyla çekirdek arasında duracak ve iletişimini sağlayacaktır.

Veriyolu Denetleyici: Hafıza işlemlerini doğru birimlere yönlendirecek. Ayrıca A uzantısı buyrukları için başka birimlerin de hafıza işlemi yapıp yapmadığını kontrol edecek.

Veri Önbelleği Denetleyici: Veri önbelleğiyle iletişimini sağlar. Detaylı bilgi Bölüm 3.2'dedir.

3.1.5. Geri Yaz Aşaması: Buyrukta yapılan işlemin sonucu yazmaca öbeklerine geri yazılır.

3.1.6. Boru Hattı Sorunları Çözümleri:

Kaynak Sorunu: Modifiye Harvard mimarisi kullandığımız için buyruk ve veri önbellekleri birbirinden ayrıdır ve kaynak sorunu oluşturmamaktadır. Ana belleğe iki önbellek denetleyicisinin aynı anda erişmesi engellenmiştir.

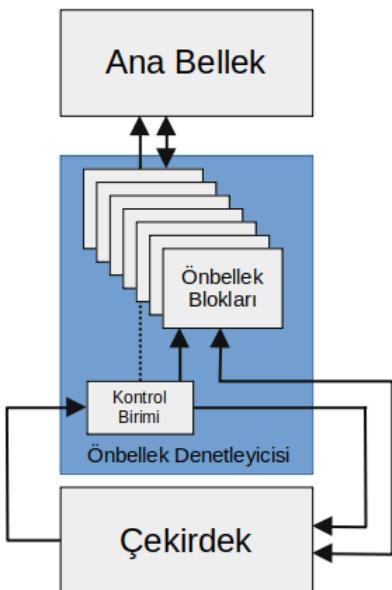
Veri Bağımlılığı Sorunu: Buyuklar arası oluşabilecek veri bağımlılıkları sorunları Yönlendirme Birimi ve Sorun Tespit Birimi ile çözüldü. Tek çevrimde aynı yazmaca okuma ve yazma sorununu çözmek için ise, saatin yükselen kenarda yazmaca yazılırken, düşen kenarda ise yazmacıları okunuyor.

Denetim Sorunu: Her zaman dallanmaz yaklaşımını kullanıldı. Herhangi bir dallanma olduğu takdirde Getir ve Çöz aşamalarındaki komutlar temizlenir.

3.2. Önbellekler

Tasarımızda biri buyruk için, biri veri için olmak üzere iki önbellek bulunmaktadır. Bu önbellekler önbellek denetleyicilerinin içinde bulunur ve bloklara ayrılmışlardır. Buyruk ve veri önbelleğinin ikisi de 128 byte uzunlığında 16 bloktan oluşmaktadır, ve bu şekilde 2KB buyruk ve 2KB veri önbelleği bulunur[7][8].

Not: Önbellekler için uygun blok boyut ve sayısını bulmak için test yapılacaktır. Bu testin sonucunda yukarıda verilen rakamlar değişimelidir.



Şekil 3: Önbellek Denetleyicisi

Kullanılan önbellek denetleyicilerinin yapısı Şekil 2'de gösterilmiştir. Her iki önbellek denetleyicisinin içinde birer kontrol birimi vardır.

Kontrol birimi önbellek bloklarını denetler ve gereğinde ana bellekten günceller. Önbellek blokları ana belleğe doğrudan eşleştirilmiştir (direct mapped cache).

Çekirdekten herhangi bir bellek işlemi geldiğinde; kontrol birimi gerekli bloğa doğru verinin yüklü olup olmadığını kontrol eder. Eğer doğru veri yüklüse bu blok üzerinde istenen işlem yapılır. Eğer değilse; çekirdek bir süreliğine durdurular, o blokta bulunan veri ana belleğe kaydedilir, yeni veri bu bloğa okunur ve ondan sonra istenen işlem yapılp çekirdek devam ettirilir.

Not: Yazma işlemlerinin blok yüklenmeden yapılması mantıken mümkündür. Blok yüklenirken yazma yapılmış olan adresler değiştirilmez ve bu şekilde kullanılır. Bu optimizasyon şu anki tasarımda bulunmamaktadır.

3.3. Sentez ve Serim

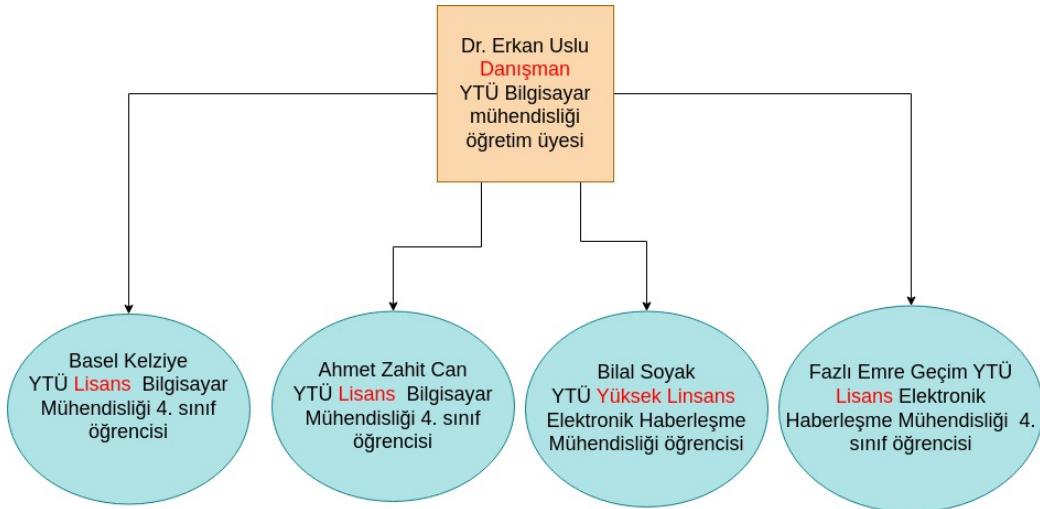
Günümüzde sayısal bir çip tasarlayabilmek ve tasarlanan çipin üretime uygun bir çıktısını alabilmek için bir dizi aşamayı tamamlamak gereklidir. Bu aşamaların her birini tamamlayabilmek için açık kaynaklı araçlar mevcuttur. Bu araçların birçoğunu bünyesinde bulunduran OpenLane [9] çip tasarım akışında kullanılacak önemli bir projedir. OpenLane, VLSI tasarım akışını sağlayan ve birçok açık kaynaklı aracın bir araya gelmesiyle oluşturulmuş bir projedir. OpenLane içerisinde, entegre devrelerin (IC) tasarım süreci için çeşitli adımlar ve araçlar bulunur. Çip tasarımının ilk aşaması olan RTL sentez aşamasında OpenLane Yosys kullanılır. Sentez, yazılmış olan HDL tasarım kodlarının okunarak bir RTL devre şemasına dönüştürüldüğü bir aşamadır. Bu aşamada veri yolu ve kontrol yolu hataları ortaya çıkabilir. Sentez kısmında bu sorunları çözmek için simülasyon, optimizasyon araçları kullanılır.

Ardından OpenSTA aracı ile kapı seviyesinde Statik Zamanlama doğrulaması yapılır. Tasarım isterlerinin karşılandığından emin olunduktan sonra yerlesim (Placement) aşamasına geçilir. Bu aşamada tasarımın yerlesimi Floorplan, TritonMacroPlacer, RePLACE araçları kullanılarak yapılır. Yerlesim (Placement) aşamasında bileşenlerin yerlesimi, sinyal ve güç dağıtımını önemlidir. Verimsiz bir yerlesim gerekliliği alanları ve güç tüketimini artırabilir. Alan kullanımını ve güç tüketimini verimli hale getirmek için yerlesim aşamasında yolların uzunluğu mümkün oldukça kısa tutulmaya çalışılır. Yolları kısa tutmak fiziksel kısıtlamalar getirebilir. Tasarımın güç gereksinimlerini ve saat sinyallerinin dağıtımını sağlamak için PDN ve TritonCTS araçları kullanılır. Bu aşamalardan sonra tasarımın bağlantılarının en uygun şekilde yönlendirilmesi için Global Routing FastRoute programı kullanılır. Global yönlendirmeden sonra tasarım mantıksal eşlik testi ve detaylı yönlendirmeye tabi tutulur. Mantıksal eşlik testi için Yosys tarafından gerçekleştirilen global yönlendirme çıktısı, doğrulama sürecinden geçer. Ardından, detaylı yönlendirme işlemi TritonRoute tarafından yapılır. Elde edilen çıktı, optimize edilmesi gereken noktalar varsa, tekrar akış döngüsüne alınabilir. Aksi takdirde, tasarımın geri kalan kısımları, elde edilen fiziksel çıktıının doğrulanması üzerine odaklanır. Tasarımın doğruluğunu kontrol etmek için DRC, LVS, XOR eşleme ve Anten testleri gerçekleştirilir. Son olarak,

tasarım akışının tamamlanmasıyla elde edilen çıktı, GDSII dosyasına dönüştürülerek sona erer. Bu dosya, entegre devrelerin üretiminde kullanılan bir standart dosya formatıdır. GDSII dosyası ile HDL tasarım kodları silikon üzerine aktarılıp gerçekleşir.

4. Takım Organizasyonu ve İş Planı

Sekil 4'de takımın üyeleri ve eğitim bilgileri görülmektedir.



Sekil 4: Takım üyeleri

Sekil 5'de ise, gantt görev-zaman diagramı görülmektedir.



Sekil 5: Proje için Gantt diyagramı

Kaynakça

- [1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann, 2017.
- [2] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," Proceedings 29th Annual International Symposium on Computer Architecture, Anchorage, AK, USA, 2002, pp. 25-34, doi: 10.1109/ISCA.2002.1003559.
- [3] RISC-V Foundation, "The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA, Version 2.2," May 2017. [Online]. Available: <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>
- [4] Wikipedia, "Harvard architecture," Wikipedia, The Free Encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Harvard_architecture
- [5] *RISC-V Architecture Test SIG*, RISC-V Foundation [Online]. Available: <https://github.com/riscv-non-isa/riscv-arch-test>
- [6] Wishbone Interconnect. "Introduction," Read the Docs. [Online]. Available: https://wishbone-interconnect.readthedocs.io/en/latest/01_introduction.html
- [7] Redis, "Cache Memory," Redis Labs Glossary. [Online]. Available: <https://redis.com/glossary/cache-memory/>
- [8] Alan Jay Smith. 1982. Cache Memories. ACM Comput. Surv. 14, 3 (Sept. 1982), 473–530. <https://doi.org/10.1145/356887.356892>
- [9] The OpenROAD Project, "OpenLane," GitHub. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenLane>